A Decision Support System for Logistics Operations

María D. R-Moreno, David Camacho, David F. Barrero and Miguel Gutierrez

Abstract This paper describes an Artificial Intelligence based application for a logistic company that solves the problem of grouping by zones the packages that have to be delivered and propose the routes that the drivers should follow. The tool combines from the one hand, Case-Based Reasoning techniques to separate and learn the most frequent areas or zones that the experienced logistic operators do. These techniques allow the company to separate the daily incidents that generate noise in the routes, from the decision made based on the knowledge of the route. From the other hand, we have used Evolutionary Computation to plan optimal routes from the learning areas and evaluate those routes. The application allows the users to decide under what parameters (i.e. distance, time, etc) the route should be optimized.

1 Introduction

The actual demand of precise and trustable information in the logistic sector has driven the development of complex Geographic Information Systems (GIS) very useful for planning their routes. Those systems are combined with Global Position Systems (GPS) for positioning the different elements involved in the shipping. However, those systems although very useful, cannot take decisions based on, for example shortcuts in the route that human operators learn from the experience.

David F. Barrero, María D. R-Moreno
Departmento de Automática. Universidad de Alcalá, Madrid, Spain
e-mail: (david|mdolores)@aut.uah.es

David Camacho
Departamento de Informática. Universidad Autónoma de Madrid, Madrid, Spain
e-mail: david.camacho@uam.es

Miguel Gutierrez
Espi & Le Barbier.
e-mail: miguel.gutierrez@espilebarbier.com

In our particular problem, the logistic company has a set of logistic distributors with General Packet Radio Service (GPRS) connexions and Personal Digital Assistants (PDAs) (around 1200 in the whole Spanish geography) where the list of shippings is stored for each day. The order in which they should be carried out is decided by the distributor depending on the situation: density of the traffic, state of the highway, hour of the day and place of the shipping. For example, it is better to ship in industrial areas early in the morning if the traffic is less dense. The list of tasks for each distributor is supplied each day from a central server.

Within this context, the company needs a decision support tool that, from the one hand allows them to decide the best drivers behaviors and learn from them. And from the other hand, to plan the routes and evaluate and compared them under different parameters.

There are several available tools that address (partially) the described problem. For example, the ILOG SOLVER provides scheduling solutions in some domains such as manufacturing or transportation and the ILOG DISPATCHER provides extensions for vehicle routing. STRATOVISION is a Decision Support and Modeling System for Logistics Strategy Planning that allows the user to represent a scenario and interact with it in an easy way. AIMSUN allows evaluating different transportation solutions. But, none of these tools combine what the experience operators do (learning from their decisions), and plan optimal routes from the learnt knowledge.

This paper presents a tool that combines Case-Based Reasoning (CBR) to learn humans decisions and Evolutionary Computation (EC) to plan for optimal routes. The structure of the paper is as follows. Section 2 presents the different subsystems that the application is subdivided into. Next, section 3 describes in detail the CBR architecture and the algorithms used. After, the GA and the parameters used for the route optimization are introduced in section 4. Then, section 5 shows an experimental evaluation of the application with the real data provided by the logistic company. Finally conclusions and future work are outlined.

2 Application Architecture

The application is subdivided in four subsystems:

- The Data Processing Subsystem: takes the data from the files provided by the logistic company in CSV format (Comma Separated Values). It analyses that the files are valid, loads the information in data structures and performs a first statistic analysis of each of the drivers contained in the file such as the number of working days, number of physical acts, average of physical acts performed, or average of the type of confirmation received by the client.
- The Loading Data Subsystem: is in charge of obtaining and loading the geographic information (latitude and longitude) of each address and the distances among addresses. This information is provided by mean of the Google Maps API. During the process of calculating the coordinates, we group the same addresses (a driver can ship several packages in the same address) for a given date

and a driver into one. We name that *act*. So in the database, the addresses will be loaded as *acts* and an extra field will be added to represent the number of deliveries and pick ups for that act.

- The Learning Working Areas (LWA) Subsystem: creates zones or working areas for each driver using the data loaded and processed in the previous subsystems. To generate that subdivision we have used the zip code as the baseline. So, a zone or a working area can be represented by a zip code or more than one. It also allows us to visualize (using the Google Maps API) the different working areas.
- The Learning Manager Task (LMT) Subsystem: plans for routes in a date selected by the user. We can specify the number of different plans we want as output and compare them with the original plan performed by the driver. In the comparison and the generation of plans, we can use different parameters to measure the quality of the plans such as the total distance, positive rate of LWA, time, etc. The planning algorithm can adapt its answer to the driver behavior and generate plans according to it, if that is what we want.

