Soft Comput (2016) 20:2951–2966
DOI 10.1007/s00500-015-1924-x

FOCUS

Ant colony optimization with immigrants schemes for the dynamic railway junction rescheduling problem with multiple delays

Jayne Eaton[1] · **Shengxiang Yang**[1] · **Michalis Mavrovouniotis**[1]

Published online: 19 November 2015

Abstract Train rescheduling after a perturbation is a challenging task and is an important concern of the railway industry as delayed trains can lead to large fines, disgruntled customers and loss of revenue. Sometimes not just one delay but several unrelated delays can occur in a short space of time which makes the problem even more challenging. In addition, the problem is a dynamic one that changes over time for, as trains are waiting to be rescheduled at the junction, more timetabled trains will be arriving, which will change the nature of the problem. The aim of this research is to investigate the application of several different ant colony optimization (ACO) algorithms to the problem of a dynamic train delay scenario with multiple delays. The algorithms not only resequence the trains at the junction but also resequence the trains at the stations, which is considered to be a first step towards expanding the problem to consider a larger area of the railway network. The results show that, in this dynamic rescheduling problem, ACO algorithms with a memory cope with dynamic changes better than an ACO algorithm that uses only pheromone evaporation to remove redundant pheromone trails. In addition, it has been shown that if the ant solutions in memory become irreparably infeasible it is possible to replace them with elite immigrants, based on the best-so-far ant, and still obtain a good performance.

Keywords Dynamic railway junction rescheduling · Ant colony optimization · UK railway network · Rail transportation · Dynamic optimization problem

Communicated by D. Neagu.

This work was supported by the Engineering and Physical Sciences Research Council (EPSRC) of UK under Grant EP/K001310/1.

✉ Jayne Eaton
jayne.eaton@email.dmu.ac.uk

Shengxiang Yang
syang@dmu.ac.uk

Michalis Mavrovouniotis
mmavrovouniotis@dmu.ac.uk

[1] Centre for Computational Intelligence (CCI), School of Computer Science and Informatics, De Montfort University, The Gateway, Leicester LE1 9BH, UK

1 Introduction

The problem of rescheduling trains after a delay is an important concern of the railway industry. Although timetables are designed to ensure that trains run on time and without conflict a delayed train may miss its scheduled time slot at a junction or a station and may cause knock-on delays to other trains in the railway network.

The problem is further complicated by the fact that, while a train controller is trying to minimise the delay at a particular point in time, more trains will be arriving at the affected area. These trains may have different priorities to those already waiting to be rescheduled, which makes the problem a dynamic one that changes over time. In addition, in an extremely disrupted situation, more trains may be delayed during the time period of the disruption, so that while the train controller is trying to resolve one delay, a different train may be delayed for a totally unconnected and unrelated reason.

The rescheduling of trains after a perturbation is usually dealt with by human controllers (Fan et al. 2012), who often use simple rules such as first come first served (FCFS) (D'Ariano et al. 2007). Although FCFS may resolve the immediate problem, it may not be the optimal solution in terms of minimizing the effect of a train delay in a dynamically changing environment.

The aim of this work is to extend the dynamic junction simulator for the dynamic railway junction rescheduling problem (DRJRP), introduced in Eaton and Yang (2014), to include multiple delays taking place over the period of the simulation. As ant colony optimization (ACO) gave promising results in the original single delay simulator, it is proposed to investigate its application in the new multiple delay simulator. In addition, the algorithms will be adapted to be able to resequence the trains at the stations as well as at the junctions. This is seen as the first step in extending the algorithms to address a wider section of the railway network.

Allowing the ants to sequence trains at the stations as well as at the junction means that the ants must have the power to change the arrival order of the trains at the junction. This has implications for the memory repair operation that is needed to reinitialise population-based ACO (P-ACO) algorithms after a dynamic change has taken place (see Sect. 4.3). In effect it makes the repair extremely difficult. For this reason, it is proposed to investigate replacing the ants in memory after a dynamic change with random, elite or hybrid immigrants. Random immigrants are randomly created solutions; elite immigrants are based on the best-so-far solution and hybrid immigrants are a mixture of random and elite immigrants. The ACO algorithms with immigrant schemes will be evaluated against an algorithm that has no built in ability to cope with dynamic problems, i.e., the Max–Min ant system (Stützle and Hoos 1997) and a P-ACO algorithm that does not have the ability to resequence the trains in the stations.

The rest of this paper is organised as follows. Section 2 describes previous work in the area of train scheduling and rescheduling using evolutionary computation (EC) techniques. Section 3 explains the original DRJRP, the extension introduced for this research, and the extended simulator created to model the junction and the station. Section 4 considers ACO algorithms and their previous application to dynamic scheduling problems while Sect. 5 gives details of the algorithms used in this research. An experimental study carried out to investigate the ability of different ACO algorithms to solve the extended DRJRP is described in Sect. 6. Finally, Sect. 7 concludes this paper with ideas for future work.

2 Related work

EC techniques are a group of techniques inspired by nature, which imitate evolution and natural self-organised systems. They include, among others, genetic algorithms (GAs), ACO, evolutionary strategies and particle swarm optimization.

There has been previous promising work on both train scheduling and train rescheduling using EC techniques (Fang et al. 2015). Gorman (1998) combined a tabu search algorithm with a GA to produce an optimised schedule for a major US freight railroad with the objective of minimizing operating costs. The schedule produced had a potential cost saving of 4 % and a reduction in service delay of 6 %. Tormos et al. (2008) addressed the difficult problem of adding new trains to a train schedule without affecting the existing trains. Their objective was to minimise the overall delay. Using a GA, they produced a timetable solution in around 300 s and their system outperformed comparison algorithms based on random sampling and regret biased based random sampling (RBRS). The GA created has been embedded into a computer-aided tool that is being successfully used by the Spanish manager of railway infrastructure. Qin et al. (2010) used a differential evolution algorithm to schedule a 185 km double-track section of the Shenyang–Siping railway corridor, while Abbas-Turki et al. (2011) used a GA to tackle the problem of scheduling high speed trains on the Thameslink route in London.

In contrast to train scheduling, which involves creating a fixed timetable of train times without conflict, rescheduling is concerned with recovering the railway timetable after a disruption. There have been some interesting approaches to solving the problem using EC techniques. Khan et al. (2006) used a GA to reschedule trains after a delay produced by randomly varying the departure and arrival times for trains on a simulated single track railway section. The GA produced a solution that reduced the train delay at the destination station from 35 to 12 min.

A number of researchers have looked at the problem of resequencing trains at a junction after a delay. Ho and Yeung (2000) encoded a GA to tackle the problem of creating a feasible sequence of train to pass through a junction to minimise conflict after a train delay. They found that their algorithm could produce a solution within less than 5 % of the optimal and with a reduced computation time compared to a solution produced using dynamic programming. Fan et al. (2012) also considered the problem of sequencing trains through a junction after a delay. They applied both a GA and an ACO algorithm and compared the results to FCFS and a brute force algorithm. Brute force will always find a solution as it involves enumerating all possible solutions. They found both the GA and ACO performed well on the static junction problem although ACO performed slightly better with a smaller computation time. Chen et al. (2010) used a modified differential evolution GA to tackle the problem of rescheduling trains after a delay at the St Pancras Midland Road Junction, which has three routes and two conflict points. The aim was to reschedule 24 trains in a 1-hour time window. They found that their algorithm performed significantly better, in terms of minimising passenger delays, than FCFS on both short and long delay test scenarios.

The above research shows the potential of EC techniques for the scheduling and rescheduling of trains. However, in every case, the problem considered is a static one. In the real

world, the rescheduling of trains after a perturbation does not exist in an isolated bubble; while trains are waiting to be re-scheduled at the junction, more trains could be arriving, and their arrival will change the nature of the problem over time.

3 The dynamic railway junction rescheduling problem

3.1 Description of the problem

The DRJRP under consideration is based on a static benchmark scenario created by Fan et al. (2012). It is concerned with a section of track on the Derby to Birmingham line, which takes in the North Stafford and Stenson Junctions. Both the junctions are 'flat junctions' in that the merging railroad tracks require that other trains cross over in front of opposing trains on the same level. Two trains can pass through the junction at the same time as long as this does not cause conflict with any other trains.

Figure 1 shows the junction and delay scenario under investigation. There are three trains waiting on each of the four routes into the junction. Train 1 has been delayed by 5 min which means that train 7 has arrived before it on track A. This scenario is based on the second of Fan et al.'s (2012) delay scenarios. Figure 2 shows the junction after a dynamic change. Trains 7 and 8 have passed through the junction, but more timetabled trains have arrived while the remainder of the trains are waiting to be rescheduled. Train 13 has arrived on route A while train 14 has arrived on route C. The problem has changed as there is now a different combination of trains to sequence through the junction.

