Künstl Intell (2018) 32:61–75
DOI 10.1007/s13218-017-0516-6

TECHNICAL CONTRIBUTION

Fuzzy-Logic Controlled Genetic Algorithm for the Rail-Freight Crew-Scheduling Problem

E. Khmeleva[1] · **A. A. Hopgood**[2,3] · **L. Tipi**[1] · **M. Shahidan**[1]

Received: 7 March 2017 / Accepted: 13 October 2017 / Published online: 27 October 2017

Abstract This article presents a fuzzy-logic controlled genetic algorithm designed for the solution of the crew-scheduling problem in the rail-freight industry. This problem refers to the assignment of train drivers to a number of train trips in accordance with complex industrial and governmental regulations. In practice, it is a challenging task due to the massive quantity of train trips, large geographical span and significant number of restrictions. While genetic algorithms are capable of handling large data sets, they are prone to stalled evolution and premature convergence on a local optimum, thereby obstructing further search. In order to tackle these problems, the proposed genetic algorithm contains an embedded fuzzy-logic controller that adjusts the mutation and crossover probabilities in accordance with the genetic algorithm's performance. The computational results demonstrate a 10% reduction in the cost of the schedule generated by this hybrid technique when compared with a genetic algorithm with fixed crossover and mutation rates.

✉ A. A. Hopgood
adrian.hopgood@port.ac.uk

E. Khmeleva
e.khmeleva@shu.ac.uk

L. Tipi
l.tipi@shu.ac.uk

M. Shahidan
m.shahidan@shu.ac.uk

1 Sheffield Business School, Sheffield Hallam University, Howard Street, Sheffield S1 1WB, UK

2 Faculty of Technology, University of Portsmouth, Portland Building, Portland Street, Portsmouth PO1 3AH, UK

3 HEC Liege - Management School, University of Liege, 4000 Liege, Belgium

Keywords Fuzzy logic · Genetic algorithm · Hybrid · Crew scheduling · Rail freight

1 Introduction

While international trade continues to expand, businesses are striving to increase reliability and reduce their environmental impact. As a result, demand for rail freight increases every year and rail-freight carriers attempt to maximize their efficiency. The crew cost constitutes 20–25% of the total rail-freight operating cost and is second only to the cost of fuel. Therefore, even a small improvement in the scheduling processes can save a company millions of dollars a year.

Daily crew planning starts after the train schedule has been finalized. It consists of two phases: crew scheduling and crew rostering. Crew scheduling involves grouping a sequence of train trips into shifts. Crew rostering, on the other hand, concerns assignment of drivers to each shift.

Within the industry, the driver's schedule is called a *diagram*. Each diagram contains instructions for the driver of what he or she should do on a particular day. Each diagram must start and end at the same station and obey all labour laws and trade union agreements. These rules regulate the maximum diagram duration, maximum continuous and aggregate driving time in a diagram, and minimum break time. As diagrams lasting more than 1 day are rare in the UK, the assumption of this research is that the maximum duration is 24 h. In addition, the terms trip and journey will be used interchangeably.

All drivers are located in *depots* where they start and finish their work. Depots are distributed fairly evenly across the UK. Sometimes in order to connect two trips that finish and start at different locations, a driver has to travel on a passenger train, taxi or a freight train driven by another driver. The situation of a driver travelling as a passenger while on duty is called *deadheading*. The cost of deadheading varies and depends on the means of transportation and business agreements between operating companies. Despite the potential cost, deadheading is sometimes inevitable and it can benefit the overall schedule [1].

Due to the employment contract terms, the drivers are paid the same hourly rate for any time spent on duty regardless of the number of hours they have actually been driving the train. Moreover, in accordance with collectively bargained contracts, each driver has a fixed number of working hours per year, so the company is obliged to pay for all the stated hours in full even if some of the hours are not utilized. Paid additional overtime hours can be worked at the driver's discretion. Thus, it is in the best interests of the company to use the agreed driving hours in the most efficient and economical way.

Taking all of this into consideration, the operational objectives for the diagrams are:

1. Minimize a number of unused and excess contract hours at the end of the year. To achieve this objective, it is preferable for each diagram to be of approximately the same average length of time, which is the annual contract hours divided by the number of the working days.

$$|T_{diagram} - \overline{T}| \rightarrow \min$$

$$T_{diagram} = T_{driving} + T_{deadheading} + T_{break} + T_{idle}$$

$$\overline{T} = \frac{T_{contract}}{N_{days}}$$

2. Maximize the throttle time, $T_{throttle}$, i.e. the proportion of the work shift that is actually spent driving a train. It excludes time for deadheading and waiting between trips.

$$T_{throttle} = \frac{T_{driving}}{T_{diagram}}$$

2 Approaches to crew scheduling

The crew-scheduling problem (CSP) is usually solved in two stages. At the first stage, all possible diagrams satisfying the industrial constraints are enumerated. At the second stage, only the set of diagrams that covers the entire schedule in the most cost-effective way is identified. Diagrams are usually modelled as binary vectors (Table 1) where '1' denotes that the trip *i* is included in the diagram *j*, otherwise '0' is inserted. Each diagram has its own cost. The deadhead journeys are displayed by including the same trip in more than one diagram. In the rest of the article the terms diagram and column will be used interchangeably.

Table 1 Diagrams

	Diagram 1	Diagram 2	Diagram 3	Diagram 4
Trip1	1	0	0	1
Trip2	0	1	1	0
Trip3	0	1	0	1
Trip4	0	1	0	1
Trip5	1	1	0	0

Although the generation of the diagrams can be performed in a simple and relatively straightforward manner using various graph search and label-setting techniques [2], finding an optimal set of diagrams may be highly time-consuming. The problem boils down to the solution of the 0–1 integer combinatorial optimization set covering problem (SCP):

$$Minimize \sum_{j=1}^{m} c_j x_j$$

$$Subject\ to: \sum_{i=1}^{n} a_{ij} x_j \geq 1$$

$$x_j \in \{0, 1\}$$
$$i = 1, 2 \ldots n\ trips$$
$$j = 1, 2 \ldots m\ diagrams$$

where a_{ij} is a decision variable indicating whether a trip *i* is included in the diagram *j; x_j* shows if the diagram is included in the schedule; c_j is the cost of the diagram.

The complete enumeration of all possible diagrams is likely to be impractical due to the large geographical scope of operations, the number of train services, and industry regulations. Typically, the number of generated diagrams reaches 300,000–400,000 for small problems and can be up to 50–75 million for the large ones [3, 4].

Country-wide planning creates a large number of opportunities for drivers to change freight trains, while passenger trains and taxi services connecting a large number of stations exponentially expand the graph topology. Furthermore, checks such as maximum driving time, minimum breaks and maximum diagram length need to be conducted while traversing the graph. These checks ensure compliance with industrial regulations, but substantially increase the computation time at the diagram creation stage.

2.1 Branch-and-price

Linear programming methods such as branch-and-price [5, 6] have been popular for the solution of medium-sized CSPs in the passenger train and airline industries [7]. These methods usually rely on a column-generation approach, where the main principle is to generate diagrams in the course of the algorithm, rather than having them all constructed a priori. Despite the ability of the algorithm to work with an incomplete set of columns, the column generation method alone does not guarantee an integer solution of the SCP. It is usually used in conjunction with various branching techniques that are able to find the nearest integer optimal solution. However, this approach is less suitable for the CSP in rail freight, where the possible number of diagrams tends to be considerably higher.

2.2 Genetic algorithms

Linear programming (LP) has been used for CSPs since the 1960s [8], but genetic algorithms (GAs) were introduced more recently [9]. GAs have been applied either for the production of additional columns as a part of column generation [8] or for the solution of an SCP from the set of columns generated prior to the application of a GA [9–12], but there are not yet any reports of them solving both stages of the problem. Since the diagrams are generated outside the GA in advance, the GA cannot change or add new columns. The GA is therefore confined to finding only good combinations from a pre-determined pool of columns.

