CASPT 2018

Solving periodic timetabling problems with SAT and machine learning

Gonçalo P. Matos · Luís M. Albino · Ricardo L. Saldanha · Ernesto M. Morgado

Abstract In this research work we address periodic timetabling, namely the optimisation of public transport timetables with respect to travel time using Boolean Satisfiability Problem (SAT) and reinforcement learning approaches.

Some works already done in the field of railway timetabling propose solutions to the optimisation problem using Mixed Integer Linear Programming (MILP), genetic algorithms, SAT, the modulo network simplex, among other techniques. In this work, we propose a novel approach based on reinforcement learning and multiagents which also uses a SAT solver to get optimised solutions with respect to the travel time, a combination of techniques which (to our knowledge) has never been applied in this field.

Finally, we present promising results which show that our approach applied to real world data performs better than existing SAT approaches and even outperforms the current state-of-the-art algorithms (based on the modulo network simplex, mixed integer programming and heuristics) on some problems.

Keywords Periodic timetabling · Optimisation · Periodic Event Scheduling Problem · SAT · Reinforcement Learning

Mathematics Subject Classification (2000) 90B06 · 68T20 · 68T05

Gonçalo P. Matos
Instituto Superior Técnico and **SISCOG, Sistemas Cognitivos SA**
E-mail: goncalo.p.matos@tecnico.ulisboa.pt

Luís M. Albino
SISCOG, Sistemas Cognitivos SA
E-mail: lmalbino@siscog.pt

Ricardo L. Saldanha
SISCOG, Sistemas Cognitivos SA
E-mail: rsaldanha@siscog.pt

Ernesto M. Morgado
Instituto Superior Técnico and **SISCOG, Sistemas Cognitivos SA**
E-mail: ernestomorgado@tecnico.ulisboa.pt

1 Introduction

Timetabling is a major step in the overall process of planning the operations in a public transport network, and is closely related to other planning activities such as line planning, and vehicle and crew scheduling.

In this work, we focus on periodic timetabling for public transport. In a periodic timetable, each event (e.g. a train departure from a certain station) scheduled for some time instant actually occurs recurrently at that same instant, at each period (e.g. every hour).

Concerning periodic timetables, one can be interested in finding a feasible timetable or an optimised one, according to some criteria. Henceforth, we will refer to these problems as Periodic Timetable Satisfaction Problem (PTSP) and Periodic Timetable Optimisation Problem (PTOP), respectively.

Over the last years, several approaches were studied for modelling and solving either the PTSP or the PTOP, namely constraint propagation [de Waal (2005)], MILP [Peeters (2003)], heuristic search [Ingolotti et al (2006); Staereling (2015); Goerigk and Liebchen (2017)], and genetic algorithms [Nachtigall and Voget (1997)]. A well-known approach for modelling the PTSP, called the Periodic Event Scheduling Problem (PESP), was introduced by Serafini and Ukovich (1989).

Since PESP models were proposed, several approaches were used to solve the problem with increasing success. Recent works [Großmann et al (2012a), Großmann (2011), Kümmling et al (2015)] suggest that currently the best known approach for solving the PESP and the PTSP is to model them as a SAT[1]. Essentially, the PESP problem is encoded as a propositional formula and solved using a state-of-the-art SAT solver.

Besides solving the PTSP, which is already challenging *per se*, solving the PTOP with an objective function such as minimising the travel time is also a major area of interest. Several works [Peeters (2003), Ingolotti et al (2006), Goerigk and Liebchen (2017)] have been done in this area, some of which also used SAT approaches [Großmann et al (2012b), Großmann et al (2015), Gattermann and Nachtigall (2016)].

This paper presents new developments on a research [Matos et al (2017)] performed in the context of a Master's thesis [Matos (2018)] at Instituto Superior Técnico[2] and SISCOG[3]. In particular, our goal is to optimise the passengers' total travel time (a linear objective function defined by Peeters (2003)) using incremental calls to a SAT solver, on real-world, large-sized timetabling problems.

We propose a machine learning based method to guide the search, something which (to our knowledge) was never done before in this domain. We evaluate the performance of our solution by comparing our achieved results (in terms of objective value) on a publicly available set of problems with the ones obtained by other authors of state-of-the-art approaches to the PTOP. Furthermore, we also compare it to our previously developed approach [Matos et al (2017); Matos

[1] The Boolean Satisfiability Problem (SAT) is to decide whether a propositional formula is *satisfiable* or not. It is a well known and long time studied problem [Cook (1971)]

[2] The University of Lisbon engineering faculty.

[3] SISCOG - Sistemas Cognitivos, SA (`http://www.siscog.pt`), a Portuguese software company specialised in solutions for creating and updating operational plans of companies that provide regular transportation services, such as passenger railway operators and metro systems.

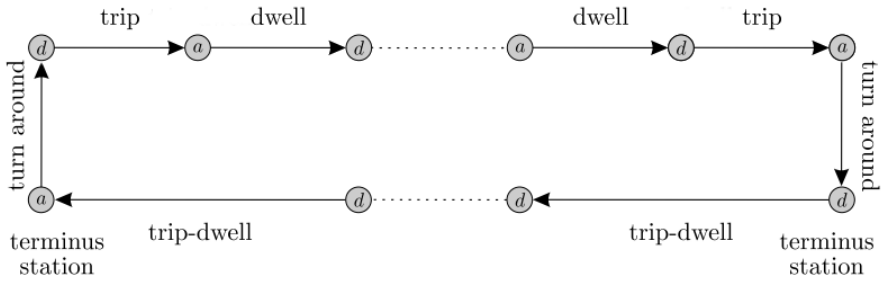

Fig. 1 A sample PTSP represented in the form of a graph. The events (nodes) correspond to periodic departures (d) and arrivals (a) to stations, whereas the constraints (arcs) model the elapsed times between those events. The edges on the bottom are the result of merging a trip and a dwell arc, which is possible if one of such constraints has a fixed value. *Adapted from Peeters (2003).*

(2018)], based on incremental SAT calls, a binary search procedure and an heuristic to compute better upper bounds.