Each train has a delay penalty associated with it, which is the cost in pounds sterling per minute that a train company has to pay if the train is delayed. This is the same objective used by Fan et al. (2012). The aim is to find the best order of trains to pass through the junction that minimises the overall cost of the delay.

3.2 The extended DRJRP

The original DRJRP has been extended to introduce more delays to the simulator over time. In the extended version, trains added at the station during a dynamic change may also be delayed. This means that instead of dealing with just one delay at the beginning of the simulation, the algorithm also has to deal with additional delays that occur over time.

Introducing these delays relies on there being enough trains arriving at the station to ensure that a train arriving after its scheduled time slot at a station has an impact on the problem. For this reason, only the high magnitude dynamic problem, where eight trains are introduced at each dynamic change, is considered in this extended DRJRP problem. It is only when several trains are arriving at a station within the same time frame that a station delay can be engineered. In fact, an investigation into the pattern of arrival of the trains at each change showed that it is only at station D that sufficient trains are present at the station within the same time-frame to allow the introduction of station delays.

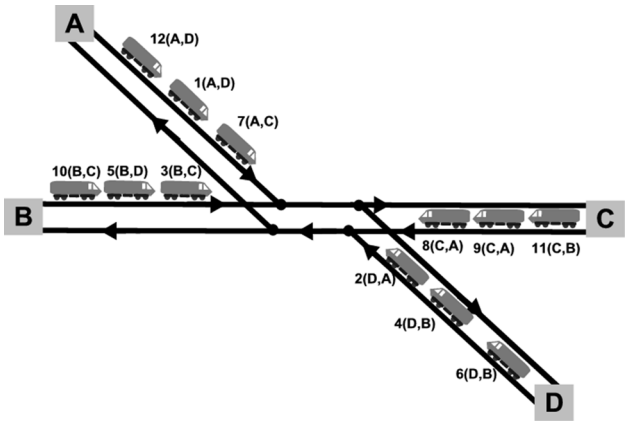

Fig. 1 The junction before a dynamic change (taken from Eaton and Yang 2014)

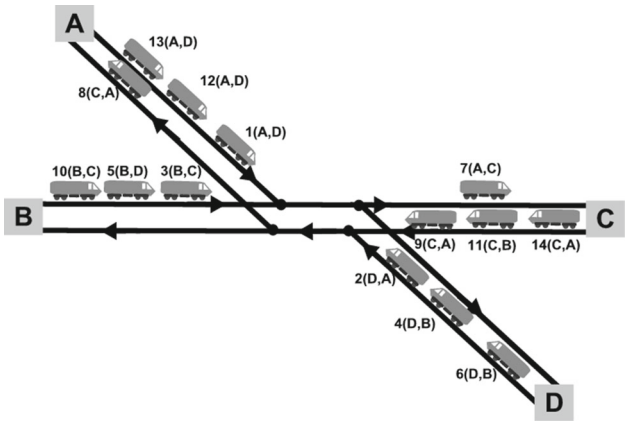

Fig. 2 The junction after a dynamic change (taken from Eaton and Yang 2014)

Table 1 shows the pattern of trains that arrive at station D when eight new trains are arriving every 10 min. After 10 min have passed (dynamic change 1), three trains arrive at station D: trains 14 and 18 arrive at platform 1 and train 16 arrives at platform 2. The first delay at this station is simulated by switching the arrival order of the two trains on the same platform; trains 18 and 14. A further delay can be introduced by also switching the arrival order of the two trains arriving after 40 min has passed (dynamic change 4). This involves switching trains 42 and 38. Delays after the fourth change are not introduced because they would not be visible in the low frequency dynamic scenarios where only four dynamic changes take place over the period of dynamic change.

Table 1 Train arrivals at station D

Time passed (min)	Platform 1	Platform 2
10	Train 14	Train 16
	Train 18	
20	Train 26	Train 28
30	Train 30	
40	Train 38	Train 40
	Train 42	
50	Train 50	Train 52
60	Train 54	

3.3 The Stenson Junction train simulator

To investigate the effectiveness of the algorithms under consideration, a train simulator was developed to allow the trains in each ant's solution to be evaluated. More details about the simulator can be found in Eaton and Yang (2014).

Table 2 shows the trains used, their routes through the junction, the penalty for delay and their scheduled arrival times. The delay penalties are different for each type of train and are taken from Fan et al. (2012). The timetable was created by running all trains, in their numerical order, through the simulator and recording their arrival times. This gave a baseline measurement to be able to calculate the delay of the trains after a perturbation. Each train is one of three types: Class 150 with a maximum running speed of 120 km/h, Class 200 with a maximum running speed of 200 km/h, or F2-mixed freight train with a maximum running speed of 110 km/h (Fan et al. 2012). Each type of train is of a different length, the Class 150 train is 80.24 m long, the Class 220 train is 187.4 m long and the F2-freight train is 355 m long. The length of a train as well as its speed affects how long the train takes to clear its previous track section. This in turn determines how quickly the following train can move into the train's vacated track.

Dynamism was introduced to the simulator by adding a specified number of trains (m) at a specified time interval (f). The number of trains added represents the magnitude of change, and the time interval relates to the frequency of change. The new trains are added by repeating the timetable shown in Table 2 in blocks of m trains. For example, when $m = 8$, at the first dynamic change, trains 1–8 are added to the simulation; at the second change, trains 9–12 and 1–4 are added and at the third change trains 5–12 are added to the simulator. Repeating the pattern in this way means that the trains are distributed across all the station platforms.

The extra trains can be thought of as an extended timetable for the train junction and each combination of the magnitude and frequency of change is run through the simulator to obtain the conflict-free timetable. All new trains are placed at the stations and are not allowed onto the track until the track section leaving the station is clear. At the point of change, any trains that are about to move into, or have moved into, the junction are removed by the simulator from the set of trains that need to be passed to the algorithm.

Table 2 also lists the station platform of each train in the simulation. Assigning trains to platforms is necessary to be able to model the trains at the stations as well as at the junction. This extension to the simulator is explained in more detail in the next section.

3.4 The extended Stenson Junction train simulator

For this investigation, the simulator was extended to model the stations that feed into the junction as well as the junction itself. Each station corresponds to a real-world station on the

Table 2 The scheduled timetable for each train with delay penalties (based on Fan et al. 2012)

Train number	Train type	Route	Delay penalty (£/min)	Scheduled arrival	Station platform
1	Class 150	A–D	20	12:10	1
2	Class 220	D–A	40	12:12	1
3	Freight	B–C	10	12:19	1
4	Class 220	D–B	40	12:15	2
5	Freight	B–D	10	12:20	2
6	Class 150	D–B	20	12:19	1
7	Freight	A–C	10	12:28	2
8	Class 150	C–A	20	12:22	1
9	Class 220	C–A	40	12:27	2
10	Class 220	B–C	40	12:32	3
11	Freight	C–B	10	12:39	1
12	Class 150	A–D	20	12:36	3

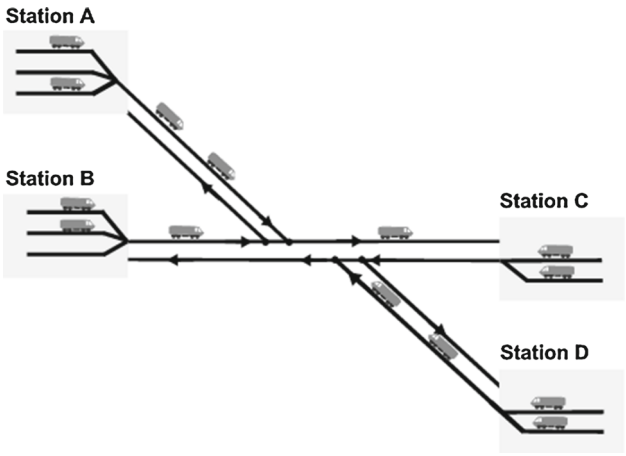

Fig. 3 The Stenson Junction modelled with stations

UK Railway Network. Station A is Crewe station; station B is Birmingham station; station C is Derby station; and station D is Nottingham station. Crewe and Birmingham are much larger stations than Derby and Nottingham. Both Crewe and Birmingham have 12 platforms, Derby has six platforms and Nottingham has seven platforms (National Rail Enquiries 2015). For this reason, stations A and B have been modelled with three exit platforms that feed into the junction while stations C and D have been modelled with only two. Figure 3 shows the stations with their platforms.

Modelling the section of the network in this way allows the algorithms to reschedule the trains at the stations as well as at the junction. More details about the approach taken to allow them to do this can be found in Sect. 5.1.

The following sections describe the algorithms used in this research and the related background concerning their previous use in similar dynamic rescheduling problems.