3 Case Based Reasoning Architecture

Cased Based Reasoning (CBR) [1] is an Artificial Intelligence (AI) technique that solves new problems based on the acquired knowledge about how similar problems were solved in the past. Any CBR system can be defined using two main elements: the *cases* or concepts which are used to represent the problem, and the knowledge base that is used to store and retrieve the cases. To handle this knowledge the CBR systems needs to define several processes to *Retrieve* the most similar case(s) in the knowledge base, *Reuse* the selected case using its solution adapted to the current problem, *Revise* the adapted solution to verify if it can solve the actual case (if not the process starts again), and finally, *Retain* the solution if it is satisfactory.

In our case, the knowledge base is extracted from the drivers experience and later it is reused for optimization and logistic issues. This knowledge is given by the files provided by the company (see the Data Processing Subsystem section).

The information loaded in the database can be used it straightforward (i.e. address and time delivery can be used to estimate how much time is necessary to complete a particular ship), or it can be used to indirectly learn from the experience drivers. The drivers know which areas have thick traffic so they can take alternative paths to reduce time and gas consumption. During the working day, the driver may stop in places not related to the shipping to rest or eat. So this information has also some disadvantages, mainly caused by the deficiency and irregularity of the data given by the company. The information has a lot of noise: there are many mistakes in the zip codes and the street names that are hard to correct. The shipping time that could be used to calculate the time between some points in the path, it is not reliable since sometimes the drivers annotate the hour after they have done some deliveries.

So, our CBR algorithm uses this information in a simplified way to prevent that the noisy information could affect to the optimization process. The generation of

the final CBR information, such as the working areas of the drivers, are carried out using statistical considerations (average of behaviours followed by all the drivers analysed) to minimize the noise.

Then, the following step in our CBR algorithm is to group the shippings. We can define the concept of the *working zone* or *working area* (WA). It represents a particular zone in a city, or in a country, where one or several drivers will work shipping objects (as mentioned before, we have called them **acts**). Usually the logistic company define (using a human expert) these WA and assign them to a set of drivers. Minimizing the overlapping of these areas among different drivers is essential to minimize the number of drivers and the deliver time for a set of acts. The automatic generation of these working areas is the target of our CBR algorithm, these WA will be used later in the optimization process.

The algorithm starts defining a grid that represents all the available postal codes ($Z_i, i \in [1..n]$) for a particular city (or county). These postal codes are used to fix a geographical centroid so we can calculate (using any standard algorithm, i.e. based on GPS values) the distance between two postal codes Z_i and Z_j ($distZ_i, Z_j$). On the other hand, the information from driver's log is used to calculate how many acts belongs to the same postal code, and how many acts occurs between adjacents postal codes (a driver could work in a particular postal code, or could work in several postal codes). This information is represented using a parameter s_i which is calculated as $s_i = \sum(acts(Z_i \rightarrow Z_j) + acts(Z_j \rightarrow Z_i))$, this parameter represents the jumps between two postal codes (we can expect that if several acts are interleaved, and they are placed in different postal codes, these should be enough closer to maintain the shipping costs low).

Let us now consider a particular driver's log as a list of m ordered acts ($act_k, k \in [1..m]$). A new value $\theta = \sum(d_k/s_k)$, is calculated for each log, where
$d_k = dist(act_k(Z_k), act_{k+1}(Z_{k+1}))$.

Using the postal codes stored in each log we generate a set of LWA using a clustering algorithm based on the value of θ and δ [1]. The algorithm works as follows, using the drivers available (and the predefined value of δ) all the acts are grouped into a set of clusters, then these clusters are compared among drivers: If a particular cluster is detected in different drivers, it is given to the system as a new learned working area. The quality of the learned clusters will depend on the number of drivers that has this cluster, so the system can take different actions in the planning process taking into account how good this cluster is. Currently, we consider that any learned cluster that belongs at least to three different drivers has enough quality to be used directly by the system. This could be modified in the near future and allow the user to modify the planning process using a reliability factor of these learned clusters.

Finally, each LWA learned is stored as a new case in the data base, for each driver the algorithm is applied and the LWA zones are generated.

[1] The value of δ was obtained from an empirical evaluation of several drivers and working days randomly selected. This value was selected analysing the LWA_1 generated in the first execution of the algorithm.

4 Route Optimization

One of the main features of the proposed system is its capacity to suggest efficient routes. The term eficient in this context should be interpreted as relative to a set of routes designed by a human expert and provided to the system as input. So, the goal of the described system is to improve a given set of routes rather than generate complete new routes. This characteristic is used by the optimization engine to guide its search. The definition of the zones is provided by the CBR described in the previous section so it doesn't have to handle this task and thus, the route optimization can be considered as a variation of the Travel Salesman Problem (TSP). The selection of the optimization algorithm is critical to the success of the system. Evolutionary Computation (EC) [3] provides a set of tools that can be used within this scenario.