On Section 2, we define the problem that we will be solving, after introducing the necessary background. Sections 3 and 4 describe, respectively, the proposed solution for this problem and promising results which show that our approach performs better than the currently state-of-the-art approaches on some instances. Finally, we present conclusions (Section 5).

2 Definitions

2.1 The Periodic Timetable Satisfaction Problem (PTSP)

A Periodic Timetable Satisfaction Problem (PTSP) can be modelled as a Periodic Event Scheduling Problem (PESP) [Serafini and Ukovich (1989)] whose *events* correspond to periodic train/bus departures (arrivals) from (to) specific stations in the context of specific train/bus trips (see Figure 1). The goal is to find a schedule for such a set of periodically recurring events, subjected to a set of constraints (for instance, safety headways, trip and dwell times) under periodic time windows, or to conclude that no such assignment exists.

Any constraint between a pair of events (i, j), given the corresponding scheduled times v_i and v_j, has the form:

$$v_j - v_i + Tp_{ij} \in [l_{ij}, u_{ij}] \tag{1}$$

where p_{ij} is an auxiliary integer decision variable, T is the period of the timetable, and l_{ij} and u_{ij} are respectively the lower and upper bound, i.e., the minimum and maximum admissible values for that constraint.

2.2 The Periodic Timetable Optimisation Problem (PTOP)

The Periodic Timetable Optimisation Problem (PTOP) is to find an optimal solution for a PTSP, that is, to select from the set of all feasible solutions to a PTSP one that minimises (or maximises) a provided objective function.

One of the objective functions presented by Peeters (2003), which will be the main reference for this paper, is minimising the total travel time. This objective function depends on *process times*, which can be defined as:

$$x_{ij} \equiv v_j - v_i + Tp_{ij} \tag{2}$$

Connection, dwell and trip times influence the total passenger travel time. For instance, each minute of dwell above the minimum dwell time (denoted by l_{ij}) adds to the minimum possible travel time. So, these excess process times are given by $(x_{ij} - l_{ij})$. If we define a weight w_{ij} for each constraint, then we want to minimise:

$$\sum_{(i,j)\in A} w_{ij}(x_{ij} - l_{ij}) \tag{3}$$

3 Solution architecture

3.1 Overview

In this work, we propose a new approach to address the PTOP which combines SAT with machine learning to get optimised solutions iteratively for larger problems than those addressed by our previous approach [Matos et al (2017, 2018)], based on a binary search procedure and supported by an heuristic. On both approaches we use a SAT solver and model the PTOP in SAT using the order encoding approach suggested by Großmann (2011).

The overall architecture is summarised in Figure 2. The block *"Create PEN graph"* consists in transforming the provided PTOP into a Periodic Event Network (PEN) graph using the tools already available in ONTIME[4]. The block *"Compaction"* represents a set of graph operations that were already implemented in ONTIME in order to reduce the PEN as much as possible before actually starting solving the problem (for instance, the contraction of edges with a fixed length, or the merge of multiple parallel edges (constraints) between the same pair of events). The *"Order encode PTSP"* procedure consists in simply encoding the satisfaction problem with the order encoding (as explained in Section 3.2). The block *"Encode objective function"* represents the encoding of the optimisation part, which will be explained in Section 3.3. By separating the encoding of the satisfaction and optimisation parts of the problem we simplify the process of testing the model with different objective functions, which is something that can be useful to perform extensions of this work (see Matos (2018)). The block *"Incremental SAT optimisation"* is a black box representing any method to find optimised solutions to the PTOP, namely our approach based on machine learning (ML) (Section 3.5 and Section 3.6) or an alternative approach developed by us in a previous work

[4] One of SISCOG Suite's products. For more information, access the ONTIME product page on SISCOG's web site (`http://www.siscog.pt`).

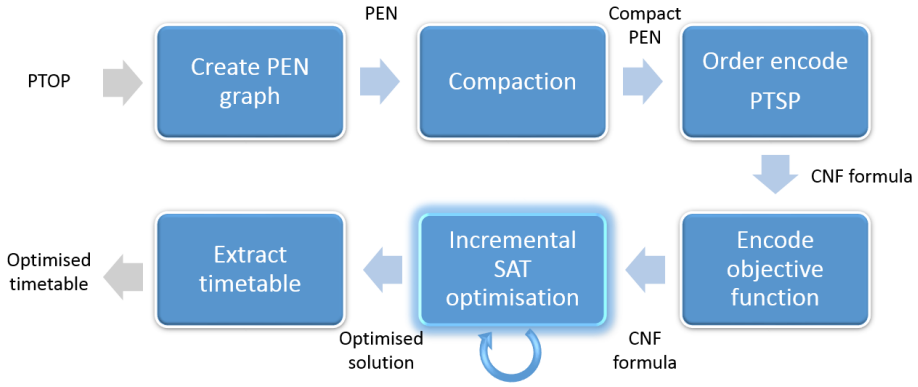

Fig. 2 Overview of the proposed solution architecture.

(Section 3.4). Finally, *"Extract timetable"* consists in getting a PTOP solution (i.e., an assignment of schedule times to events in the PEN graph) from the optimised SAT solution.

To improve a solution with respect to the objective function (Equation 3) we have to reduce some process times. However, in order to be able to reduce a process time, one usually needs to *increase* another process time to satisfy the constraints and get a SAT answer, since implicit relations exist between PTOP constraints. The great difficulty of this process is to identify which processes to decrease and which to increase because we don't know explicitly those relationships.