4 ACO for dynamic optimization problems (DOPs)

4.1 Basic ACO algorithm

An ACO algorithm is an optimization algorithm inspired by the ability of ants to follow pheromone trails laid down by other ants to discover food (Dorigo and Stützle 2004). As ants move backwards and forwards from the nest to a food source, they lay down pheromones on the ground, which can be sensed by other ants. Ants choosing the shortest path to the food source will return quicker, which ensures that the shortest path accumulates more pheromone. Ants tend to probabilistically choose paths with the strongest pheromone concentration, which means that a path with high pheromone levels will attract more ants and accumulate even more pheromone. In this way, the shortest path to a food source is marked by the strongest pheromone trail. However, if this trail were to persist after the food source was depleted, it would seriously hamper the ants' ability to find food. Therefore, pheromone trails evaporate over time to allow old decisions to be forgotten.

To apply this principle to an optimization problem, it has to first be decomposed into a fully connected weighted graph $G = (V, E)$, where V is a set of vertices or nodes and E is a set of edges or connections between the nodes. The ants move along the edges of the graph from node to node recording the nodes visited. This list of visited nodes, sometimes called the ant's tour, is one possible solution to the optimization problem. Pheromones are deposited on the edges of the graph by the ants according to how good an ant's solution is in terms of the optimization objective. On the next iteration, the updated pheromone levels help to guide the ants to choose better nodes. Pheromones can be decreased as well as increased to model the process of evaporation, which allows previous bad decisions to be forgotten. In addition to the pheromone, the edges may also be associated with a heuristic value, which is based on problem-specific knowledge and provides additional guidance to the ants.

4.2 Max–Min ant system (MMAS)

One of the most popular ACO algorithms is the Max–Min ant system (MMAS) (Stützle and Hoos 1997). In this algorithm, all of the pheromone trails are initialised to a maximum value. After each iteration, all pheromone trails are evaporated as in Eq. (1).

$$\tau_{ij} \leftarrow (1-\rho)\tau_{ij}, \quad \forall (i,j) \in L, \tag{1}$$

where $L = E$ is the set of all pheromones and $0 < \rho \leq 1$ is the pheromone evaporation rate (Dorigo and Stützle 2004), which is a constant parameter of the algorithm.

After each iteration, the pheromone trails are updated to correspond to the tour T^{best} of either the best-so-far ant or the best iteration ant as in Eq. (2).

$$\tau_{ij} \leftarrow \tau_{ij} + \Delta\tau_{ij}^{\text{best}}, \quad \forall (i,j) \in T^{\text{best}}. \tag{2}$$

The update value $\Delta\tau_{ij}^{\text{best}}$ is $\frac{1}{C}$, where C is the fitness of the best ant. In a minimization problem, as the fitness of the ant improves, the size of the pheromone update increases correspondingly.

An ant, say ant k, when at node i, chooses the next node j in its neighbourhood N_i^k, probabilistically as follows:

$$p_{ij}^k = \frac{[\tau_{ij}]^\alpha [\eta_{ij}]^\beta}{\sum_{l \in N_i^k} [\tau_{il}]^\alpha [\eta_{il}]^\beta}, \quad \text{if } j \in N_i^k, \tag{3}$$

where τ_{ij} is the pheromone information and η_{ij} is the heuristic information, α and β are constants which determine the relative influence of the pheromone and the heuristic values, respectively.

The pheromone trails in MMAS are bounded between a minimum $\tau_{\min}$ and a maximum $\tau_{\max}$ value. The reason for this is to counteract the increased possibility of stagnation that may occur as a result of allowing only the best ant to deposit pheromone. In addition, stagnation is addressed by reinitialising all trails to $\tau_{\max}$ when the algorithm shows stagnation behaviour or there has been no change in the best fitness for a set number of iterations. MMAS is unusual in that all pheromone trails are initialised to the maximum value, this together with a small evaporation rate increases the exploration of the search space at the start of the search (Stützle and Hoos 1997).

The pheromone bounds are given as follows: $\tau_{\max} = \frac{1}{C}$ and $\tau_{\min} = \frac{\tau_{\max}}{a}$, where a is a constant parameter of the algorithm. Each time a new best ant is found, the values for $\tau_{\min}$ and $\tau_{\max}$ are updated. In a minimization problem, this means that as the fitness of the best ant, i.e, C, improves the values for both $\tau_{\min}$ and $\tau_{\max}$ increase.

In the version of MMAS used in this investigation, the ant used to update the pheromones is chosen in a ratio of 10:1 of the best-so-far ant to the best iteration ant. For ten iterations, the best-so-far ant is chosen followed by one iteration when the best-iteration ant is chosen. This was found to give the best performance in preliminary investigations.

4.3 Population-based ACO (P-ACO)

The above algorithm, however, does not provide any extra mechanism to allow the ants to adapt to a change in the environment apart from the evaporation of pheromone trails which can be slow (Stützle and Hoos 1997). Once the ants have converged on a solution, the resulting loss in diversity will make it difficult for them to adapt to a change in the problem and, in addition, the pheromone trails laid down for the previous environment may not be relevant to the new environment.

One option is to restart the algorithm after a change but such an action is not only computationally wasteful but also results in the loss of information that has the potential to be useful in the new environment. To address this problem, Guntsch and Middendorf (2002) introduced a population-based ACO (P-ACO) algorithm. In this algorithm, the best ant found at each iteration is stored in a memory, called the population-list, and only the ants in this list are used to update the pheromone levels. When the population-list reaches its designated limit, an ant is removed and the pheromone trail for that ant is negatively updated. This provides a mechanism for allowing previous bad decisions to be forgotten. To prevent the pheromone levels from building up to a level which means that all ants follow the same path, the amount of pheromone on each edge is bounded between a minimum and a maximum value.

This memory of best iteration ants means that solutions made before the change can be retained to provide valuable information for the new environment. However, to make the ants suitable for the new environment, they may have to undergo a repair operation. Once repaired, the pheromone information for the new environment can be computed from the tours of the fittest ants created before the change, thus ensuring that information from the previous environment can be passed over into the new environment. Guntsch and Middendorf (2002) found P-ACO to perform better than restarting the algorithm when the environment change was small and frequent and comparable with restart when the change was large and slow.

P-ACO also differs from MMAS in two other aspects. First, the ants do not always choose the next node probabilistically using Eq. (3). Instead, they choose the next node probabilistically with a probability of $1 - q_0$; otherwise, they choose the next best node in terms of the pheromone and heuristic values. Second, in MMAS, the amount of pheromone laid down by the best ant is proportional to that ant's fitness, whereas in P-ACO it is a constant value.

In P-ACO, there are several strategies for removing ants from the memory when it becomes full. In the algorithm used in this paper, the worst ant is removed from the memory to make space for a new ant. This approach was found to work better than removing the oldest ant in a previous application of P-ACO to this problem (Eaton and Yang 2014).

4.4 ACO for dynamic rescheduling

There has been little work on using ACO for dynamic train rescheduling problems. As previously mentioned, Fan et al. (2012) used ACO for the Stenson Junction benchmark problem with promising results. However, it was a static problem.

There is a similarity between this DRJRP and a dynamic travelling salesman problem (DTSP). A static travelling salesman problem involves finding the shortest route for a salesman to visit a set of cities; it can be made dynamic by changing the number of cities (Guntsch and Middendorf 2001; Mavrovouniotis and Yang 2010, 2011a), or by changing the distances between cities (Mavrovouniotis and Yang 2013) over time. In both the DRJRP under investigation and the DTSP, the objective is to find the best sequence of nodes (trains or cities) that minimises an objective.

Several researchers have applied ACO to DTSPs (Guntsch and Middendorf 2001; Mavrovouniotis and Yang 2010, 2011a, 2013). Here, one issue is that once the ants have converged on a solution they will still follow the same pheromone trails after a dynamic change unless the trails are updated in some way to take into account the new environment.

Guntsch and Middendorf (2001) tackled this problem by modifying the pheromone trails after a change, either globally or locally to the city being removed or added. However,

the study involved only the insertion or deletion of one city at a time and they acknowledge that the results may have been different with multiple insertions or deletions. In addition, their solution requires knowledge of where the change has taken place to identify the pheromone trails in the local area.

Mavrovouniotis and Yang (2011a) also applied ACO to the DTSP, where the dynamic environment was generated by removing half of the cities from the problem and replacing existing cities with the removed cities. Again, the number of cities in the problem does not change overall as the number of cities removed is the same as the number of cities replaced. They found that an ACO algorithm, modified with a local search scheme, performed well on this problem. This previous work on the DTSP suggests that ACO may be applicable to the DRJRP.