For the solution of a CSP with a GA, chromosomes are normally represented by integer or binary vectors. Integer vector chromosomes contain only the numbers of the diagrams that constitute the schedule. This approach requires knowledge of the minimum number of diagrams in the schedule and this information is usually obtained from the cost lower bounds. Lower bounds are usually acquired through the solution of LP relaxation for an SCP [13]. Since the number of diagrams in the optimal solution tends to be higher than the lower bound, Costa et al. [14] have suggested the following approach. In the first population, the chromosomes have a length equal to the lower bound. Then, if a solution has not been found within a certain number of iterations, the length of the chromosome increases by one. This process repeats until the termination criteria are met.

In the binary vector representation, each gene stands for one diagram. The figure ‘1’ denotes that the diagram is included in the schedule, otherwise it is ‘0’. Although the detailed information about times and locations is stored separately and only applied when a chromosome is decoded into the schedule, such chromosomes usually consist of several hundred thousand genes. The number of diagrams can be unknown and the algorithm is likely to need a large number of iterations in order to solve the problem.

The application of genetic operators often violates the feasibility of the chromosomes, resulting in certain trips being highly over-covered (i.e. more than one driver assigned to the train) or under-covered (i.e. no drivers assigned to the train). One way of resolving this difficulty is to penalize the chromosome through the fitness function in accordance with the number of constraints that have been violated. However, the development of the penalty parameters can be problematic as in some cases it is impossible to verify them analytically and they are usually designed experimentally [15]. The penalty parameters are therefore data-dependent and likely to be inapplicable to other industries and companies. Moreover, the feasibility of the entire population is not guaranteed and might be achieved only after a large number of iterations.

Another more straightforward approach to maintaining the feasibility is to design heuristic “repair” operators. These operators are based on the principles “REMOVE” and “INSERT”. They scan the schedule and remove certain drivers from the over-covered trips and assign those drivers to under-covered journeys [13, 15]. This procedure might have to be repeated several times, leading to high memory consumption and increased computation time.

2.3 Adaptable genetic algorithm

Two of the common challenges associated with design of GAs are stalled evolution and premature convergence. Multiple genetic operators, random offspring generation, and dynamic parameter adjustment are among the methods for tackling these problems [16, 17]. The challenges in the design of an efficient GA with multiple operators are: identification of the optimal quantity of genetic operators, selection of those operators that would complement each other’s strengths, and definition of utilization rules. Creation of offspring at random, rather than through the crossover operator, can be inefficient for a large-scale problem due to the large number of potential gene permutations, lowering the probability of producing more fit and diverse offspring.

Genetic parameters such as crossover rate and mutation rate govern the exploration and exploitation phases. Poor selection can lead to premature convergence due to reduced diversity in the population over several iterations [18]. While the mutation operator is usually responsible for the maintenance of diversity, an extremely high level of mutation at the beginning can impede convergence on the solution. On the other hand, a very low level of mutation at the beginning might lead to poor exploration of the search region and the algorithm might not be able to arrive at the optimal solution.

To achieve a balance, several adaptive techniques that dynamically adjust the mutation and crossover rates have been proposed. One approach modifies the values of GA parameters proportionally to the distance between the best and average fitness in the population [19]. Designing an evolutionary algorithm for the crew scheduling problem, Kwan et al. [20] suggest selecting the mutation probability individually for each chromosome rather than for the entire population. The longer the individual has been in the population, the higher its probability of undergoing mutation. Both approaches rely on pre-defined crisp rules. However, the criteria for optimal selection of crossover and mutation are ambiguous and hard to model. Crisp rules cannot always adequately deal with the intricacies of the parameter adjustment process. For this reason, fuzzy-logic controllers, which are able to handle uncertainty and imprecision, have been applied in this research.

Wang et al. [21] were amongst the first researchers to propose the incorporation of fuzzy logic controllers within GAs in order to optimize the GA parameters. The configuration of a standard fuzzy-logic controller (FLC) is illustrated in Fig. 1. At each iteration of the GA, the information about its current performance is passed onto the FLC. The FLC then processes it and produces a recommendation for how the GA parameters should be altered in order to achieve more optimal execution. There are four critical components that support the FLC: a rule-base, a fuzzification unit, an inference engine, and a defuzzification unit.

The rule-base contains expert knowledge, expressed in the form of IF-THEN rules, which determine the relationship between the input and output. When applied to GA parameter management, the typical principle is to increase the mutation rate and decrease the crossover rate when the algorithm is converging [22–26].

Following the rules stored in the rule-base, the fuzzification unit estimates the degree to which the parameters belong to fuzzy sets. In the context of GA parameter control, fuzzy sets represent the crossover and mutation rates. The membership functions of the fuzzy sets are defined by linguistic variables (i.e. Low, Medium, and High).

The role of the inference engine is to identify the required level of changes to the GA parameters at a given iteration. The decision is made on the basis of the information received from the rule-base and fuzzification units. Finally, the defuzzification element returns scalar values of crossover and mutation rates.

While the architecture of the FLC remains the same across different fields of research and applications, the input parameters vary significantly. The input parameters can be broken down to two types: phenotype-based and genotype-based parameters. The first group deals with changes in the fitness function, whereas the genotype-based group concerns the structure of chromosomes.

As an example of phenotype measurements, Herrera and Lozano [22] utilize the convergence measure (CM), defined as the ratio between the best fitness on the current iteration and the best fitness on the previous iteration. In another experiment, they enhance this ratio with the number of generations of unchanged best fitness and the variance of the fitness, in order to amend both mutation and crossover rates. Hongbo et al. [25] use the average fitness value in relation to the best fitness in the population and changes of the average and best fitness over several iterations to solve the crew grouping problem in military operations. This approach was adopted later for the detection of high-resolution satellite images [23] and for optimal wind-turbine micrositing [26]. Homayouni and Tang [27] propose the use of indicators such as the best value of the fitness function, the frequency of the chromosomes with the similar best value, and the percentage of the same chromosomes in the population. In contrast, another FLC [28] relies on the changes in the value of the best fitness and population diversity.

Along with phenotype attributes, some authors consider genotype properties [24, 29]. They assess the Hamming distance between the chromosomes with the best fitness and the worst fitness in relation to the length of the chromosome. This approach promotes diversity, not only in the fitness functions, but also in the structure of the individuals.

3 GA-generated crew schedules

This section presents the use of a genetic algorithm to generate crew schedules in the context of UK freight-train logistics. It starts with an explanation of the input data types and chromosome encoding procedure. Then the designed crossover and mutation operators are presented.