3.2 Encoding PTSP into SAT

Our first step towards solving the PTOP, either with our current approach based on machine learning or with our former approach based on a binary search [Matos et al (2017, 2018)], is to encode the problem into SAT. Our choice for this step was to encode the underlying PTSP problem using the order encoding, as described in Großmann (2011) for PESPs. This approach assumes that the domains of all variables are finite and ordered. The description presented here is based on Großmann (2011) and Kümmling et al (2015).

Let i be an event with domain D. We introduce a set of auxiliary propositional variables $q_{i,k}$, $\forall k \in D$, such that $q_{i,k}$ represents the proposition $v_i \leq k$, i.e., the scheduled time v_i for event i is lower or equal to k. Hence, $q_{i,k} = true \Rightarrow v_i \leq k$. Since in a PTSP all variables (event times) have domain $[0, T[$, there is an implicit ordering relation between the q variables given by the ordering of the integer numbers. So, we encode event i, taking into account these ordering constraints, as:

$$encode\_event : i \mapsto (\neg q_{i,-1} \wedge q_{i,T-1} \bigwedge_{k \in [0,T-1]} (\neg q_{i,k-1} \vee q_{i,k})) \tag{4}$$

This ensures that, whenever $q_{i,k-1}$ is $true$, $q_{i,k}$ is also $true$, because indeed $\forall i, k \in \mathbb{Z}, v_i \leq k-1 \implies v_i \leq k$.

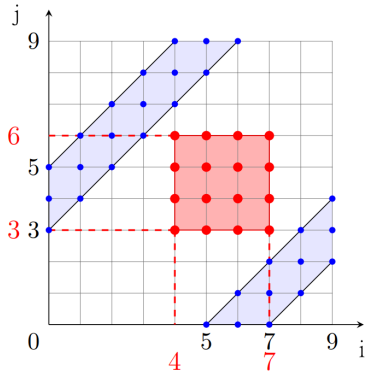

Fig. 3 Feasible (blue stripes) and infeasible (white) regions for constraint $[3,5]_{10}$, with $T = 10$, between events i and j. The red square shows an infeasible rectangle. The generated clause for this rectangle, according to Eq. 5, is $(\neg q_{i,7} \vee q_{i,3} \vee \neg q_{j,6} \vee q_{j,2})$. *Adapted from Großmann (2011).*

In order to encode a constraint, we can compute a set of rectangular regions containing all the infeasible values (Figure 3), using an auxiliary function ζ described by Großmann (2011). Then, for each excluded rectangle $r = ([x_1, x_2] \times [y_1, y_2])$ related with the constraint between the pair of events $(i, j) \in A$, we apply the following function which produces a single clause:

$$encode\_rec : i, j, ([x_1, x_2] \times [y_1, y_2]) \mapsto (\neg q_{i,x_2} \vee q_{i,x_1-1} \vee \neg q_{j,y_2} \vee q_{j,y_1-1}) \quad (5)$$

So, each PTSP constraint can be encoded into SAT with the function:

$$encode\_constraint : (i, j) \mapsto \bigwedge_{r \in \zeta(l_{ij}, u_{ij})} encode\_rec(i, j, r) \quad (6)$$

Finally, the complete PTSP model is translated into SAT by applying the previous functions to all its events and constraints:

$$\bigwedge_{i \in N} encode\_event(i) \bigwedge_{(i,j) \in A} encode\_constraint(i, j) \quad (7)$$

Großmann (2011) also shows that it is possible to extract exactly one schedule time for each event i given a solution (called interpretation I) returned by the SAT solver, with respect to the previous encoding, using a function that we present below, simplified and adapted to our case:

$$v_i = \xi_i(I) = k \quad \text{such that } q_{i,k-1} = false \wedge q_{i,k} = true, k \in [0, T-1] \quad (8)$$

Essentially, the SAT solution will have, for each event i, a sequence of q variables with value $false$ followed by a sequence of q variables with value $true$, since Equation 4 forbids alternating $true$ and $false$ values. By looking at the transition point one concludes that $v_i > k - 1$ and $v_i \leq k$, so actually $v_i = k$.

3.3 Encoding the objective functions into SAT

We describe next how we encode the objective function, which is (in the case of this paper) to minimise the total travel time. This encoding was much inspired by the work of Gattermann and Nachtigall (2016).

With that purpose, we add auxiliary variables $\tau_{ij}^k \in \{0,1\}$, for all constraints $(i,j) \in A$ and $k \in \{0,...,u_{ij} - l_{ij}\}$, representing the *difference* between the tension x_{ij} and the lower bound l_{ij} for the constraint, such that the variable τ_{ij}^k is true if the proposition $v_j - v_i - l_{ij} + T \cdot p_{ij} \geq k$ holds, being $p_{ij} \in \mathbb{Z}$ auxiliary decision variables.

Then, we impose order encoding constraints on those τ variables:

$$(i,j) \mapsto (\tau_{ij}^0 \wedge \neg\tau_{ij}^{u_{ij}+1} \bigwedge_{k \in \{1,...,u_{ij}+1\}} (\neg\tau_{ij}^k \vee \tau_{ij}^{k-1})) \tag{9}$$

and encode the following implications:

$$(v_j - v_i - l_{ij} + T \cdot p_{ij} \geq k) \implies \tau_{ij}^k \tag{10}$$

which can be written as follows:

$$(v_j - v_i + T \cdot p_{ij} \in [l_{ij}, k + l_{ij} - 1]) \vee \tau_{ij}^k, \tag{11}$$

where the first part of the disjunction can be encoded in SAT with the use of Equation 6.