EC is a collection of algorithms inspired in the biological evolution. Regardless of the flavour of the specific EC technique, they share three characteristics, (1) they use a population of individuals that represent each one a solution in the search space, (2) individuals are modified using a genetic operator, and (3), individuals are under a selective pressure. The result is a evolution of the population that would eventually converge to a global solution. From an AI point of view, EC is a set of stochastic search algorithms. Depending on how individuals are represented and, how they are modified, we can find several algorithms, one of the most sucessful ones are Genetic Algorithms (GA) [4].

The GA implemented is based on the work of Sengoku described in [5]. This GA encodes the solution using a classical permutation codification [2], where the acts are coded as integer numbers between one and the number of acts in a fixed length chromosome. Since no act can be visited twice, no integer in the chromosome is allowed to be repeated. Indeed the valuable information is not the presence of the integer but rather information is kept in the adjacency. In this way, the chromosome $A_1 = \{4,3,2,5,6,1\}$ represents the path $4 \rightarrow 3 \rightarrow 2 \rightarrow 5 \rightarrow 6 \rightarrow 1$, and it is equivalent to another chromosome $A_2 = \{5,6,1,4,3,2\}$ representing $5 \rightarrow 6 \rightarrow 1 \rightarrow 4 \rightarrow 3 \rightarrow 2$. The route that they code is the same because the genes have the same adjacency.

GAs require a mechanism to evaluate the quality of individuals, or fitness. This is a key subject in any GA design that may determine its sucess or failure. The fitness function has a close relationship with the value that the solution provides to the user. When optimizing routes, the fitness that an individual scores provide a refference of time or distance savings. In our case, we have used an aggregate scalar fitness function which evaluates the individual based on different characteristics.

A human made route is given to the GA as reference as well as a classification of act in zones. The fitness evaluation is done comparing the solution to this reference route. Of course, a route can outperform or not another route depending on the criteria that is applied. Our system uses four criteria or qualifications: Time, Distance, Zone and Act.

Using these qualifications we can evaluate different aspects related to the route, such as time or distance. In order to provide a syntetic estimation of the quality

of the individual, these qualifications are aggregated in a linear combination that conforms the fitness function, as is expressed in equation 1.

$$f(A) = 0.4\omega_d\Delta_d(A) + 0.4\omega_t\Delta_t(A) + \omega_z\Delta_z(A) + \omega_a\Delta_a(A) \quad (1)$$

where ω_i are coeffients associated to distance (ω_d), time (ω_t), zones (ω_z) and acts (ω_a). These coefficients are used to weight the contribution of each category to the fitness. It is possible to change the priority of qualifications that the user prefer to optimize just modifing the coefficients ω_i. The function $\Delta_i(A)$ returns the relative difference between the route codified in the chromosome A and the reference route for each one of the described categories. By default, ω_d and ω_t are set to 0.35 while ω_z and ω_a are both set to 0.15.

The TSP is a problem where it is not possible to know when a global maximum is achieved, so a convergence criteria is required to stop the execution of the GA. In this case, the GA is suppossed to have converged if the average fitness in generation i is less than 1% better than the fitness in generation $i-1$. The initial population is generated randomly avoiding repeated individuals.

Our GA uses the same evolution strategy than [5]. Given a population M of routes in generation i, the N best routes are cloned. In an attempt to avoid a premature convergence due to the loose of genetic diversity, the routes are sorted by their fitness value and the adjacent one are compared. Those routes whose fitness have a difference less than ε are removed. To maintain constant the number of individuals in each generation, $M-N$ individuals are generated by means of a Greedy Subtour Crossover [5] where the parents are radomly selected. Then a mutation operator called 2opt is applied with a probability p_c. This operator swaps two random points in the route if and only if the new individual is fitter than the old one. In case that 2opt did not generate a fitter individual it is not modified.

5 Experimental Results

One of the main problems we encountered in testing was the high percentage of mistakes in the addresses in the input files. Without any pre-processing the number of errors in the streets or zip codes is around 35%. After some preprocessing (such as using the address to update incorrect zip codes, or eliminating or adding some characters and searching again) the percentage drops to 20%. This is still very high when attempting to perform an automatic comparison of the routes followed by the drivers and the ones generated by our tool.

In our first attempt to compare the results, we manually cleaned the input files of 10 drivers during one month. The drivers choosen average 25 to 40 acts each day. The improvement obtained on average by our tool is 26% if we use the distance as the comparison parameter and 20% if we use the time.

But these results require of some explanations using a *typical* driver of the set analised previously. By typical we mean that his behaviour is normal and the route

generated does not contain too much noise (i.e. a noisy route would perform in 30 mn half of the shippings).