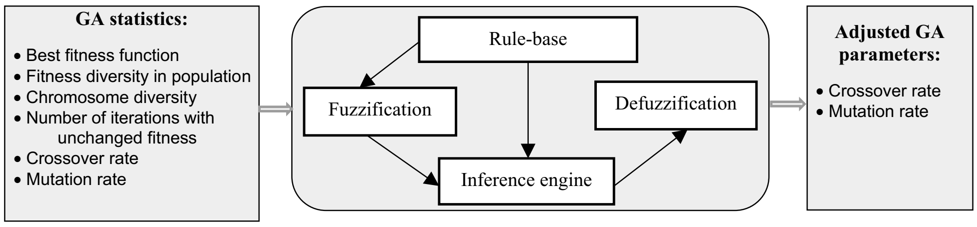
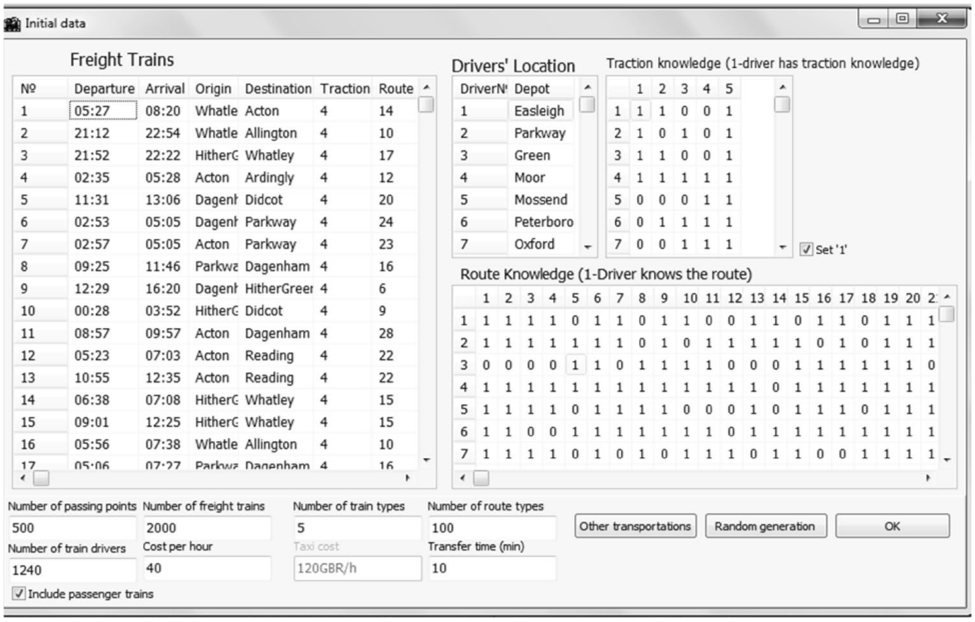


Fig. 1 Fuzzy-logic controller

3.1 Initial data

The process starts with a user uploading the freight train and driver data (Fig. 2). Each train has the following attributes: place of origin, destination, departure time, arrival time, type of train, and route code. The last two attributes indicate the knowledge that a driver must have in order to operate a particular train. The system also stores information about the drivers, i.e. where each driver is located and his or her traction and route knowledge. In the boxes marked 'traction knowledge' and 'route knowledge', each row represents a driver and each column denotes either a route or traction code. The binary digits indicate whether a particular driver is capable of driving a certain train or knows a certain route. The program also captures all the passenger trains and distance between cities, which is needed to calculate any taxi costs (Fig. 3).

After all the necessary data have been uploaded, the GA is applied to construct an efficient schedule. The proposed algorithm overcomes the aforementioned challenges through



Freight Trains

Nº	Departure	Arrival	Origin	Destination	Traction	Route
1	05:27	08:20	Whatle	Acton	4	14
2	21:12	22:54	Whatle	Allington	4	10
3	21:52	22:22	HitherG	Whatley	4	17
4	02:35	05:28	Acton	Ardingly	4	12
5	11:31	13:06	Dagenh	Didcot	4	20
6	02:53	05:05	Dagenh	Parkway	4	24
7	02:57	05:05	Acton	Parkway	4	23
8	09:25	11:46	Parkwa	Dagenham	4	16
9	12:29	16:20	Dagenh	HitherGreen	4	6
10	00:28	03:52	HitherG	Didcot	4	9
11	08:57	09:57	Acton	Dagenham	4	28
12	05:23	07:03	Acton	Reading	4	22
13	10:55	12:35	Acton	Reading	4	22
14	06:38	07:08	HitherG	Whatley	4	15
15	09:01	12:25	HitherG	Whatley	4	15
16	05:56	07:38	Whatle	Allington	4	10
17	05:06	07:27	Parkwa	Dagenham	4	16

Drivers' Location

DriverNº	Depot
1	Easleigh
2	Parkway
3	Green
4	Moor
5	Mossend
6	Peterboro
7	Oxford

Traction knowledge (1-driver has traction knowledge)

	1	2	3	4	5
1	1	1	0	0	1
2	1	0	1	0	1
3	1	1	0	0	1
4	1	1	1	1	1
5	0	0	0	1	1
6	0	1	1	1	1
7	0	0	1	1	1

[✓] Set '1'

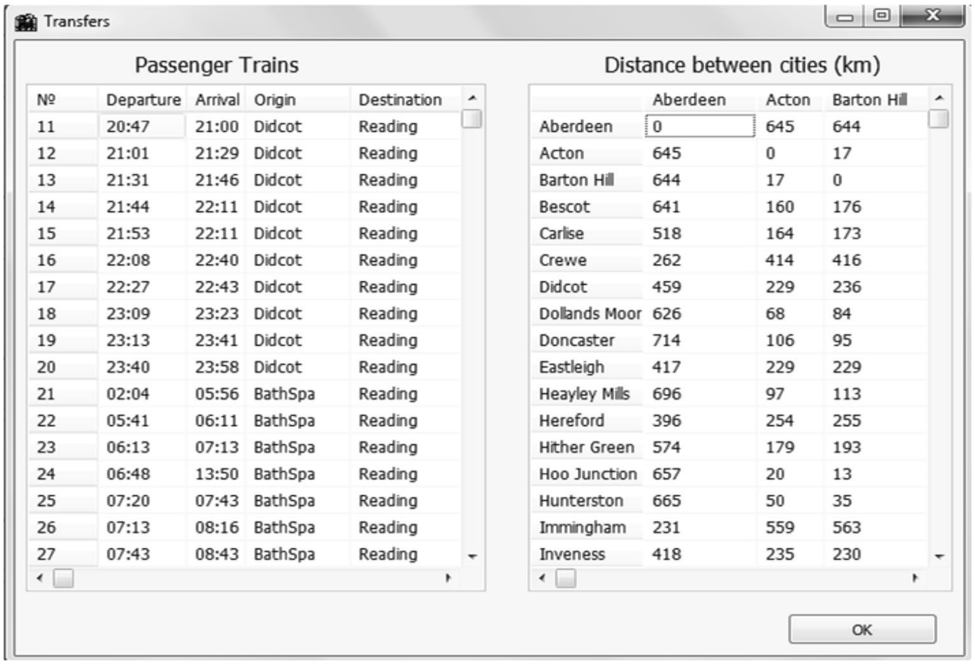
Route Knowledge (1-Driver knows the route)

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	2
1	1	1	1	1	0	1	1	0	1	1	0	0	1	1	0	1	1	0	1	1	1
2	1	1	1	1	1	1	1	0	1	0	1	1	1	1	1	0	1	0	1	1	1
3	0	0	0	0	1	1	0	1	1	1	1	0	0	0	1	1	1	1	1	1	0
4	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1	1	1	1	1	1	1
5	1	1	1	1	0	1	1	1	1	0	0	0	1	0	1	1	1	0	1	1	1
6	1	1	0	0	1	1	1	1	1	1	1	0	1	1	1	1	1	1	1	1	1
7	1	1	1	1	0	1	0	1	0	1	1	1	0	1	1	0	0	1	1	1	1

Number of passing points	Number of freight trains	Number of train types	Number of route types
500	2000	5	100
Number of train drivers	**Cost per hour**	**Taxi cost**	**Transfer time (min)**
1240	40	120GBR/h	10

[✓] Include passenger trains

Fig. 2 Freight trains and drivers



Passenger Trains

Nº	Departure	Arrival	Origin	Destination
11	20:47	21:00	Didcot	Reading
12	21:01	21:29	Didcot	Reading
13	21:31	21:46	Didcot	Reading
14	21:44	22:11	Didcot	Reading
15	21:53	22:11	Didcot	Reading
16	22:08	22:40	Didcot	Reading
17	22:27	22:43	Didcot	Reading
18	23:09	23:23	Didcot	Reading
19	23:13	23:41	Didcot	Reading
20	23:40	23:58	Didcot	Reading
21	02:04	05:56	BathSpa	Reading
22	05:41	06:11	BathSpa	Reading
23	06:13	07:13	BathSpa	Reading
24	06:48	13:50	BathSpa	Reading
25	07:20	07:43	BathSpa	Reading
26	07:13	08:16	BathSpa	Reading
27	07:43	08:43	BathSpa	Reading

Distance between cities (km)

	Aberdeen	Acton	Barton Hill
Aberdeen	0	645	644
Acton	645	0	17
Barton Hill	644	17	0
Bescot	641	160	176
Carlise	518	164	173
Crewe	262	414	416
Didcot	459	229	236
Dollands Moor	626	68	84
Doncaster	714	106	95
Eastleigh	417	229	229
Heayley Mills	696	97	113
Hereford	396	254	255
Hither Green	574	179	193
Hoo Junction	657	20	13
Hunterston	665	50	35
Immingham	231	559	563
Inveness	418	235	230

Fig. 3 Passenger trains and taxis

a novel alternative chromosome representation and special decoding procedure. It allows the feasibility of chromosomes to be preserved at each iteration without the application of repair operators. As a result, the computational burden is considerably reduced.