4.5 Using immigrants for DOPs

The aim of an EC approach is to converge to an optimum solution. However, this is an issue if the problem is a dynamic one as there may not be enough diversity in the population to allow the algorithm to adequately explore the search space after a change in the problem. To help solve this issue, Grefenstette (1992) introduced the idea of adding random solutions, named immigrants, to the population to maintain diversity and to give the algorithm the flexibility to adapt to a changing environment.

Grefenstette found that adding random solutions to a population improved the performance of a GA on DOPs (Grefenstette 1992). In later work, Yang (2005) found that basing the immigrants on existing individuals that have been found to perform well in the new environment efficiently improved the performance of GAs for DOPs. While basing the immigrants on the best solution of the current generation worked well particularly when the changes in the environment were slow and slight (Yang 2007, 2008).

This idea has been extended to ACO algorithms on the DTSP where the problem was made dynamic by removing and adding cities (Mavrovouniotis and Yang 2010). In this case, at every iteration, a proportion of the worst ants was replaced with either random or elite immigrants or a hybrid mix of the two. This removed the need, in this DTSP problem, for the expensive and domain-specific repair of the solutions in memory that is needed by the P-ACO algorithm after a change in the environment. The addition of immigrants was found to significantly improve the performance of the P-ACO algorithm with the elite immigrants performing significantly better than random immigrants on slow changing environments and the random immigrants performing slightly better than elite immigrants on most fast changing environments. The hybrid immigrants were found to significantly outperform both the random and elite immigrants.

Mavrovouniotis and Yang (2013) also investigated the addition of immigrants schemes to the DTSP with traffic factors, where the distances between cities change over time in either a random or cyclic pattern. Again, the immigrants were found to provide significant improvement over the algorithms without immigrants schemes.

The above findings suggest that incorporating immigrants into the ACO algorithms for DOPs is not only possible but also effective.

5 Proposed P-ACO algorithm for the extended DRJRP

5.1 Framework of the proposed algorithms

In the original investigation (Eaton and Yang 2014), the problem was decomposed into a fully connected, partly one-directional, weighted graph (see Fig. 4) based on that designed by Van Der Zwaan and Marques (1999) for the job-shop scheduling problem. Each node in the graph is a train waiting to be rescheduled at the junction. Each horizontal line represents one of the routes into the junction; the first line from the top represents route A; the second route B; the third route C and the fourth route D. The one-way arrows along each horizontal axis ensure that the ants cannot schedule trains before other trains that are in front of them on the same track. The ants move around this graph selecting trains to make up a tour which gives the order that the trains are to be scheduled through the junction.

Node 0 represents the start node. At the beginning of each iteration, all ants are placed on this node. They then make a choice about which train node to choose next. After selecting a node, the next train on that train's track becomes visible to the ant and is included in its next decision. After all nodes

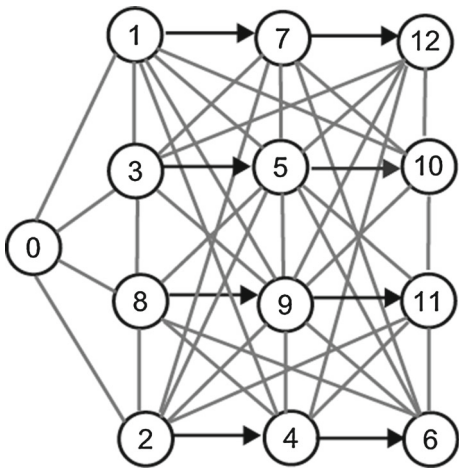

Fig. 4 The problem before a change modelled by a fully connected, partly one-directional, weighted graph (Eaton and Yang 2014)

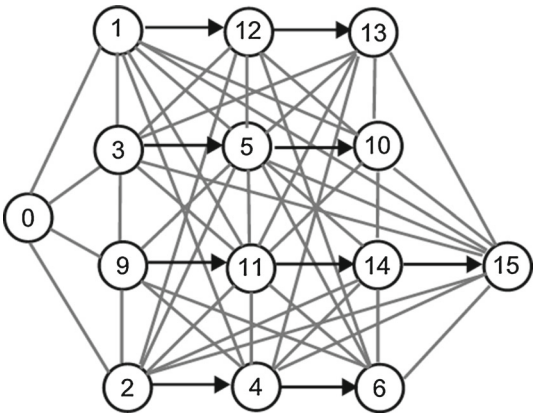

Fig. 5 The problem after a change modelled by a fully connected, partly one-directional, weighted graph (Eaton and Yang 2014)

have been selected, the ant's tour is complete and the ant's solution is evaluated by running it through the simulator. In the case of the P-ACO algorithm, the best ant of the iteration is stored in memory and the pheromone is positively updated for that ant's tour, while in the case of the MMAS algorithm, the best ant is used directly to update the pheromone matrix.

In this implementation, the ants rely only on the pheromone values to guide them while making their choices and the value of β is set to zero. A computationally efficient and effective problem-specific heuristic is not available. The natural choice for a heuristic would be the delay caused by sequencing each train. However, the delay of each train is dependent on the sequence of trains that went before it through the junction and is extremely difficult to establish as it changes for each ant's tour. An advantage of using only the pheromone values to guide the ants is that it reduces the amount of problem-specific knowledge needed to run the algorithm.

After a change, the graph may shrink or grow depending on the number of trains added and the number of trains removed from the problem because they have passed through the junction and are no longer relevant. Figure 5 shows the graph after a dynamic change. Train 8 has been removed from the graph because it has passed through the junction and is no longer available for resequencing. Trains 14 and 15 have arrived on route C and have been added to the graph on the line that corresponds to route C.

However, this graph only allows the ants to resequence the trains through the junction. It is proposed to also allow the ants to resequence the trains at the stations in the hope that this will allow them to resolve the delays introduced at the stations. This is also seen as a first step in extending the algorithms to be able to deal with a much larger area of the railway network.

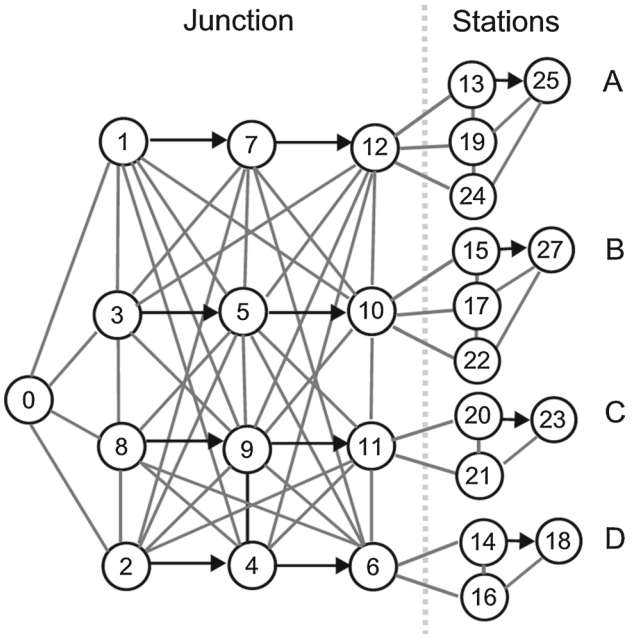

Fig. 6 The directed edge graphs for the junction and associated stations

The stations that feed into the junction have been modelled within the simulator and each train has been allocated to a platform at the station. The original directed edge graph has to be updated to reflect this change. Figure 6 shows that the problem has now been modelled as several linked graphs: one graph for the junction and one for each station.

Each horizontal line in the station graphs represents a platform at the station. As previously mentioned, stations A and B, being much larger stations, have been modelled as having three platforms while stations C and D have only two platforms. Once the ant gets to the end of a horizontal line on the junction section graph, it has access to the graph that represents the trains at the station. An ant can only choose trains in the order that they arrive at the platform as denoted by the one-way arrow. This constraint ensures ants can only make feasible schedules and removes the possibility of trains having to pass over other trains waiting before them on the same platform. As before, the shape of the graphs changes over time as new trains are added and trains that have passed through the junction are removed.

In the version of the algorithm that does not sequence the trains at the stations, denoted non-station sequencing P-ACO (NSS-PACO), trains arriving at the stations are simply added onto the back of the graph as shown in Fig. 5.

5.2 Adapting the memory in P-ACO after a change

Once a change has taken place and extra trains have been added to the simulation, the solutions held in memory are no

longer applicable to the problem because they contain trains that have already passed through the junction and they do not include the extra trains just added. One technique for resolving this problem is to perform a repair on the solutions in memory to make them applicable to the environment after a dynamic change. In NSS-PACO, where trains are not resequenced at the station, this is carried out using the KeepElitist strategy of Guntsch and Middendorf (2001). It involves removing nodes that are no longer of relevance to the ants and adding new nodes where they cause the minimum increase in the objective function.