Finally, we want to minimise the total travel time, i.e.:

$$\sum_{(i,j) \in A} (x_{ij} - l_{ij}) = \sum_{(i,j) \in A} \left(\sum_{k=1}^{u_{ij}-l_{ij}} \tau_{ij}^k \right) \tag{12}$$

The right-hand side of this equation is called a pseudo-boolean expression, because it is a sum of boolean variables. Several methods exist in the literature to convert Pseudo-Boolean (PB) constraints into propositional formulas in Conjunctive Normal Form (CNF) in order to be solved with a SAT solver. Moreover, specialised solvers — called PB solvers — exist to solve and/or optimise problems modelled with PB expressions without the need for converting them into clausal form first (see, for instance, Manthey and Steinke (2012)).

3.4 Optimising with SAT and a binary search approach

Before we describe how we addressed the *"Incremental SAT optimisation"* step of our solution architecture (see Section 3.1), we describe in brief terms a first attempt to address this step, which, as explained in Matos et al (2017), consisted in performing a binary search on the value of the objective function, supported by a lower bound (LB) and an upper bound (UB) that we compute beforehand.

As we saw before, our goal is to minimise an objective function in the form of a pseudo-boolean expression. Henceforth, we model the function as a constraint instead, with the general form:

$$\sum_l w_l \cdot b_l \leq C \tag{13}$$

being C an integer constant, b_l the boolean variables and w_l their respective coefficients (weights). Then, we can apply a binary search procedure to replace C by actual integer values and perform iterative SAT calls to tighten the values of LB and UB until the optimal solution is found (LB = UB). In each iteration, depending on whether the problem is feasible or infeasible, the following tightenings can be done: UB $= C$ or LB $= C + 1$, respectively.

In order to compute a better UB to begin with the binary search, and thus improve the solving time, we developed a new heuristic which uses information from UNSAT cores[5]. Furthermore, we also used this heuristic alone to actually obtain optimised solutions for large PTOPs which the binary search could not address due to memory and performance limitations. Further details on this approach, including the heuristic pseudo-code, are given in Matos (2018).

3.5 Modelling the problem with agents and reinforcement learning

Our first attempt to address the *"Incremental SAT optimisation"* step of our solution architecture (see Section 3.1) suffers from two drawbacks. First, the binary search + heuristic procedure cannot be used to solve medium or large-sized problems due to performance and memory problems. Second, the heuristic procedure alone, although it can solve large-sized problems, obtains poor solutions due to the fact that it ignores the relations between the PTOP constraints implicitly present in the problem and thus is too greedy.

Therefore, after the first attempt, we developed an approach based on a multi-agent, reinforcement learning system to solve the PTOP trying to "guess" the relationships between constraints based on data from hundreds of experiments. In the end, this method aims to get a good PTOP solution (although the optimal one cannot be guaranteed), performing a proper balance — inferred by *experience* — between decreased and increased process times. The results obtained with this new approach were successful enough to position it among the state-of-the-art, as we will see in Section 4.1.

In this approach, we associate to each process time x_{ij} an *agent* whose possible actions are the differences between the possible values that x_{ij} may assume and its corresponding lower bound, that is, the integer values in $[0, u_{ij} - l_{ij}]$. Henceforth, each agent will determine a set of SAT assumptions to be added on top of the original formulation, which will further constrain the problem and guide the search, by deciding which value each tension should have and consequently fixing the appropriate values for some τ variables[6] (more on this on Section 3.6).

The architecture of this approach is based on a *decide–act–observe reward* cycle such that all agents are given the opportunity to decide on their action, then all the actions are performed simultaneously and influence the *environment* — i.e., a solution for the PTSP is produced with some cost —, and finally all the agents receive feedback on how well they performed so that they can improve and learn from their mistakes and/or from their good decisions.

[5] An UNSAT core is a sub-set of clauses from the SAT problem which is unsatisfiable. UNSAT cores can give insightful information about the problem being solved, and thus techniques have been developed to extract and exploit them (for instance, Fu and Malik (2006)).

[6] Defined in Section 3.3.

Agents receive feedback by means of a *reward* function that we must define, which should provide an individual value for each agent based on how good or bad its decision was. The reward function developed in this work is a function with two terms:

– A **global term** — which rewards the agent accordingly to the overall solution cost, i.e., the lower the solution cost is relative to the initial[7] UB the greater is the reward the agent receives. This is the most important term, therefore it receives a greater weight in the function, since this is the term which instructs the agents to *cooperate* for the common welfare of producing a better PTOP solution, leading some agents perhaps to choose a higher edge tension value so that others can choose it lower. Notice that this term is equal for all agents, since it depends only on the solution cost — which is a measure of the performance of the group of agents as a whole — and not on the individual decisions;

$$global\_term = \frac{UB_{initial} - solution\_cost}{UB_{initial}} \tag{14}$$

– An **individual term** — which encourages the agent to choose actions which correspond to lower tension values since, in general, the lower the edge tensions are the lower the solution cost is. However, this term leads to selfish and greedier, less cooperative agents, thus it gets a small weight on the reward function;

$$individual\_term = \frac{agent.NrActions - agent.action\_chosen}{agent.NrActions} \tag{15}$$

In Equation 15 object-oriented style is used to refer to the number of different actions available and to the chosen action of some particular agent. We consider that the actions — which correspond to integer edge tensions in $[0, u_{ij} - l_{ij}]$ — are ordered and numbered from 1 to $u_{ij} - lij + 1$, being action 1 the one with the lowest tension (0). This means that each action a of the agent associated with the edge (i, j) satisfies the condition:

$$a \in [1, u_{ij} - l_{ij} + 1] \tag{16}$$

The complete reward function used in this work is the following:

$$reward = global\_term * (1 - 0.3 * w) + individual\_term * (0.3 * w); \tag{17}$$

where w is the relative importance of the agent, computed as $w = \frac{w_{ij}}{\max_{(i,j)\in A} w_{ij}}$, being w_{ij} the weight of the constraint represented by the considered agent in the PTOP.