3.2 Chromosome representation

The chromosome is represented by a series of integers, where each integer stands for the number of the trip (Fig. 4). The population of chromosomes is generated at random and then the trips are allocated in series to the diagrams using a specific decoding procedure, which is discussed below and summarized in Table 2.

Starting from the leftmost gene, the procedure finds a driver with the necessary route and traction knowledge to operate that trip and creates a new diagram for him or her. Then the procedure checks if the same driver is able to drive on the next journey (i.e. the second gene). If it is possible, then that trip is added to his or her diagram. If the origin station for the current trip differs from the destination station of the previous trip, the algorithm first searches for passenger trains and the freight company's own trains that can deliver a driver within the available time slot to the next job location, e.g. Diagram 1, between trips 3 and 8 (Fig. 4). If no such trains have been found, but there is a sufficient interval between the trips, then the algorithm inserts a taxi journey.

The information regarding driving times and the current duration of the diagrams is stored. Before adding a new trip, the algorithm inserts breaks if necessary. If the time expires and there are no trains to the home depot that a driver can drive, the deadheading activity completes the diagram, as in Diagram 2 (Fig. 4). If a trip cannot be placed in any of the existing diagrams, the procedure takes another driver from a database and creates a new diagram for him or her.

On rare occasions, a few diagrams might be left with only a few trips and a duration that is less than the minimum (as shown in lines 38–52 in the pseudocode). This is due to the fact that other drivers are either busy at this time or located at different stations. In order to tackle this problem, a mechanism has been added for finding and assigning a driver from a remote depot with the lowest workload. This approach not only solved the problem of the short diagrams, but also helped in distributing the workload more equally across the depots. After the implementation of this procedure, the algorithm has been tested on various data sets including real and randomly generated data. None of the chromosomes has been reported to violate the constraint.

The given representation has a visual resemblance to the flight-graph representation suggested by Ozdemir and Mohan [30], but the decoding procedures are different. The flight-graph representation generates trips based on a depth-first graph search, whereas in the proposed GA they are produced at random. Random generation is beneficial since it does not exclude situations where a driver can travel to another part of the country to start working in order to have even workload distribution across the depots, while depth-first search usually places only geographically adjusted trips together.

The advantage of the proposed chromosome representation is that it creates both the crew schedule and the crew roster for a single day within the same algorithm, thereby giving the GA greater control over the solution. It also does not require the generation of a large number of diagrams at the beginning. In addition, this representation does not leave under-covered trips and ensures that no unnecessary over-covering happens. This is because chromosome scanning and trip allocation continue until all the trips are placed into diagrams, even if a new diagram is created for a single trip. Over-covering only occurs when a deadhead is required and does not occur otherwise. It is possible that at the beginning of the algorithm this chromosome representation might produce schedules with a high number of deadheads. However, due to the specific fitness function and genetic operators, the number of chromosomes containing deadheads decreases rapidly with evolution.

3.3 Cost function

The objective function, i.e. the function to be optimized, is represented as the cost of the schedule. The cost (to be minimized) is the opposite of the fitness (to

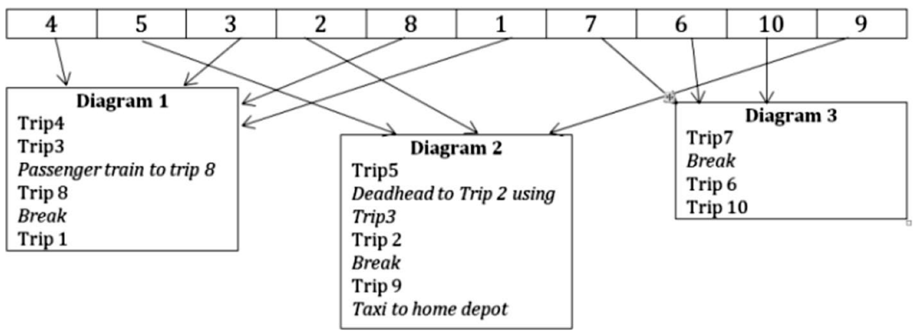


Fig. 4 Chromosome representation and decoding procedure

Table 2 Pseudocode: decoding procedure

```
Inputs:
    Set of train trips T
    Set of drivers: J
Outputs:
    Set of Diagrams D

1:  $CurrentLocation_d \leftarrow 0$
2:  $CurrentTime_d \leftarrow 0$
3:  $NoBreakTime \leftarrow 0$
4:  $DrivingTime_d \leftarrow 0$
5:  $Cost_d \leftarrow 0$
6:  $TransferTime \leftarrow 0$
7:  $TransferCost \leftarrow 0$
8:
9:  DEFINE SUBROUTINE FindTransfer ($Origin_i$, $CurrentLocation_d$,
10:     $CurrentTime_d$, $Departure_i$)
11:     Get the transfer route using Google Maps API
12: RETURN (Transfer, TransferCost, TransferTime, TransferDepartureTime,
13:     TransferArrivalTime)
14:
15: DEFINE SUBROUTINE AddTheTripToDiagram ($Destination_j$,
16:     $CurrentTime_d$, $Arrival_j$, $DrivingTime_d$, T)
17:     // The diagram starts earlier if the driver has to travel to the first job
18:     IF FirstTrip = TRUE AND Transfer = TRUE THEN
19:         $StartTime_d \leftarrow TransferDepartureTime$
20:         $FirstTrip \leftarrow$ FALSE
21:     END IF
22:     // Add the trip into the diagram and update the location and time
23:     $CurrentLocation_d \leftarrow Destination_j$
24:     $CurrentTime_d \leftarrow Arrival_j$
25:     // Keep track of driving time and ensure compliance with minimum break
26:             regulations
27:     $DrivingTime_d \leftarrow DrivingTime_d + Arrival_i - Departure_i$
28:     $NoBreakime \leftarrow NoBreakTime + Arrival_i - Departure_i$
29:     IF $NoBreakime > MaxBreakTime$ THEN
30:         $CurrentTime_d \leftarrow CurrentTime_d + MinBreakTime$
31:         $NoBreakime \leftarrow 0$
32:     END IF
33:     $Transfer \leftarrow$ FALSE
34:     Delete the trip i from the set
35: RETURN ($CurrentLocation_d$, $CurrentTime_d$, $DrivingTime_d$, NoBreakime
36:     FirstTrip, T)
37:
38: DEFINE SUBROUTINE CompleteTheDiagram ($CurrentLocation_d$,
39:     $HomeDepot_j$, $Cost_d$)
40:     // Checking whether this is the first trip allows for prevention of blank
41:         diagrams in the scenario when a driver could not operate any of the
42:         trips that are as yet unassigned
43:     IF FirstTrip = FALSE THEN
44:         IF $CurrentLocation_d \neq HomeDepot_j$ THEN
45:             GetTransfer ($Origin_i$, $CurrentLocation_d$, $CurrentTime_d$, $Arrival_i$)
46:         END IF
47:         $CurrentLocation_d \leftarrow HomeDepot_j$
48:         $CurrentTime_d \leftarrow TransferArrivalTime$
49:         $Cost_d \leftarrow Cost_d + TransferCost$
50:         $TransferCost \leftarrow 0$
51:     END IF
52: RETURN (Diagram)
```