In the algorithms where station resequencing is performed, it is extremely difficult to execute this repair operation because the algorithm can change the arrival order of the trains at the junction. This means the solutions held in memory after a change may contain trains with different arrival orders to the solution used to make the snapshot of the junction at the point of the dynamic change (see Sect. 5.7). In effect, the memory now contains infeasible solutions. This makes it extremely difficult to repair the ants in memory by just removing the trains that have passed through the junction and adding in the new trains. The arrival order of the trains in the solutions also needs to be amended, which would involve reshuffling the order of all the trains in the solutions. Performing this reshuffle would require extreme domain knowledge and would result in solutions so different from the original solution that the information they contained would be lost.

To get around this problem, it is proposed, in the algorithms that are able to schedule the trains at the stations, to discard all the ants in memory after a change and replace them with new solutions or immigrants. The immigrants can be either elite immigrants, based on the best ever ant, random immigrants, or a mixture of elite and random immigrants. The algorithms that use these schemes are labelled EI-PACO, RI-PACO and HI-PACO, respectively.

Once the new solutions have been made, the fact that the graph representing the problem may shrink or grow after each change means that they still have to be repaired. This is achieved by implementing the KeepElitist strategy.

Each type of immigrant is described in more detail below.

5.3 Random immigrants

Random immigrants are made by constructing solutions where the ants randomly choose the next feasible train node with no regard for the pheromone trails. This results in an ant with a feasible solution that does not represent the existing pheromone trails in anyway. Random immigrants have the advantage of being able to inject diversity into the population but may result in the loss of information.

5.4 Elite immigrants

The elite immigrants are based on the best ant so far. This makes sense in terms of the operation of the algorithm as at each change the best solution for that change period is chosen to run the simulator to provide the snapshot of the state of the junction to pass to the algorithm. Care must be taken when the immigrants are constructed. It is not enough to simply swap around trains in the best ant's solutions as this would very quickly result in infeasible solutions where trains would be scheduled in front of trains that precede them on the track.

Previously, research has created elite immigrants using the adaptive inver-over operator (Mavrovouniotis and Yang 2010, 2011b). However, this algorithm does not preserve the order of the trains in the solution and would have a high probability of creating infeasible solutions.

What is required is a way to generate new solutions from the best ant while preserving the order of trains on each of the routes into the junction. In this paper, this was achieved using a path-preserving local search heuristic to create feasible solutions from the current best solution. The heuristic used is based on the *lpp-3-exchange* search procedure used by Gambardella and Dorigo (2000). The local search algorithm as applied to this problem is described below.

5.5 Path-preserving local search

To explore all possible feasible combinations that can be made from a solution, the path-preserving local search heuristic makes two passes through the solution vector; a forward pass and a backwards pass.

5.5.1 Forward pass

This part of the search starts at the beginning of the train sequence and looks for feasible swaps between the first train and the subsequent trains. To explain, let us consider a feasible train order $\langle 3, 8, 9, 5, 11, 2, 4, 7, 6, 1, 10, 12\rangle$.

First, we would consider a swap between train $\langle 3\rangle$ and, the next train in the sequence, i.e., train $\langle 8\rangle$. Swapping these two trains would have no effect on the feasibility of the solution because they are on different routes into the junction (see Fig. 1). It would result in the feasible train sequence $\langle 8, 3, 9, 5, 11, 2, 4, 7, 6, 1, 10, 12\rangle$. No train in this sequence arrives before any other train on the same route and therefore this solution would make a feasible elite immigrant.

We then consider a swap between train $\langle 3\rangle$ and trains $\langle 8, 9\rangle$ in the original train sequence. This would involve swapping train $\langle 3\rangle$ with trains $\langle 8, 9\rangle$ to give the solution $\langle 8, 9, 3, 5, 11, 2, 4, 7, 6, 1, 10, 12\rangle$. Again, this is a feasible sequence as even though 8 and 9 are both on route C into the junction, their order is preserved and so the solution is feasible. In addition, trains 8 and 9 are on route C, whereas

train 3 is on route B, which means that placing trains 8 and 9 before train 3 does not produce a situation where one train would have to pass through another on the same route to get to the junction.

The next step is to consider a swap between train ⟨3⟩ and trains ⟨8, 9, 5⟩. However, this causes a problem as a swap between these trains results in the solution ⟨8, 9, 5, 3, 11, 2, 4, 7, 6, 1, 10, 12⟩. In this case, train 5 is now before train 3 in the sequence and therefore train 5 would be required to pass through train 3 to reach the junction, which is a physical impossibility. Once an infeasible solution is reached there is no point continuing with any more comparisons using train 3, as every other combination would mean that train 5 would have to pass through the junction before train 3.

At this point, the search for feasible swaps would move onto evaluating swaps between the first two trains in the sequence and the rest of the trains in the sequence, that is, trains ⟨3, 8⟩ with train ⟨9⟩, then train ⟨3, 8⟩ with trains ⟨9, 5⟩ etc. Again, once an infeasible swap is detected, the search using ⟨3, 8⟩ is halted and the search moves onto evaluating swaps between trains ⟨3, 8, 9⟩ and the rest of the train sequence.

The process continues until the evaluation of the final swap between trains ⟨3, 8, 9, 5, 11, 2, 4, 7, 6, 1, 10⟩ and train ⟨12⟩. After this swap evaluation, a backward pass, using a similar search procedure, is performed on the same sequence of trains.

5.5.2 Backward pass

The backward pass is similar to the forward pass but starts at the end of the sequence of trains. It first considers a swap between train ⟨12⟩ and train ⟨10⟩ to make the feasible solution ⟨3, 8, 9, 5, 11, 2, 4, 7, 6, 1, 12, 10⟩. It then evaluates the feasibility of swapping train ⟨12⟩ with trains ⟨1, 10⟩ to give the sequence ⟨3, 8, 9, 5, 11, 2, 4, 7, 6, 12, 1, 10⟩. This is an infeasible sequence as train 12 cannot pass through the junction before train 1. The search for feasible solutions with train ⟨12⟩ is halted and the search for feasible swaps moves onto the next two trains in the sequence, trains ⟨10, 12⟩. Again, this process continues until the final swap of trains ⟨8, 9, 5, 11, 2, 4, 7, 6, 1, 10, 12⟩ with train ⟨3⟩.

Carrying out the search for feasible solutions in this way has the advantage of cutting down on the number of evaluations needed as the procedure is halted as soon as an infeasible solution is found. In addition, in this particular train sequencing problem, the process of determining if a swap is feasible is a relatively quick one as it only needs to be performed on trains that are on the same route.

The elite immigrants are made using the above technique, to find all the feasible local search solutions, and randomly choosing one of those solutions to become the immigrant. This means that the elite solutions are very similar to the best solution used to make the snapshot of the junction after a dynamic change and allow pertinent information to be carried over to the new environment.

5.6 Hybrid immigrants

There is a danger with elite immigrants in that their exploitation of a good solution could limit the algorithm's exploration of the search space. For this reason, a third immigrant scheme is also investigated where the immigrants introduced consist of a half and half mix of elite and random immigrants. The hope is that the addition of random immigrants will encourage the ants to explore the search space whereas the elite immigrants will allow the ants to exploit the previously found good solution.

In EI-PACO, RI-PACO and HI-PACO algorithms, the pheromone matrix is reinitialised after a dynamic change and updated with the solutions of the new ants in memory. If only elite immigrants are used, this will give a pheromone trail that is based on the solution used to create the snapshot of the simulation before a change. This will encourage exploitation of this solution but may have the disadvantage of reducing diversity. However, if random immigrants are used, then this is in effect a restart of the algorithm and will result in the loss of any information from before the change. However, they will have the advantage of introducing diversity into the algorithm, which will encourage the exploration of the search space. Hybrid immigrants may be able to combine the advantages of both these approaches by making use of information before the change but also introducing diversity to encourage further exploration of the search space.

This research takes a slightly different approach to Mavrovouniotis and Yang (2010, 2013) as instead of adding immigrants at every iteration they are only added after a change to equip the algorithm for the new environment.

5.7 Dynamics implementation

Even though the trains and junctions are simulated, this is a real-world problem and requires consideration of how it could be implemented in a real-world delayed-train scenario. The supposition is that after a perturbation, the algorithm is run very quickly in parallel on several computers to find a solution as near optimal as possible in the time available. Ant algorithms are very suitable for running in parallel (Dorigo and Stützle 2004) and doing so would allow a solution to be found in a realistic time.