[7] The initial (naive) UB is the sum of the upper limits for all constraints.

3.6 The learning algorithm

The learning algorithm we implemented is based on the Upper Confidence Bound (UCB1) [Auer et al (2002)] action selection method, with some ideas from the Q-learning algorithm [Watkins and Dayan (1992)] along with other small improvements that take advantage from knowledge from the structure of a PTOP problem.

In each cycle, every agent is given the opportunity to choose an action. Then, the actions of all agents are combined, i.e. each agent sets an upper bound to the corresponding process time, by means of the assumptions feature of the SAT solver. Next, the SAT solver is called to test whether a solution to the PTOP with such upper bounds for the process times exists and, in the affirmative case, to obtain a model (a feasible assignment of times to the events).

According to the answer returned by the SAT solver, a reward is computed and provided for each agent. If the answer was SAT, then a reward is computed according to the aforementioned reward function. If, otherwise, the answer was UNSAT, then an UNSAT core is extracted and the agents corresponding to the constraints involved in the core receive 0 as a reward (a penalty), whilst the remaining agents do not receive any reward at all and hence do not learn anything (as if their action never occurred) because we cannot know for sure whether their decisions were good or bad. The introduction of knowledge from the UNSAT cores is one of the particularities of our learning algorithm.

Agents maintain a set of values, called the Q-values, which estimate the *total expected reward* for each action, i.e., estimate the quality of each action in terms of the reward it is expected to produce in the long term. In our approach — unlike the traditional Q-learning algorithm — these Q-values do not depend on a *state*, so our agents learn a very simple policy which is a single action supposed to be always the best independently of the state of the environment. With this simplification we reduce the Q-values table, whose size should be $states \times actions$, to a single vector whose size is equal to the number of actions available to the agent. This simplification allows us to reduce the amount of memory and time necessary for the agents to learn and converge to a stabilised behaviour. In fact, without this simplification, we could not even fit our huge instances in memory.

After an initialisation period in which each agent tries all its actions once, the action for the time-step t is chosen according to the formula:

$$action_t = \underset{a \in agent.Actions}{\operatorname{argmax}} \quad QValues[a] + \sqrt{\alpha * \frac{log(t)}{NumTrials[a]}} \tag{18}$$

where $NumTrials$ is the total number of times the action was performed in the past, and α is the diversification parameter of the algorithm. In our experiments, we found that the value 0.001 worked well for this parameter. As Equation 18 shows, an action is more likely to be chosen if it has provided good results in the past or if it has not been tried for a long time.

When an agent chooses an action a, we fix a set of τ variables with the value $false$ in the SAT solver assumptions which actually state that the value for that constraint *must not be greater* than a, instead of stating that the value must be exactly a. Because of that slight relaxation, which is also a unique feature of our learning algorithm, sometimes the SAT solver finds a solution with values for the constraints which are in fact lower than those selected by the agents. In those cases

we are lucky to find a solution which is better (and never worse) than the selection made by the agents, so we want them to learn this better decision instead. In order to do so, we compute the reward not only for the single action that the agent has chosen, but also for all actions from the *actual* action found by the SAT solver (which is always equal or lower) up to the chosen one. Since the individual term of the reward is dependent on the action and encourages lower actions (see Equation 15), the actual (lower) action found by the SAT solver is given a higher reward than the one initially chosen by the agent, so we are in fact reinforcing the better one.

After executing an action a and incrementing the corresponding $NumTrials(a)$ counter, an agent updates the Q-value for the action a, given a reward, using the formula:

$$QValues[a]' = \frac{QValues[a] * (NumTrials[a] - 1) + reward}{NumTrials[a]} \tag{19}$$

which essentially keeps an average of the rewards obtained in the past when that action was chosen.

After thousands of iterations, the agents' behaviour converges to an optimised solution (see Figure 4).

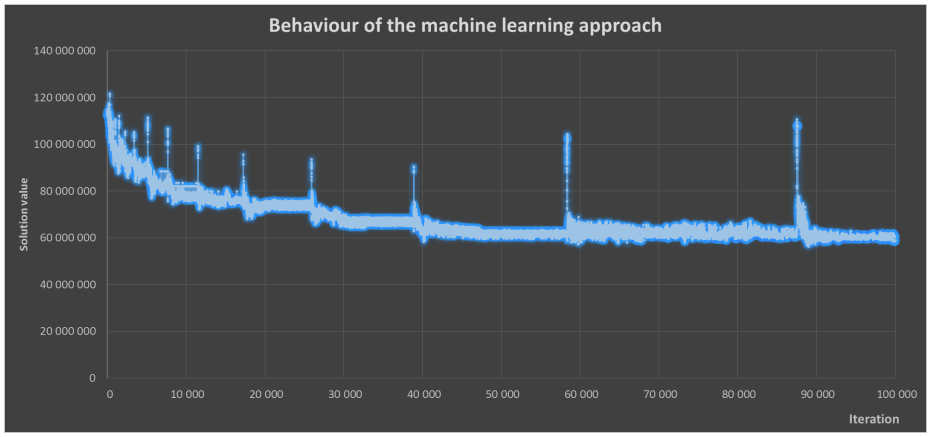

Fig. 4 Behaviour of the machine learning approach when solving the R4L1 problem from PESPlib. Solution values (y axis) across algorithm iterations (x axis). Lower values are better.