Table 2 (continued)

```
53:
54: START
55: // Check whether the trip can be inserted into the diagram
56: WHILE trips set T is not empty
57:     // Check if there are no trips in the diagram
58:     IF FirstTrip = TRUE THEN
59:         $CurrentLocation_d \leftarrow HomeDepot_j$
60:     END IF
61:     // Check whether the trip starts after a driver becomes available
62:     IF $Departure_i > CurrentTime_d$
63:     // Ensure that the driver has required skills to operate the trip
64:     AND RouteKnowledge (j, i) = TRUE
65:     AND TractionKnowledge (j, i) = TRUE
66:     // Verify that the insertion of the trip will not violate the regulations
67:             related to the diagram length
68:     AND $Arrival_j$ - $StartTime_d$ < DiagramMaxTime
69:     THEN
70:         // Check if a driver needs to travel to the next job location
71:         IF $Origin_i = CurrentLocation_d$ THEN
72:             AddTheTripToDiagram ($Destination_j$, $CurrentTime_d$, $Arrival_j$,
73:             $DrivingTime_d$, $Cost_d$, TransferCost)
74:             i = i + 1
75:         ELSE
76:             FindTransfer ($Origin_i$, $CurrentLocation_d$, $CurrentTime_d$,
77:                 $Departure_i$)
78:             IF Transfer = TRUE THEN
79:                 AddTheTripToDiagram ($Destination_j$, $CurrentTime_d$, $Arrival_j$,
80:                     $DrivingTime_d$, $Cost_d$, TransferCost)
81:             ELSE
82:                 i = i + 1
83:             END IF
84:         END IF
85:     END IF
86:
87:     // The end of the chromosome is reached and the set of trips is not empty
88:     IF i = NumberOfTrips
89:     OR $CurrentTime_d$ - $StartTime_d$ > DiagramMaxTime
90:     THEN
91:         CompleteTheDiagram ($CurrentLocation_d$, $HomeDepot_j$, $Cost_d$)
92:         // Select the next driver
93:         $j \leftarrow j + 1$
94:         // This condition prevents blank diagrams (in which a driver cannot
95:             operate any of the trips)
96:         IF FirstTrip=FALSE THEN
97:             $d \leftarrow d + 1$
98:         END IF
99:     END IF
100: END WHILE
101: FINISH
```

be maximized). The direct cost consists of the drivers' working hours and expenses for additional transportation. In order to penalize those solutions with unequal workload distribution or where the diagram length deviates from the target value, the second part of the cost function represents potential losses associated with these two additional criteria:

$$Cost = \sum_{i=1}^{N} \left(T_{diagram_i} \times C_{HourlyRate} + C_{deadheading_i} + | T_{diagram_i} - \bar{T}_{diagram} | \times C_{HourlyRate} \right) + \sqrt{\frac{1}{m} \left(T_{depot_m} - \bar{T}_{depot} \right) \times C_{HourlyRate}}$$

where i is the number of trips, m is the number of depots, and the average diagram duration is assumed to be 8.5 h

3.4 Selection

Preference was given to binary tournament selection due to the smaller bias towards fittest individuals, lower selection pressure, non-reliance on population sorting and ranking procedures, and execution time and memory efficiency [31]. It is also a popular selection strategy that is used in numerous GAs for CSP [9, 30]. Binary tournament selection can be described as follows. Two individuals are selected at random from the population and the fittest among them constitutes the first parent. The same process repeats for the selection of the second parent.

3.5 Crossover and mutation

Since one- or two-point crossover might produce invalid offspring by removing some trips or copying the same journey several times, a crossover mechanism has been designed to utilize domain-specific information without interfering with the number of the trips. The process is illustrated in Fig. 5. Firstly, the process detects genes responsible for diagrams with a high throttle time in the first parent. As the throttle time shows the proportion of productive work time in the diagram, the higher the throttle time, the fewer deadhead trips and unnecessary breaks between the trips are included in the diagram. The trips constituting diagrams with a higher throttle time are shown in darker shades in Fig. 5. Typically, these diagrams consist of a large number of trips. However, in some cases, they can comprise just a few trips of long duration. In both scenarios, the throttle time would be high.

Once diagrams with high throttle times have been identified, these genes are copied to the first child and the rest of the genes are added from the second parent. The same procedure is then used to form the second child. By preserving the good parts of the chromosome accumulated through evolution, the implemented crossover was able to provide a schedule with a high throttle time much faster than traditional crossover that randomly mixes the parents' genes to form their offspring.

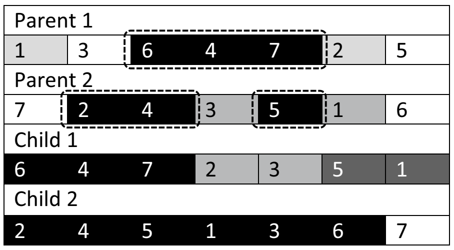


Fig. 5 Crossover. Trips enabling higher throttle time are shown in darker shades

In order to maintain diversity in the population, randomly selected genes are mutated with 40% probability. The mutation is performed by swapping two randomly identified genes. The mutation probability was determined through numerous tests and empirical observations.

4 Fuzzy-logic controller

Unlike the algorithm devised by Ozdemir and Mohan [30], the proposed algorithm manipulates both the crossover and mutation rates. Both adjustments are required for the attainment of an optimal balance between the exploration and exploitation phases. The aim was to maintain a substantial level of diversity, while at the same time attempting to avoid random walking [24]. The technique is a modification of the algorithms proposed by [25, 32] and is presented in more detail below.

The population statistics are computed after each iteration using the following formulas:

$$CF = \left(\frac{Cost_{best(t-1)}}{Cost_{best}} - 1 \right) \times 100\%$$

$$VF = \frac{\overline{Cost_{(t)}} - Cost_{best(t)}}{Cost_{best(t)}}$$

where CF is the increase in the objective function from the previous iteration, VF is the variance of the fitness in the population, and UF is the number of iterations without improvement in the fitness function. These parameters are sent to the FLC for processing. Three linguistic variables {Low, Medium, High} are employed. The corresponding membership functions for fuzzification of CF, VF and UF are illustrated in Fig. 6a–c. The output is the level of adjustment of the mutation and crossover rates (Δp_m and Δp_c, respectively). Figure 6d shows some possible alterations in mutation and crossover rates. While the expert knowledge and fuzzy rules were derived from published work [31], the membership parameters were obtained experimentally. The general principle for these rules is to increase the mutation rate when the fitness function remains unchanged in order to facilitate exploration. Conversely, the crossover rate is increased as necessary to facilitate exploitation, i.e. to encourage algorithm to converge faster. The fuzzy rules that have been applied are presented in Table 3 and can be expressed textually as follows:

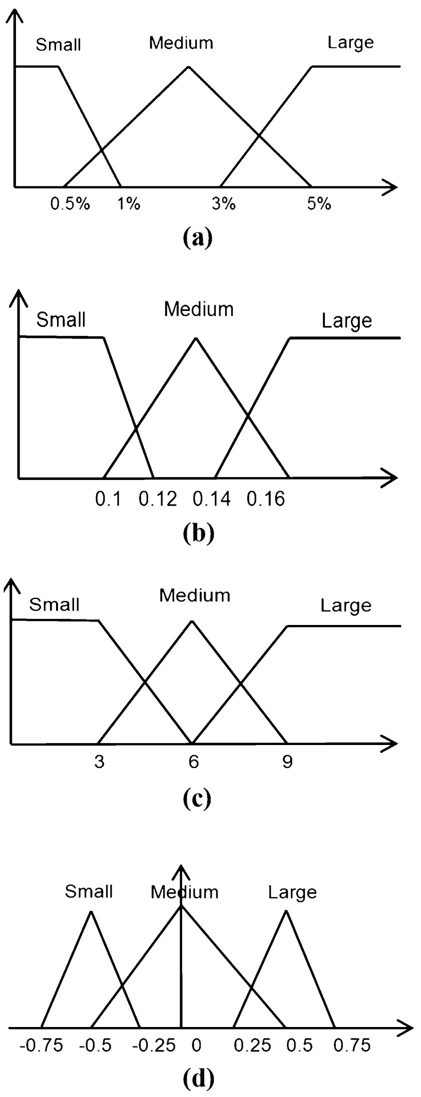


Fig. 6 **a** Membership functions for CF. **b** Membership functions for VF. **c** Membership functions for UF. **d** Membership functions for Δp_m and Δp_c