The sequence of trains in the best solution is then run through the junction until the dynamic change occurs. This change is triggered by the arrival of more timetabled trains. At the point of change, a 'snapshot' is taken of the junctions by the simulator. This records the status of the trains, track and

Table 3 Summary of the algorithms investigated

Abbreviation	Algorithm description	Station sequencing
EI-PACO	P-ACO where the memory is replaced with repaired elite immigrants after a change	Yes
RI-PACO	P-ACO where the memory is replaced with repaired random immigrants after a change	Yes
HI-PACO	P-ACO where the memory is replaced with repaired elite and random immigrants after a change	Yes
NSS-PACO	P-ACO where the ants in memory are retained and repaired with KeepElitist repair operation	No
MMAS	The Max–Min AS algorithm	Yes

junction at that moment in time. The snapshot, plus the new trains, is passed to the P-ACO algorithm, and the algorithm is run again to find the best solution for the new environment. In this way, the algorithm and the simulator are very loosely coupled. The algorithm only acts on the information given to it and does not influence the simulator in any way. This has the advantage that both the simulator and the algorithm can be modified independently of each other.

When the algorithm receives the updated train information, it reconstructs the directed edge graph to reflect the trains in the simulator snapshot. Any trains that have been removed from the snapshot are also removed from the graph and node 0 is reconnected to the next four trains sitting at the junction.

5.8 Handling constraints

One of the constraints of the extended DRJRP is that a train is not allowed to enter the junction before the train in front of it on the same track, and is not allowed to leave the station before the train in front of it on the same platform. This constraint is handled by creating the directed edge graph in such a way that ants can only make a decision as to which train to sequence next from the set of trains that are waiting at the start of the junction or are waiting at the exit of the station (see Sect. 5.1). This graph structure prevents infeasible solutions from being created by the ants and is more computationally efficient than allowing the ants to make unrestricted solutions and then having to identify and discard all the infeasible solutions.

Further constraints of the problem are that trains are not allowed to enter a block section occupied by another train and that trains are not allowed to cross the path of other trains entering or leaving the junction. Both these constraints are dealt with in the simulator (Eaton and Yang 2014). In the first case, a train can only enter the next track section if it is clear of all other trains, and in the second case a simulated interlocking system incorporated in the simulator prevents trains from crossing the path of other trains.

6 Experimental study

An experimental study was carried out to investigate the ability of five ACO algorithms to optimise the extended DRJRP with multiple delays over time. The investigated algorithms are EI-PACO, RI-PACO, HI-PACO, NSS-PACO and MMAS. A breakdown of the characteristics of each of these algorithms can be found in Table 3. The performance of a sixth algorithm based on the FCFS heuristic was also investigated. This heuristic is commonly used by railway controllers to reschedule trains after a perturbation (D'Ariano et al. 2007) and has previously been used by Fan et al. (2012) and Chen et al. (2010) to evaluate train rescheduling algorithms.

6.1 Experimental setting

For all the P-ACO algorithms, the following pheromone parameters were implemented, as recommended by Guntsch and Middendorf (2002). The maximum pheromone value ($\tau_{\max}$) was set to 1, the minimum pheromone value (τ_{init}) was set to $1/n$, where n is the number of nodes, and the pheromone update value to $(\tau_{\max} - \tau_{\text{init}})/k$, where k is the size of the memory. All pheromone levels were initialised to τ_{init}. The other parameters were established by preliminary experimentation. The best combination was found to be 12 ants with a memory size of 6 and a q_0 value of 0.1. After 150 iterations, very little improvement was found to occur in the ants' solutions. Therefore, the algorithm was run for 150 iterations before each dynamic change.

For MMAS, experimental investigation found that the best parameters were $a = 20$, $p = 0.5$, 14 ants and pheromone reinitialisation when there is no change in the best solution for 30 iterations. Again, the algorithm was run for 150 iterations before a dynamic change.

In the case of EI-PACO and RI-PACO, six immigrants were used to replace the memory after a dynamic change, whereas in HI-PACO three elite immigrants and three hybrid immigrants were used.

6.2 Experiment results

As previously mentioned in Sect. 3.2, extra delays over the course of the dynamic problem can only be introduced if the number of trains added to the simulation is of a sufficiently high number to make it feasible to swap the arrival times of the trains at the stations. Therefore, in all the following experiments, the magnitude of dynamic change (m) is eight. The addition of eight trains creates a situation at station D where the arrival order of two trains at the station can be swapped (see Table 1) and a delay introduced. Either one or two additional delays were added at the stations for each of the three different change frequencies (5, 10, 15 min). This made a total of six dynamic scenarios.

Thirty runs were completed for each dynamic environment. The graphs in Fig. 7 show the results for each of the change frequencies when one additional delay was introduced to the scenario at change 1, i.e., after 5, 10 or 15 min of running the simulation. The graphs in Fig. 8 show the results of adding two additional delays, one at change 1 and the second at change 4. The x-axis is the time passed in minutes between each dynamic change, the y-axis is the average fitness of the algorithm in terms of the delay penalty. The scale of the different graphs varies to accommodate the maximum delay penalty.

The corresponding statistical results of two-tailed t-tests with 58 degrees of freedom at a 0.05 level of significance, adjusted using the Bonferroni correction to minimise the possibility of Type-I errors, are given in Table 4. This table shows the results of comparing Algorithm1 $\Leftrightarrow$ Algorithm2, where the symbol '$s+$' indicates that Algorithm1 is significantly better than Algorithm2, while '$s-$' indicates that Algorithm1 is significantly worse than Algorithm2, and the symbols '$+$' and '$-$' indicate that Algorithm1 is insignificantly better or insignificantly worse than Algorithm2, respectively.

The results show that EI-PACO, HI-PACO and NSS-PACO all significantly outperform FCFS on the high and medium frequency delay scenarios and perform better, but usually not significantly better, than FCSS on the low frequency delay scenarios. However, RI-PACO performs significantly worse than FCFS on the high magnitude high frequency changes ($m = 8$, $f = 5$) for both the single station delay and double station delay. It performs better or significantly better than FCFS on the high magnitude medium frequency changes ($m = 8$, $f = 10$) and high magnitude low frequency changes ($m = 8$, $f = 15$).

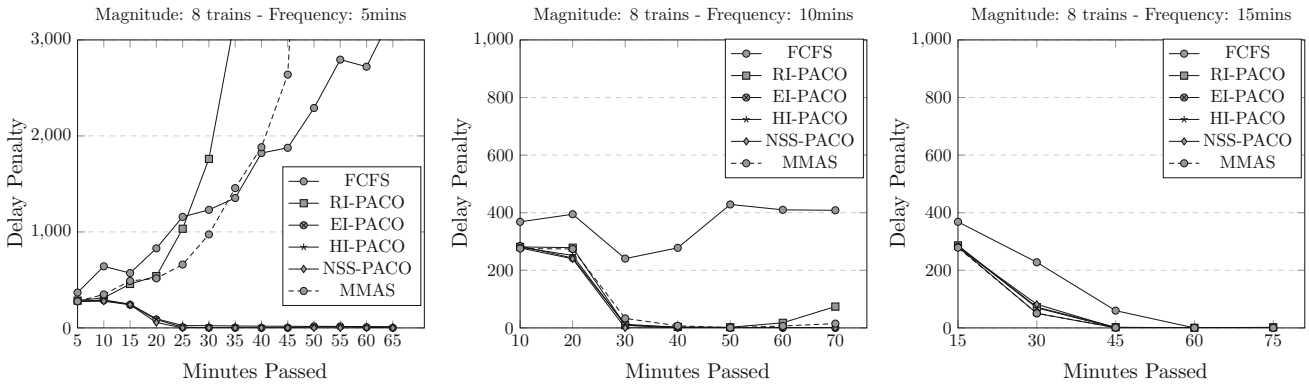

Fig. 7 Experimental results for each of the algorithms for different frequencies of dynamic change and one additional delay at station D

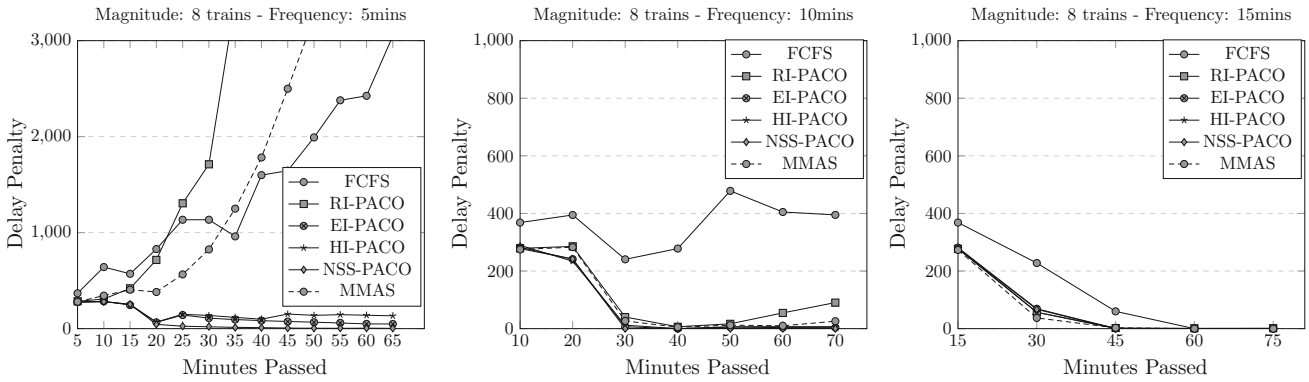

Fig. 8 Experimental results for each of the algorithms for different frequencies of dynamic change and two additional delays at station D