We also introduced the notion of *active* and *inactive* agents in our algorithm in order to control its complexity. An *active* agent is one which is able to actually decide its action in each iteration and observe the corresponding reward. On the other hand, an *inactive* agent is not given the opportunity to choose an action — the corresponding PTOP constraint does not get a fixed value and so it is freely determined by the SAT solver. However, inactive agents *do* receive rewards for the action that the SAT solver decided for them and thus they also *learn*.

With this distinction of active and inactive agents we can reduce the search space of our learning algorithm, since only the combination of actions of active agents constitute its search space. Actually, we start with a small number of active

agents and progressively activate more agents, so that the current agents have sufficient time to learn and stabilise its behaviour before new agents are introduced into the equation. Essentially, the activation criteria takes into account how many iterations passed without any improvements on the solution value, i.e. more agents are activated when there is no significant progress registered for a certain number of iterations (threshold). The activation of new agents introduces a disturbance in the results during a short time (see the peaks in Figure 4), but quickly leads to a new, better solution.

Moreover, we order the agents (PTOP constraints) by decreasing order of their weights, such that the agents which influence more the solution value are activated earlier in the process and have had, in the end, more iterations to learn. We realised in our experiments that this ordering led to better results.

Algorithm 1 summarises the learning procedure.

On lines 7 to 9, the agents select their actions simultaneously. Then, on lines 11 and 12, their decisions are passed to the SAT solver and a solution is searched.

In case we have a feasible combination of actions, we compute the cost of the solution (line 14) and use it to compute and assign rewards for each agent, which is done on lines 18 to 25. Notice, on lines 21 to 24, the distribution of rewards amongst all the actions between the one the agent chose and the one that was actually performed by the SAT solver, as we explained before.

Conversely, if the combination of actions is unsatisfiable, we extract an unsatisfiable (UNSAT) core and apply a penalty to all the agents involved in the core (lines 27 to 33).

Notice, on line 30 of Algorithm 1, that we apply the penalty to all actions equal or lower than the action chosen by the agent since we know that, in this particular domain, if a decision led to UNSAT any other lower decision would also imply UNSAT because it would constrain the problem even more.

Actually, we omitted from Algorithm 1 a small detail regarding the choice of actions. Whenever the algorithm gets stuck in a sequence of decisions recurrently producing the same objective value, we apply a previously visited set of decisions which is known to produce the best solution so far. That is, we save the agents decisions that led to the best solution found so far and replay it from time to time, reinforcing good experiences that were done in the past. This idea of experience replay was first introduced by Lin (1992, 1993), in the context of neural networks, and was later applied with great success by Zhang and Dietterich (2000) in the field of resource-constrained scheduling.

4 Results and discussion

All the tests in the following sections were run on a machine with an Intel Core i7-6700K CPU @ 4.00GHz and 64GB of RAM. We used the SAT solver Cryptominisat 5.0.1 [Soos (2016)] with one thread, the Maximum Satisfiability Problem (MaxSAT) solver Open-WBO 1.3.1 [Martins et al (2014)], the MILP solver IBM ILOG CPLEX 12.7, and a Constraint Programming (CP) solver developed from scratch at SISCOG and embedded in ONTIME.

Algorithm 1 Optimise a periodic timetable with SAT and machine learning.

Require: A PTOP problem P modelled with the order encoding

procedure MACHINE_LEARNING_HEURISTIC(P)
  $best \leftarrow +\infty$
  $t \leftarrow 1$
  while $t < MAX\_ITERATIONS$ **do**
    $activate\_agents(t)$ ▷ Progressively activate more agents
    $actions \leftarrow \emptyset$
    for each active agent in Agents **do**
      $a \leftarrow agent.select\_action(t)$ ▷ See Equation 18
      $actions \cup \{a\}$
    end for
    $assumptions \leftarrow apply\_actions(actions)$ ▷ Fix τ variables according to selected actions
    if SAT($P, assumptions$) **then** ▷ Invoke SAT solver
      $M \leftarrow$ get solution
      $C \leftarrow$ cost of solution M
      if $C < best$ **then**
        $best \leftarrow C$
      end if
      for each agent in Agents **do**
        $a \leftarrow$ action chosen by agent
        $a' \leftarrow$ action actually performed in M
        for each $action$ in $[a', a]$ **do**
          $reward \leftarrow compute\_reward(agent, action, C)$ ▷ See Equation 17
          $agent.update\_Q\_value(action, reward)$ ▷ See Equation 19
        end for
      end for
    else
      $U \leftarrow$ get UNSAT core
      for each active agent involved in U **do**
        $a \leftarrow$ action chosen by agent
        for each $action$ in $[1, a]$ **do**
          $agent.update\_Q\_value(action, 0)$ ▷ Penalty
        end for
      end for
    end if
  end while
  return $best$
end procedure

4.1 Benchmarking the machine learning approach

For benchmarking purposes, we compared the performance of our machine learning approach on a set of PTOP problems freely available as part of the PESPlib, created with the LinTim toolbox [Goerigk and Schöbel (2013)], with the results also published in that library's web page [Goerigk (2018)]. The PESPlib consists on a set of railway and bus timetabling problems formulated as PESPs. The best known solutions in terms of *travel time* are regularly published in the corresponding web page, as researchers all over the world find new solutions.

Table 1 is adapted from PESPlib's website[8] [Goerigk (2018)] and signals in bold our results whenever they outperformed the previously best ones. The results

[8] Last accessed on 4th May 2018.

Table 1 Benchmarking with PESPlib's problems — our results on all problems. In bold our own contributions to the web page. Lower values are better.