If CF is high and UF is low, then p_m becomes low and p_c becomes high
If CF is medium and UF is low, then p_m becomes low and p_c becomes high
If CF is low and UF is high, then p_m becomes high and p_c becomes low
If CF is low and UF is medium, then p_m and p_c become medium
If CF is low and UF is low, then p_m becomes low and p_c becomes high
If UF is low and VF is low, then p_m becomes high and p_c becomes medium
If UF is low and VF is medium, then p_m and p_c become medium

Table 3 FLC rules

Input variables to fuzzy rules			Controlled parameters	
CF	UF	VF	p_m	p_c
High	Low	High	Low	High
High	Low	Medium	Low	High
High	Low	Low	Low	High
Medium	Low	High	Low	High
Medium	Low	Medium	Low	High
Medium	Low	Low	Low	High
Low	High	High	High	Low
Low	High	Medium	High	Low
Low	High	Low	High	Low
Low	Medium	High	Medium	Medium
Low	Medium	Medium	Medium	Medium
Low	Medium	Low	Medium	Medium
Low	Low	High	Low	High
Low	Low	Medium	Low	High
Low	Low	Low	Low	High
High	Low	Low	High	Medium
Medium	Low	Low	High	Medium
Low	Low	Low	High	Medium
High	Low	Medium	Medium	Medium
Medium	Low	Medium	Medium	Medium
Low	Low	Medium	Medium	Medium

After processing these parameters and performing centroid defuzzification, the FLC updates the mutation and crossover rates that are applied in the next generation.

5 Experimental results

A standard genetic algorithm (SGA) has been tested on a full daily data set obtained from one of the largest rail-freight operators in the UK. These real-world data comprised 2000 freight-train legs, 500 cities, 39 depots, 1240 drivers, 500,000 passenger-train links, and taxi trips connecting any of the stations at any time. Figures 7 and 8 illustrate a 3-h run of the algorithm. The SGA reduced the cost of the schedule while achieving the two operational objectives of maximized throttle time and minimized deviation from the average shift duration. Increasing the throttle time indicates a reduction in deadheads and unnecessary waiting, thereby reducing the number of drivers required to operate the given trains. The decrease in deviation of the diagram duration from the average can be translated into equal utilization of the contract hours during the year.

In our previous work [33], the efficiency of the standard genetic algorithm (SGA) customized for the CSP (known as GACSP) was compared against two established approaches. The first was branch-and-price (B&P), i.e. the

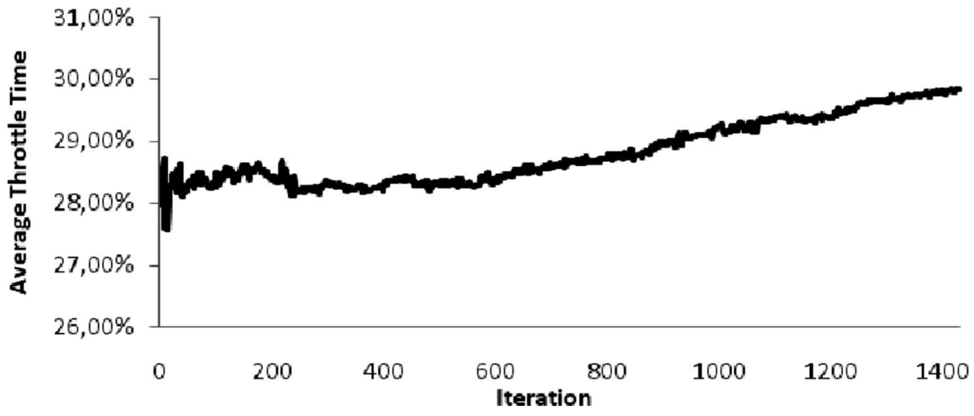


Fig. 7 Maximizing average throttle time

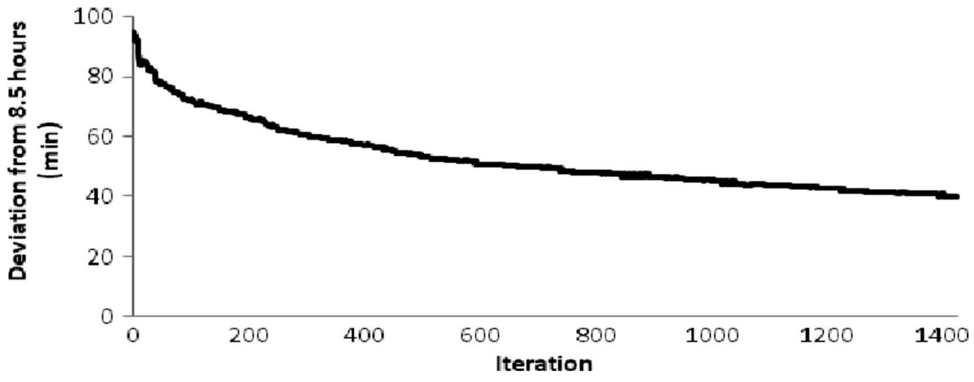


Fig. 8 Minimizing deviation from the average shift length of 8.5 h

combination of column generation and branch and bound methods [6]. The second comparator was Genetic Algorithm Process Optimization (GAPO), a genetic algorithm for CSP enhanced with repair and perturbation operators [9]. A reduced data set of six cities, 180 train legs, and 500 passenger-train links was used, as the B&P method failed to converge with the full data set. For the GAs, the population size was 20, crossover rate 90%, and mutation probability 40%. The tests showed that the SGA produces an acceptable solution within a shorter timeframe than either of the alternatives (Table 4).

In order to evaluate the contribution of the fuzzy-logic controller, we experimentally compared a fuzzy genetic algorithm (FGA) against the SGA. They were both implemented in C++ Builder and run on a computer with 4 GB RAM and a 3.4 GHz Dual Core processor. For both SGA and FGA, the population consisted of 100 individuals. Throughout the SGA execution, the crossover and mutation rates were fixed at 90% and 40% respectively. These same rates were used as initial values for the first iteration of the FGA.

Figure 9a illustrates the reduction of the cost defined in Sect. 3.3 as each algorithm progresses through each of 2000 iterations. Figure 9b shows the changes in mutation and crossover probabilities in the FGA.

Although the FGA started from a worse solution than that of the SGA, the cost descended faster and it successfully outperformed the SGA. The crossover rate initially increased while the mutation rate decreased, allowing better exploitation of the beneficial aspects of the existing solutions. From that point onwards, the crossover rate fell while the mutation rate grew, thereby balancing population diversity and exploration of the search space with exploitation of the optimal region.

In order to validate the comparison, additional experiments were carried out on ten test instances. The artificially generated data imitated the real data sets. Each instance was tested 20 times. In order to provide fair comparison and to give an equal chance to all data sets to complete both stages of exploitation and exploration, the number of iterations was allocated in accordance with their sizes. The structure of the ten randomly created data sets and the summary of the results for each data set are displayed in Table 5. Figure 10 demonstrates the difference between the average results obtained through a standard GA and the GA enhanced by the fuzzy-logic parameter controller.

The FGA yields better results than the SGA in terms of the average and minimum cost. While the FGA outperforms the SGA in all instances regardless of the size of the problem, the best results were achieved on instance 7, where the FGA outperformed the SGA by more than 10%. From a financial perspective, this improvement can be translated into a substantial cost saving of £105,801.