Table 4 The t-test results of comparing the algorithms on different delay scenarios

Algorithms	1 station delay			2 station delays		
	$f \Rightarrow 5$	10	15	$f \Rightarrow 5$	10	15
RI-PACO ⇔ FCFS	$s-$	$s+$	+	$s-$	+	+
EI-PACO ⇔ FCFS	$s+$	$s+$	$s+$	$s+$	$s+$	+
HI-PACO ⇔ FCFS	$s+$	$s+$	+	$s+$	$s+$	+
NSS-PACO ⇔ FCFS	$s+$	$s+$	+	$s+$	$s+$	+
MMAS ⇔ FCFS	−	$s+$	+	−	$s+$	+
RI-PACO ⇔ EI-PACO	$s-$	$s-$	−	$s-$	$s-$	−
RI-PACO ⇔ HI-PACO	$s-$	$s-$	−	$s-$	$s-$	−
RI-PACO ⇔ NSS-PACO	$s-$	$s-$	−	$s-$	$s-$	−
RI-PACO ⇔ MMAS	$s-$	−	−	$s-$	−	−
EI-PACO ⇔ HI-PACO	+	+	+	+	+	−
EI-PACO ⇔ NSS-PACO	−	−	+	$s-$	−	+
EI-PACO ⇔ MMAS	$s+$	$s+$	+	$s+$	$s+$	+
HI-PACO ⇔ NSS-PACO	−	−	+	$s-$	$s-$	+
HI-PACO ⇔ MMAS	$s+$	$s+$	+	$s+$	$s+$	+
NSS-PACO ⇔ MMAS	$s+$	$s+$	+	$s+$	$s+$	−

This is most likely because using random immigrants to initialise the pheromone trails after a change is effectively restarting the algorithm from scratch. All the information from the environment before the change is lost. In a situation when trains are arriving at high frequency, this lack of information from the previous environment seriously hampers the ability of the algorithm to find an effective solution. In such a scenario, trains are added more quickly than trains are lost from the problem, which means that the size of the problem, and the size of the search space, increases after each dynamic change. Restarting the algorithm with such a large search space has the consequence that the ants may not be able to explore it sufficiently in the time available and may be unable to find a good solution. When the frequency of changes is smaller, the time between the addition of new trains to the problem is larger, which means that more trains will have had time to pass through the junction and that the number of trains the ants have to deal with is smaller, with a consequently smaller search space. In this case, even when restarting the algorithm from scratch, the ants are able to explore the whole search space to find an optimal solution. This indicates that in the extended DRJRP, when the search space is large, knowledge carried over from previous environments is especially valuable.

MMAS performs worse than FCFS when the change is of high magnitude and high frequency ($m = 8$, $f = 5$), significantly better than FCFS for the medium frequency changes, and better than FCFS for the low frequency changes, for both the single station delay and double station delay scenarios. This is because MMAS uses only the pheromone evaporation to remove trails that are no longer relevant to the new environment and in addition does not have the advantage of repaired solutions from the previous environment to initialise the amended pheromone matrix and to guide it after a dynamic change. Higher frequency train additions mean a larger search space and the algorithm may struggle to explore it effectively without any guidance. Low frequency train additions mean that the search space is smaller and the algorithm is able to explore it effectively and to find an optimal solution.

RI-PACO performed significantly worse than EI-PACO on the high and medium frequency changes and worse, but not significantly worse, on the low frequency changes. This is because basing the immigrants on the solution used to make the snapshot of the simulation for the next change period means that the solutions are better suited to the new environment than randomly created ants. This may be a peculiarity of this real-world scenario. Due to the need to keep the trains running during the period of disruption, a decision has to be made as to the best solution to choose to run the trains for the next dynamic change period. Thus, the new environment is influenced by the solution used to run the trains for the next change period and immigrants based on that solution have an advantage in the new environment.

RI-PACO also performs significantly worse than HI-PACO on almost all the delay scenarios. This is most likely because the addition of the elite immigrants to the HI-PACO algorithm means that there are some ants well suited to the new environment which helps the search. In fact, RI-PACO appears to be one of the worst performing algorithms; it is also outperformed by MMAS and NSS-PACO on all scenarios. In this dynamic problem, adding random immigrants to the memory after a dynamic change does not appear to be an effective solution.

EI-PACO performs better but not significantly better than HI-PACO in almost all scenarios, which indicates that in this dynamic problem the addition of the random immigrants to the elite immigrants does not significantly improve the performance of the algorithm and may impair it to a small extent. This is most likely again because of the nature of the problem. The fact that the best solution is used to make the snapshot for the next change period means that the environment after the change is similar to the environment before the change and therefore what is required is the exploitative power of the elite immigrants rather than the disruption introduced by the random immigrants.

MMAS performed worse or significantly worse than EI-PACO and HI-PACO on all delay scenarios. Again, this is most likely because MMAS does not have any mechanism to cope with dynamic change apart from the evaporation of redundant pheromone trails and this is not as effective as P-ACO's method of coping with dynamic change by completely removing redundant trails.

The addition of extra delays over the period of investigation appears to have the most marked effect when the frequency of dynamic change is high. When the dynamic change is low or moderate, most algorithms appear to be able to resequence the trains to mitigate the effect of the multiple delays. This suggests that, in a very busy section of the railway track, delays have a more pronounced effect than in a less busy section. Nevertheless, EI-PACO, HI-PACO and NSS-PACO managed to efficiently remove or mitigate the delay in both the one-delay and two-delay scenarios.

Finally, although it was believed that adding the capability to sequence trains at the stations would improve the algorithms' ability to find a solution to the extended DRJRP, this has not been found to be the case. NSS-PACO outperformed all algorithms on almost all the delay scenarios. However, EI-PACO and HI-PACO performed slightly better than NSS-PACO on the low frequency changes for both the single and double station delay scenarios. Low frequency changes mean a smaller search space which suggests that the failure of the station sequencing algorithms to perform as well as the non-station sequencing algorithms may be because station sequencing results in an extended search space with more choices for the ants. When the search space is small, sequencing at the stations may offer a slight improvement. This suggests that the number of ants used may not have been enough to cope with the increase in the size of the search space and additional mechanisms may be necessary to deal with this larger search area.

6.3 Algorithm computation time

The algorithms were executed on a single-core Xeon Woodcrest Linux processor running at 2.83 GHz. The computation times varied according to the number of trains in the problem at a given change. The reason for this is that over 99 % of the computation time was taken up by the time taken to evaluate the ants' solutions in the simulator. The more trains in the problem, the longer the evaluation process. All the investigated ACO algorithms performed similarly in terms of execution time. Therefore, to illustrate the computation time, we will consider the execution time of just one of the algorithms, EI-PACO. Table 5 shows the computation times for this algorithm when eight trains are added every 15 min, while Table 6 shows the computation times when eight trains are added every 5 min. In the tables, the first, second and third lines are the average time, the minimum time and the maximum time over all 30 runs. The last line of the tables shows the percentage of computation time that was taken up by evaluating the ants' solutions in the simulator.

Adding eight trains every 15 min means that many trains have passed out of the problem before the new trains arrive at each change. Therefore, the average execution time does not increase greatly over the course of the changes. For example, for change 1 the average execution time for 150 iterations is 1 min 26 s (0.57 s per iteration) while at change 4 the average execution time is 2 min 6 s (0.84 s per iteration) (see Table 5).

Table 5 Algorithm execution times in min for EI-PACO with $m = 8$ and $f = 15$

Change	1	2	3	4
Average	1.43	1.47	1.84	2.10
Minimum	1.36	1.37	1.63	1.85
Maximum	1.49	1.66	2.08	2.52
Evaluation (%)	99.90	99.90	99.92	99.92

Table 6 Algorithm execution times in min for EI-PACO with $m = 8$ and $f = 5$

Change	1	2	3	4	12
Average	1.97	2.89	3.96	5.07	18.87
Minimum	1.93	2.77	3.77	4.81	17.16
Maximum	2.02	3.03	4.25	5.47	34.82
Evaluation (%)	99.91	99.92	99.93	99.93	99.95

However, when eight trains are added every 5 min, the average execution time at change 1 is 1 min 58 s (0.79 s per iteration), at change 4 it is 5 min and 4 s (2.03 s per iteration) while at change 12, because of the large number of trains in the system, it has risen to an average of 18 min and 52 s (7.55 s per iteration) (see Table 6).