Name	Events	Constraints	Previously published value (on 26/09/2017)	Our value (on 3/10/2017)	Best published value (on 04/05/2018)
R1L1	3 664	6 386	31 838 103	40 061 093	31 099 786
R1L2	3 668	6 544	38 248 408	41 708 809	31 682 263
R1L3	4 184	7 032	38 612 098	**38 160 248**	30 535 261
R1L4	4 760	8 529	35 955 823	36 855 913	27 893 098
R2L1	4 156	7 362	53 708 802	**52 501 839**	42 502 069
R2L2	4 204	7 564	47 836 571	49 690 834	43 068 782
R2L3	5 048	8 287	46 530 294	50 116 385	39 942 656
R2L4	7 660	13 174	42 848 107	**42 429 329**	33 063 475
R3L1	4 516	9 146	53 299 647	55 161 582	45 483 668
R3L2	4 452	9 252	53 441 333	56 730 879	46 228 200
R3L3	5 724	11 170	48 707 212	49 491 444	43 039 089
R3L4	8 180	15 658	40 597 536	43 507 576	35 547 064
R4L1	4 932	10 263	59 225 243	61 240 481	51 650 471
R4L2	5 048	10 755	59 292 152	61 045 967	51 965 758
R4L3	6 368	13 239	54 975 374	55 514 383	45 881 499
R4L4	8 384	17 755	40 608 497	47 025 797	38 836 756
BL1	2 688	7 988	7 440 845	**7 387 963**	**7 387 963**
BL2	2 606	7 488	8 288 126	**8 143 507**	**8 143 507**
BL3	3 044	9 311	7 983 383	**7 826 762**	**7 826 762**
BL4	3 816	13 502	9 435 913	**7 359 779**	**7 359 779**

are ordered by objective value (lower is better). We were able to actually improve the previously known best values on some problems, as is shown in that table.

The results achieved by our machine learning based approach surpassed the ones obtained with other state-of-the-art approaches in several problems, namely the *R1L3*, *R2L1*, *R2L4*, *BL1*, *BL2*, *BL3*, and *BL4*. Moreover, for the latter four mentioned problems, our results remain (as of 4th May 2018) at the top of the table, which allows us to say that our approach is currently positioned amongst the state-of-the-art approaches for solving this kind of problems.

Since the results published on the PESPlib web page do not feature running times, we were not particularly worried about them. Therefore, we did not impose a hard time limit on the execution time for this experiment, and let the algorithm run until we were satisfied with the results. We ended up giving several days of execution time to most of the instances. Nevertheless, to give an impression on the performance of our approach we can say that, for most of the instances, we could achieve in about eight to twenty hours results within 2.5%-10% from the ones on Table 1. That is, we could achieve results that lie very close to the ones on Table 1 in less than a day, which is acceptable for long-term scheduling.

4.2 Comparison between the binary search and machine learning approaches

We would like to apply both our binary search [Matos et al (2017)] and ML approaches to the set of PESPlib problems to compare their performance, but unfortunately we cannot encode all the pseudo-boolean constraints of any single instance from that library for our binary search approach since it would not fit in the 64GB of memory that we had available. Therefore, the first conclusion is straightforward: many real-world, large-sized (actually, *interesting*) problems are simply out of the question for the binary search approach.

However, if we discard the binary search itself, but keep the best solution found by our new core guided heuristic used on that approach (see Matos (2018)), we can actually compare it with the best solution found by the ML approach. Figure 5 shows precisely the results of that comparison.

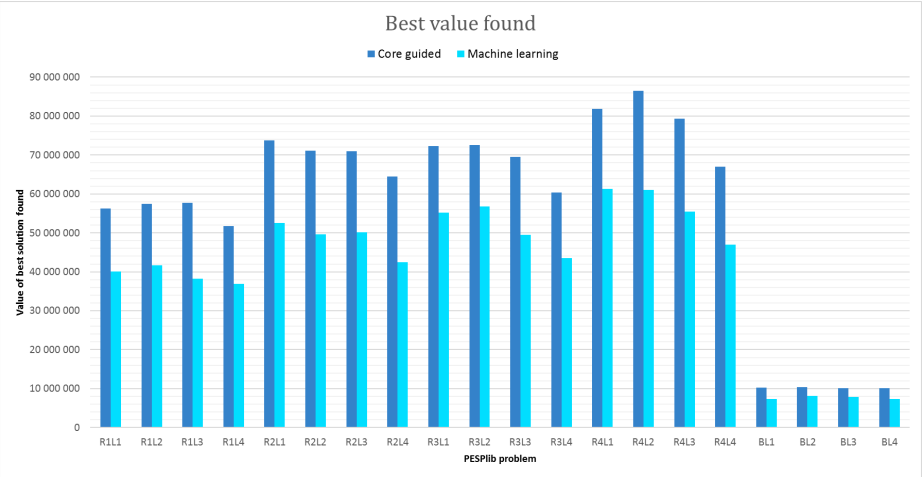

Fig. 5 Comparison between the core guided heuristic and the machine learning approach on PESPlib problems. Lower values are better.

As we can see, the ML approach could find better solutions than the ones found by our core guided heuristic on all PESPlib's problems with no exception, with improvements ranging 21% to 33%. In fact, our best solutions for the PESPlib problems were found by the ML approach (see Section 4.1). Therefore, we realised that a machine-learned heuristic, developed by trial and error from the data underlying the problem being solved, outperformed our human-designed core guided heuristic, which makes use of general knowledge from the problem domain, at least on large problems.

In order to compare the complete core guided + binary search approach with the ML approach, we had to artificially develop some problems whose encoding, when approached by the binary search, could fit into 64GB of memory and still present some interesting challenge. We started with the real-world problem *M1_24_original* from Table 2 and, in order to harden the problem, progressively added to it artificial connection constraints simulating the interest of some passengers in transferring from one train to another at some intersection stations. Then we did the same on another real-world problem (*M6_120*), and ended up producing the instances presented on Table 2.