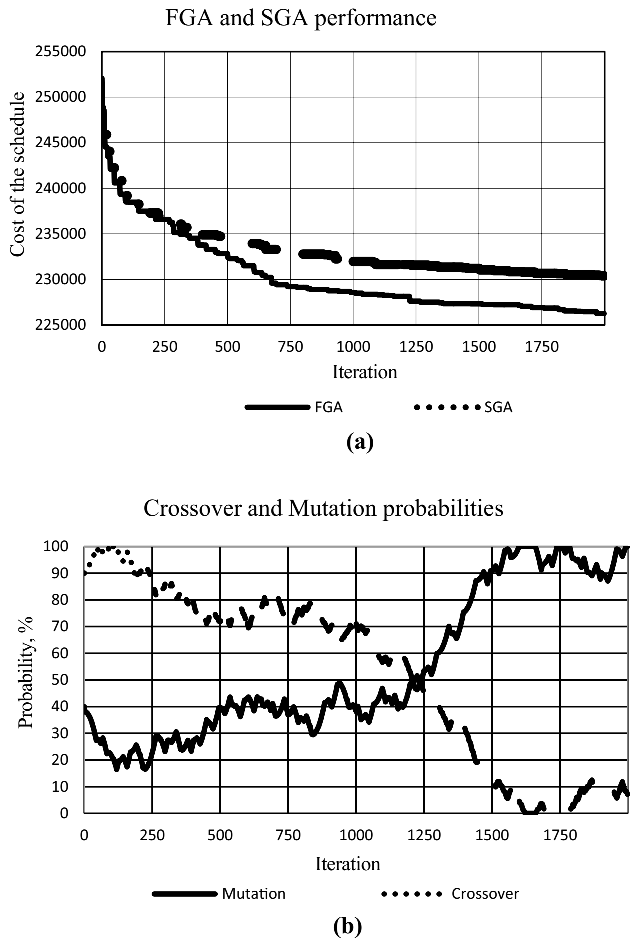


Fig. 9 **a** The performance of SGA and FGA. **b** Adaptation of p_m and p_c

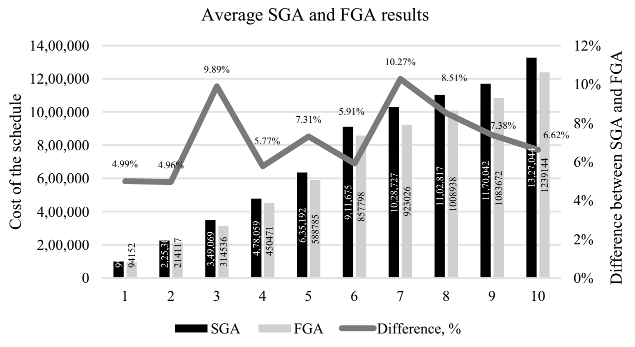


Fig. 10 The difference in SGA and FGA average results

In terms of the standard deviation of the results, the FGA was found to be less stable than the SGA. This can be explained by the fact that the fuzzy-logic controller forced the algorithm to explore a broader search space, and the FGA did not converge to the same degree as the SGA within the same number of iterations. It also can be noticed that the level of standard deviation increases with the size of the problem, which can be related to a larger number of possible permutations of the trips and hence higher population diversity.

Finally, Table 6 displays a user-friendly example of the solution, i.e. one of the diagrams produced. It shows the sequence of trips and breaks that a driver needs to take on a particular day.

As such complex and large-scale scheduling operations are currently performed manually, automation of these operations can result in substantial operational benefits. These include enhancement of the schedule quality, reduction in the cost of generating the schedule, and faster schedule creation. Schedule cost savings can be invested in business development. Saved time can be spent on dealing with last-minute customer requests, and the staff can be allocated to less routine and more value-added business activities.

6 Potential implementation and integration issues

The most common implementation problems with software for scheduling transit systems concern robustness [34], i.e. the ability of the schedule to remain valid despite disturbances such as delays and cancellations. An example of such disturbances might be the delay of the previous train, resulting in the driver being unable to catch the planned

Table 4 Comparison of experimental results using a reduced data set

Computation time (mins)	B&P			GAPO			GACSP		
	60	120	228	60	120	228	60	120	228
Number of diagrams	–	–	22	32	28	26	25	23	23
Throttle time (%)	–	–	63	50	56	59	60	62	62
Average Number of deadheads per shift	–	–	1.36	2.21	1.85	1.60	1.66	1.47	1.47
Deviation from the average (mins)	–	–	46	51	48	47	62	57	57

Table 5 Comparative results of SGA and FGA

#	Depots	Trips	Iterations	Standard GA			Fuzzy GA		
				Min	Avr	Std Dev	Min	Avr	Std Dev
1	2	50	5281	99,063	99,102	38	94,104	94,152	47
2	3	123	8791	225,019	225,300	269	213,818	214,117	291
3	5	196	11,401	348,159	349,069	883	313,572	314,536	927
4	7	269	16,033	477,144	478,059	875	449,388	450,471	1033
5	9	342	18,215	633,389	635,192	1756	586,888	588,785	1798
6	13	488	25,056	909,782	911,675	1848	855,522	857,798	2150
7	11	415	22,794	1,025,868	1,028,727	2773	920,068	923,026	2839
8	15	561	30,949	1,099,869	1,102,817	2835	1,005,723	1,008,938	3136
9	17	634	45,500	1,166,330	1,170,042	3570	1,079,760	1,083,672	3813
10	19	707	55,735	1,322,024	1,327,044	4743	1,234,070	1,239,144	4832

A cost function is used, so the lower the result the better the performance of the algorithm

Table 6 A typical diagram, i.e. driver schedule

Driver	Start time	End time	Activity	Origin	Destination
113	12:18	12:28	Book on	Westbury	Westbury
113	12:28	13:35	Driving	Westbury	Swindon
113	13:47	15:07	Driving	Swindon	Wootton Wawen
113	15:07	15:49	Break	Wootton Wawen	Wootton Wawen
113	15:59	17:29	Driving	Wootton Wawen	Swindon
113	17:37	21:00	Passenger Train	Swindon	Westbury
113	21:00	21:10	Book off	Westbury	Westbury
Diagram length: 8:52				Throttle time: 46%	

train. In our system, the transfer time regulates how much time is allocated for a driver to leave the previous train and start working on the next one. The larger the interval between trips, the lower the risk that the next freight train will be delayed by the late arrival of the previous one. On the other hand, a large transfer time decreases the throttle time and requires more drivers to cover the trips. The best way to tackle this situation is to have an effective re-scheduling mechanism that makes changes in as few diagrams as possible.

In addition, the crew scheduling process is extremely complex. It is not always possible to model all the rules, nuances and exceptions of the schedule. For this reason, the system-generated diagrams have to be revised and amended by an experienced human planner until all the knowledge has been fully acquired.

Finally, although GAs are able to find an acceptable solution relatively quickly, their susceptibility to premature convergence around a sub-optimal solution has inspired the current investigation into a fuzzy controller for parameter adjustment. Convergence can be controlled either by embedding variations in the selection procedure or by changing the mutation and crossover rates.

7 Conclusions

In this paper, the complexities of the CSP in the rail-freight industry in the UK have been described. Due to a high monetary cost of train crews, the profitability and success of a freight company might rely heavily on the quality of the constructed crew schedule. Given the wide geographical spread, numerous regulations, and severely constrained planning time, an effective automated crew scheduling system can increase staff productivity and equip a company with valuable decision-making support.

In order to solve the CSP problem, we have proposed a novel FGA. The permutation chromosome representation and genetic operators are able to preserve the validity of the chromosomes. This design enables the user to retrieve a feasible schedule at any iteration. It also eliminates the need for additional repair operators or penalty functions, thereby saving memory resources.