The comparison of the results is carried out on the fitness of the best individual obtained after the execution of the algorithm compared to the fitness of the original itinerary on the acts of one day. Table 1 shows the values of the fitness in both itenaries and the values of the different parameters that are in the fitness.

Table 1: Values of the fitness in the original and planned routes.

Route	Fitness value	Distance (kms)	Distance improv. (%)	Time (minutes)	Time improv. (%)	Zone calification	Act calification
Original	2.839	140.7	0.0	209.8	0.0	0.892	1.0
Planned	10.75	97.0	31.04	149.5	28.75	1.0	0.585

Comparing the original fitness (2.839) with the planned one (10.75) does not offer much information until we analyse the contributions of the different parameters. The number of kms varies from 140.7 Km in the original itinerary to 97 Km in the planned one. This provides a distance improvement of the 31.04%.

The information related to the time is extracted from the columns time (min) and time improvement in (%). The driver took 209.8 minutes to perform all the shippings while our algorithm took 149.5 minutes. This means a reduction of 28.75% of the employed time.

Analazing graphically the results, a similar behavior is observed with the time and the distance in both routes. At the beginning, the shippings performed by the driver takes a big advantage over the planned ones (over 30 kms or 60 minutes). But in the last part of the shippings the planned route beats the original plan. This is a common behaviour in all the drivers analysed: at the beginning the humans try to do the shippings closer to the starting point. Instead, the EC algorithm looks for a solution that does not minimize the initial part of the route but all of it. Figure1 shows the original and planned routes using the Google Maps API. Another common mistake that we have detected is that the drivers try to do the shippings that are in the same street. Although it seems a logical reasoning, some streets cross the city from north to south or east to west. Following that criteria can higly increase the number of kms since the driver has to drive back to a point close to the previous shipping.

The time for generating the results by our EC algorithm is not crucial since planning routes can be generated off-line by our tool and given to the driver before he starts his working day. The logistic company distributes the packages of each driver in advance and just a small percentage of new unknown pick ups occur when the driver has already started the shipping.

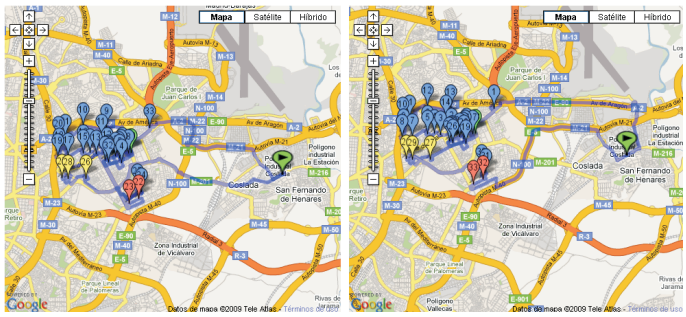

Fig. 1: Plans comparison using Google Maps.

6 Conclusions and future work

In this paper we have described the AI-based application that we have developed for a logistic operator company that combines Case-Based Reasoning (CBR) and Evolutionary Computation (EC) techniques. CBR is used to separate and learn the most frequent areas that the experienced drivers follow. These techniques allow one to separate the daily incidents that generate noise in the routes, from the decision made based on the knowledge of the route. The EC techniques plan optimal routes from the learning areas and evaluate those routes.

Our next step will be to include a new module in the application that pre-process automatically or in a mixed-initiative way the addresses bad introduced by the drivers. Due that the 20% of the streets and zip code cannot be automatically corrected, this represent a bottleneck to efficiently show the results of the tool.

Acknowledgements We want to thank Antonio Montoya for his contribution in the tool developed. This work has been supported by the Espi & Le Barbier company and the public projects funded by the Spanish Ministry of Science and Innovation under the projects COMPUBIODIVE (TIN2007-65989), V-LeaF (TIN2008-02729-E/TIN) and by Castilla-La Mancha project PEII09-0266-6640.

References

1. Agnar Aamodt and Enric Plaza. Case-based reasoning: Foundational issues, methodological variations, and system approaches. *Artificial Intelligence Communications*, 7(1):39–52, 1994.
2. Thomas Bäck, David B. Fogel, and Zbigniew Michalewicz. *Evolutionary Computation 1. Basic Algorithms and Operators*, chapter Permutations, pages 139–149. Institute of Physics Publishing, 1984.
3. A.E. Eiben and J.E. Smith. *Introduction to Evolutionary Computing*. Springer, 2009.
4. John H. Holland. *Adaptation in natural and artificial systems*. MIT Press, Cambridge, MA, USA, 1992.
5. Hiroaki Sengoku and Ikuo Yoshihara. A fast tsp solver using ga on java. In *3rd AROB*, 1998.