This execution time is, of course, unacceptable in a real world situation. However, ACO is very amenable to being run in parallel (Dorigo and Stützle 2004), which would cut down the computation time considerably and would make it feasible for real-time operation. In addition, the algorithm could be run for less iterations by choosing a different termination criterion, for example, running the algorithm until there has been no improvement in the solution for a predefined number of iterations.

7 Conclusions and future work

Train rescheduling after a perturbation is a challenging task. This paper investigates an even more challenging delay situation where there is not just one but multiple unrelated delays occurring over the time period of the investigation. The extra disruptions are caused by trains being delayed at the stations that feed into the railway junction. To cope with the extra disruption the influence of the ACO algorithms was extended to allow them to reschedule the trains at the stations as well as at the junction.

An experimental study was carried out to investigate the performance of five ACO algorithms on the extended DRJRP

with multiple delays. Three of the algorithms were based on the P-ACO algorithm but with the introduction of immigrants to replace the memory after a change. The fourth algorithm was based on MMAS, which has no in-built mechanism for adapting to change apart from the evaporation of pheromones trails. The fifth algorithm was a P-ACO algorithm that does not have the ability to reschedule the trains at the stations but simply deals with them in the order that they arrive at the junction. In addition, a FCFS heuristic was used to investigate how the trains might be sequenced in a real-world situation.

It is apparent from the results that replacing the memory with elite immigrants in the extended DJRJP is an effective solution when the ants in memory cannot be modified to make them feasible in the new environment. In addition, in this dynamic problem, random immigrants were found to be unsuitable to replace the ants in memory when changes were of a high magnitude and a high frequency. The larger search space appears to demand the knowledge carried over from previous environments.

Surprisingly, adding the ability to sequence trains at the stations was not beneficial to this set of dynamic problems. This may be because the extra decisions that the ants have to make increases the size of the search space and the ants struggle to explore it adequately. This suggestion is supported by the fact that when the search space is relatively small in the low change frequency scenarios, algorithms that sequence the trains at the stations (EI-PACO and HI-PACO) perform slightly better than algorithms without station sequencing (NSS-PACO).

This work raises some exciting possibilities for future work. The first is to investigate whether MMAS can be adapted to improve its performance in fast changing dynamic environments, for example, by the addition of elite immigrants that help to guide its search after a dynamic change.

Second, it would be interesting to explore the effect of adding more ants or random immigrants to the algorithms that resequence the trains at the stations to see if this could improve their exploration and consequently their performance on high frequency changes, where the search space is very large between changes. To maximise the use of resources, it would be worthwhile to investigate adding the number of additional ants in proportion to the increase in the size of the problem. Although it did not perform exceptionally well in this set of experiments, station sequencing in addition to junction sequencing is worth pursuing because it is one of the first steps in expanding the algorithms to take in a larger area of the railway network.

Finally, the next stage in the investigation of railway rescheduling problems is to change the DRJRP into a dynamic multi-objective problem by the addition of a second objective, e.g., minimizing the fuel cost. As many real-world scheduling problems are multi-objective as well as dynamic, this will be an important step in the application of evolutionary computation to real-world train scheduling and rescheduling problems.

Acknowledgments The authors would like to thank Dr Simon Coupland and Dr Steve Ackland at the Centre for Computational Intelligence, De Montfort University, UK for their help and advice on the physics of train simulation. They would also like to thank Dr Dave Kirkwood at the University of Birmingham, UK for sharing his expertise on train simulations and for supplying the power and resistance tables used in the simulation.

Compliance with ethical standards

Conflict of interest The authors declare that they have no conflict of interest.

References

Abbas-Turki A, Zaremba E, Grunder O, El-moudni A (2011) Perfect homogeneous rail traffic: a quick efficient genetic algorithm for high frequency train timetabling. In: Proceedings of the 14th international IEEE conference on intelligent transportation systems (ITSC'11). IEEE, pp 1495–1500

Chen L, Schmid F, Dasigi M, Ning B, Roberts C, Tang T (2010) Real-time train rescheduling in junction areas. Proc Inst Mech Eng Part F J Rail Rapid Transit 224(6):547–557

D'Ariano A, Pacciarelli D, Pranzo M (2007) A branch and bound algorithm for scheduling trains in a railway network. Eur Jf Oper Res 183(2):643–657

Dorigo M, Stützle T (2004) Ant Colony Optim. Bradford Company, Scituate

Eaton J, Yang S (2014) Dynamic railway junction rescheduling using population based ant colony optimisation. In: Proceedings of the 2014 UK workshop on computational intelligence (UKCI'14), pp 1–8

Fan B, Roberts C, Weston P (2012) A comparison of algorithms for minimising delay costs in disturbed railway traffic scenarios. J Rail Transp Plan Manag 2:23–33

Fang W, Yang S, Yao X (2015) A survey on problem models and solution approaches to rescheduling in railway networks. IEEE Trans Intell Transport Syst. doi:10.1109/TITS.2015.2446985

Gambardella LM, Dorigo M (2000) An ant colony system hybridized with a new local search for the sequential ordering problem. INFORMS J Comput 12(3):237–255

Gorman MF (1998) An application of genetic and tabu searches to the freight railroad operating plan problem. Ann Oper Res 78:51–69

Grefenstette JJ (1992) Genetic algorithms for changing environments. In: Proceedings of the 2nd international conference on parallel problem solving from nature (PPSN II), vol 2, pp 137–144

Guntsch M, Middendorf M (2001) Pheromone modification strategies for ant algorithms applied to dynamic TSP. In: EvoWorkshops 2001: applications of evolutionary computing. Lecture notes in computer science, vol 2037, pp 213–222

Guntsch M, Middendorf M (2002) Applying population based ACO to dynamic optimization problems. In: Proceedings of the 3rd international workshop on ant algorithms. Lecture notes in computer science, vol 2463, pp 111–122

Ho T, Yeung T (2000) Railway junction conflict resolution by genetic algorithm. Electron Lett 36(8):771–772

Khan M, Zhang D, Jun MS, Li ZJ (2006) An intelligent search technique to train scheduling problem based on genetic algorithm. In: Proceedings of the 2006 international conference on emerging technologies (ICET'06), pp 593–598

Mavrovouniotis M, Yang S (2010) Ant colony optimization with immigrants schemes in dynamic environments. In: Proceedings of the 11th international conference on parallel problem solving from nature (PPSN XI). Lecture notes in computer science, vol 6238, pp 371–380

Mavrovouniotis M, Yang S (2011a) A memetic ant colony optimization algorithm for the dynamic travelling salesman problem. Soft Comput 15(7):1405–1425

Mavrovouniotis M, Yang S (2011b) Memory-based immigrants for ant colony optimization in changing environments. In: EvoApplications 2011: applications of evolutionary computation. Lecture notes in computer science, vol 6624, pp 324–333

Mavrovouniotis M, Yang S (2013) Ant colony optimization algorithms with immigrants schemes for the dynamic travelling salesman problem. In: Yang S, Yao X (eds) Evolutionary computation for dynamic optimization problems, studies in computational intelligence, vol 490. Springer, Berlin, pp 317–341

National Rail Enquiries (2015) Stations services and facilities. http://www.nationalrail.co.uk/stations_destinations/. Accessed 5 Nov 2014

Qin S, Zhou LS, Yue YX (2010) A clonal selection based differential evolution algorithm for double-track railway train schedule optimization. In: Proceedings of the 2nd international conference on advanced computer control (ICACC'10), vol 1, pp 155–158

Stützle T, Hoos H (1997) MAX–MIN ant system and local search for the traveling salesman problem. In: Proceedings of the 1997 IEEE international conference on evolutionary computation, pp 309–314

Tormos P, Lova A, Barber F, Ingolotti L, Abril M, Salido MA (2008) A genetic algorithm for railway scheduling problems. In: Metaheuristics for scheduling in industrial and manufacturing applications, pp 255–276. Springer, New York (2008)

Van Der Zwaan S, Marques C (1999) Ant colony optimisation for job shop scheduling. In: Proceedings of the 3rd workshop on genetic algorithms and artificial life (GAAL'99)

Yang S (2005) Memory-based immigrants for genetic algorithms in dynamic environments. In: Proceedings of the 2005 conference on genetic and evolutionary computation, pp 1115–1122

Yang S (2007) Genetic algorithms with elitism-based immigrants for changing optimization problems. In: EvoWorkshops 2007: applications of evolutionary computing. Lecture notes in computer science, vol 4448, pp 627–636

Yang S (2008) Genetic algorithms with memory-and elitism-based immigrants in dynamic environments. Evol Comput 16(3):385–416