Unlike other GAs for CSP, the FGA has the ability to amend its mutation and crossover probabilities so as to reduce the risk of being trapped in a local optimum. While the parameters for the fuzzy membership functions would ideally be adjusted for a specific data set, the suggested parameters proved their applicability to a wide range of data sets from 50 to 707 trips.

In future work, it would be interesting to study the suitability of the FGA with the proposed parameters on other data instances and other permutation problems. As the crossover and mutation operators have a strong impact on the chromosome formation and the algorithm's behavior, it would be informative to investigate whether their dynamic change in the course of the algorithm might improve the algorithm's performance.

Acknowledgements This research has been supported by Sheffield Business School at Sheffield Hallam University. The authors would like to thank DB-Schenker Rail (UK) for the provision of data and information about real-world crew-scheduling operations.

References

1. Barnhart C, Hatay L, Johnson EL (1995) Deadhead selection for the long-haul crew pairing problem. Oper Res. doi:10.1287/opre.43.3.491
2. Drexl M, Prescott-Gagnon E (2010) Labelling algorithms for the elementary shortest path problem with resource constraints considering EU drivers' rules. Log Res. doi:10.1007/s12159-010-0022-9
3. Kwan RSK (2004) Bus and train driver scheduling. Handbook of scheduling: algorithms, models, and performance analysis. Chapman and Hall/CRC, Boca Raton
4. Klabjan D, Johnson EL, Nemhauser GL, Gelman E, Ramaswamy S (2001) Solving large airline crew scheduling problems: random pairing generation and strong branching. Comput Optim Appl. doi:10.1023/A:1011223523191
5. Duck V, Wesselmann F, Suhl L (2011) Implementing a branch and price and cut method for the airline crew pairing optimization problem. Pub Transp. doi:10.1007/s12469-011-0038-9
6. Barnhart C, Johnson EL, Nemhauser GL, Savelsbergh MWP, Vance PH (1998) Branch-and-price: Column generation for solving huge integer programs. Oper Res. doi:10.1287/opre.46.3.316
7. Derigs U, Malcherek D, Schafer S (2010) Supporting strategic crew management at passenger railway—model, method and system. Pub Trans. doi:10.1007/s12469-010-0034-5
8. dos Santos AG, Mateus GR (2009) General hybrid column generation algorithm for crew scheduling problems using genetic algorithm. Evol Comp. doi:10.1109/CEC.2009.4983159
9. Zeren B, Özkol I (2012) An improved genetic algorithm for crew pairing optimization. J Intell Learn Syst Appl 4:70–80. doi:10.4236/jilsa.2012.41007
10. Souai N, Teghem J (2009) Genetic algorithm based approach for the integrated airline crew-pairing and rostering problem. Eur J Oper Res. doi:10.1016/j.ejor.2007.10.065
11. Park T, Ryu K (2006) Crew pairing optimization by a genetic algorithm with unexpressed genes. J Intell Manuf. doi:10.1007/s10845-005-0011-z
12. Kornilakis H, Stamatopoulos P (2002) Crew pairing optimization with genetic algorithms. Methods Appl Artif Intell. doi:10.1007/3-540-46014-4_11
13. Kwan RSK, Wren A, Kwan ASK (2000) Hybrid genetic algorithms for scheduling bus and train drivers. Evol Comput. doi:10.1109/CEC.2000.870308
14. Costa L, Santo IE, Oliveira P (2014) An adaptive constraint handling technique for evolutionary algorithms. Optimization. doi:10.1080/02331934.2011.590486
15. Chu PC, Beasley JE (1998) Constraint handling in genetic algorithms: The set partitioning problem. J Heuristics. doi:10.1080/02331934.2011.590486
16. Rocha M, Neves J (1999) Preventing premature convergence to local optima in genetic algorithms via random offspring generation. In: Imam I, Kodratoff Y, El-Dessouki A, Ali M (eds) Multiple approaches to intelligent systems, IEA/AIE 1999. Lecture notes in computer science, vol 1611. Springer, Berlin, Heidelberg. doi:10.1007/978-3-540-48765-4_16
17. Herrera F, Lozano M (1996) Adaptation of genetic algorithm parameters based on fuzzy logic controllers. Genetic Algorithms and Soft Computing. Physica-Verlag, Heidelberg, Germany, pp. 95–125
18. Varnamkhasti MJ, Lee LS, Bakar MRA, Leong WJ (2012) A genetic algorithm with fuzzy crossover operator and probability. Adv Oper Res. doi:10.1155/2012/956498
19. Srinivas M, Patnaik LM (1994) Adaptive probabilities of crossover and mutation in genetic algorithms. Syst Man Cyber. doi:10.1109/21.286385
20. Kwan RSK, Kwan ASK, Wren A (2001) Evolutionary driver scheduling with relief chains. Evolut Comput. doi:10.1162/10636560152642869
21. Wang PY, Wang GS, Song YH (1996) Fuzzy logic controlled genetic algorithms. In: Proceedings of the fifth IEEE international conference on fuzzy systems. doi:10.1109/FUZZY.1996.552310
22. Herrera F, Lozano M (2003) Fuzzy adaptive genetic algorithms: design, taxonomy, and future directions. Soft Comput. doi:10.1007/s00500-002-0238-y
23. Sumer E, Turker M (2013) An adaptive fuzzy-genetic algorithm approach for building detection using high-resolution satellite images. Comput Environ Urban Syst. doi:10.1016/j.compenvurbsys.2013.01.004
24. McClintock S, Lunney T, Hashim A (1997) A fuzzy logic controlled genetic algorithm environment. In: IEEE international conference on systems, man, and cybernetics. Computational cybernetics and simulation. doi:10.1109/ICSMC.1997.635189
25. Hongbo L, Zhanguo X, Abraham A (2005) Hybrid fuzzy-genetic algorithm approach for crew grouping. In: Proceedings of the 5th international conference on intelligent systems design and applications, ISDA '05. doi:10.1109/ISDA.2005.51
26. Yang J, Zhang R, Sun Q, Zhang H (2015) Optimal wind turbines micrositing in onshore wind farms using fuzzy genetic algorithm. Math Probl Eng. doi:10.1155/2015/324203
27. Homayouni S, And Tang S (2015) A fuzzy genetic algorithm for scheduling of handling/storage equipment in automated container terminals. Int J Eng Technol. doi:10.7763/IJET.2015.V7.844
28. Lau HCW, Nakandala D, Zhao L (2015) Development of a hybrid fuzzy genetic algorithm model for solving transportation scheduling problem. J Inf Syst Technol Manag. doi:10.4301/S1807-17752015000300001
29. Zhu KQ, Liu Z (2004) Population diversity in permutation-based genetic algorithm. In: Boulicaut JF, Esposito F, Giannotti F, Pedreschi D (eds) Machine learning: ECML 2004. Lecture notes in computer science, vol 3201. Springer, Berlin, Heidelberg. doi:10.1007/978-3-540-30115-8_49

30. Ozdemir HT, Mohan CK (2001) Flight graph based genetic algorithm for crew scheduling in airlines. Inf Sci. doi:10.1016/S0020-0255(01)00083-4
31. Razali NM, Geraghty J (2011) Genetic Algorithm Performance with Different Selection Strategies in Solving TSP, World Congress on Engineering, vol II WCE 2011, ISBN: 978-988-19251-4-5
32. Yuhui S, Eberhart R, Yaobin C (1997) Implementation of evolutionary fuzzy systems. IEEE Trans Fuzzy Syst. doi:10.1109/91.755393
33. Khmeleva E, Hopgood AA, Tipi L, Shahidan M (2014) Railfreight crew scheduling with a genetic algorithm. In: Bramer M, Petridis M (eds) Research and development in intelligent systems XXXI. Springer, Cham. doi:10.1007/978-3-319-12069-0_16
34. Gopalakrishnan B, Johnson EL (2005) Airline crew scheduling: state-of-the-art. Ann Oper Res. doi:10.1007/s10479-005-3975-3