We have set a time limit of two hours and collected, for both approaches, the number of iterations that the algorithm could perform[9], and the value of the best solution found with the travel time objective function. In the case of the binary search approach, we separated the solution values and number of iterations for

[9] See the algorithms' pseudo-code in Matos (2018).

Table 2 Comparison between the binary search + core guided heuristic approach and the machine learning approach on artificially hardened real-world problems. Lower values are better.

Problem	Events	Constraints	Core guided heur + binary search					Machine learning		
			Value heur	Value binary search	Heur it.	Binary it.	Time	Value	It.	Time
M1_24_original	916	2 175	0	0	60	0*	00:00:06	0	100	00:00:14
M1_24_stressed_1	916	2 176	20	20	60	6	00:00:06	20	458	00:00:43
M1_24_stressed_2	916	2 177	63	60	60	6	00:00:07	60	648	00:00:58
M1_24_stressed_3	916	2 183	249	240	60	9	00:00:29	240	298	00:00:29
M1_24_stressed_4	916	2 202	670	641	60	11	02:00:00+	804	140 384	02:00:00 +
M1_24_stressed_5	996	2 428	792	768	60	9	02:00:00+	861	130 981	02:00:00 +
M1_24_stressed_6	996	2 240	1 104	1 104	60	9	02:00:00+	1 245	126 655	02:00:00 +
M6_120_stressed	4 980	50 391	171	7	60	7	01:32:14	7	1 046	00:26:51

the core guided heuristic (which runs first) from the ones of the binary search procedure itself (which gives the final solution).

In this set of problems the picture is very different from what we saw for the PESPlib. The ML approach does not outperform the human-designed approach on all instances anymore. In fact, the solutions found by the binary search approach are better and produced in less time in most of the cases. There is no much surprise in these results, since the binary search approach is an exact method, so it is expected to produce the optimal solution, if it is given sufficient time, or at least optimised solutions which are better than an approximate method such as our ML approach. However, what is interesting to note is that the core guided heuristic itself finds solutions that are pretty close to the final best solution found by the binary search procedure. In fact, on the instance *M1_24_original*, the core guided heuristic actually found the optimal solution and thus the binary search procedure was completely bypassed[10]. Therefore, the results indicate that our core guided heuristic combined with the binary search is still better than our ML approach on smaller problems, whereas the latter approach unleashes its full potential on bigger problems (such as the PESPlib) where the former soon gets stuck.

In light of the above, we concluded that our binary search approach [Matos et al (2017); Matos (2018)] is well-suited for small problems and its greatest advantage is the capability of finding the optimal solution and proving its optimality[11]. However, it suffers from the curse of dimensionality, and so consumes unacceptable amounts of memory and becomes impractical on real-world, large-sized problems. If we discard the binary search process and use just the core guided heuristic to produce solutions (thus avoiding the memory problems, but also loosing optimality guarantees), we can address bigger problems, but we cannot go as far as we can with the ML approach under the same conditions, in general.

The ML approach, on the other hand, is capable of approaching even the largest instances and find good, optimised solutions, even outperforming other state-of-the-art solvers (Section 4.1).

5 Conclusion

We solved the Periodic Timetable Optimisation Problem (PTOP) with an approach based on reinforcement learning, multi-agents and SAT. To our knowledge

[10] When a solution whose value equals the absolute minimum lower bound for the problem is found by the heuristic we do not even call the binary search procedure.

[11] Longer though it can take!

it is the first time that machine learning techniques are used to solve this important transportation scheduling problem.

In order to evaluate our approach we benchmarked it with a set of PTOP problem instances freely available as part of the PESPlib, with the results also published in that library's web page [Goerigk (2018)]. The PESPlib consists on a set of real-world railway and bus periodic timetabling problems.

The results achieved by our approach surpassed the ones obtained with other state-of-the-art approaches in several problems, namely the *R1L3*, *R2L1*, *R2L4*, *BL1*, *BL2*, *BL3*, and *BL4* (see Table 1). Moreover, for the latter four mentioned problems, our results remain (as of 4th May 2018) at the top of the table, which allows us to say that our approach is currently positioned amongst the state-of-the-art approaches for solving this kind of problems.

This work follows a previous work (see Matos et al (2017)) where we developed an exact method based on SAT and binary search with an heuristic for improving the upper bound. As shown above, in small problem instances the exact method returns the optimal solution, outperforming the machine learning approach, but for medium and large sized instances (namely the PESPLib ones) the exact method becomes incapable of obtaining solutions due to memory problems, while the machine learning approach obtains valuable solutions, some of them above the state-of-the-art.

References

Auer P, Cesa-bianchi N, Fischer P (2002) Finite time analysis of the multi-armed bandit problem. Machine Learning 47(2-3):235–256, DOI 10.1023/A:1013689704352

Cook SA (1971) The complexity of theorem-proving procedures. In: Proceedings of the third annual ACM symposium on Theory of computing, ACM, pp 151–158

Fu Z, Malik S (2006) On solving the partial MAX-SAT problem. SAT 6:252–265

Gattermann P, Nachtigall K (2016) Integrating Passengers' Routes in Periodic Timetabling: A SAT approach. In: Goerigk M, Werneck R (eds) 16th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS 2016), Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, pp 3:1–3:15

Goerigk M (2018) Pesplib web page. URL `http://num.math.uni-goettingen.de/~m.goerigk/pesplib/`

Goerigk M, Liebchen C (2017) An Improved Algorithm for the Periodic Timetabling Problem. In: D'Angelo G, Dollevoet T (eds) 17th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS 2017), Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, pp 12:1 – 12:14

Goerigk M, Schöbel A (2013) Improving the modulo simplex algorithm for large-scale periodic timetabling. Computers and Operations Research 40(5):1363–1370, DOI 10.1016/j.cor.2012.08.018

Großmann P (2011) Polynomial Reduction from PESP to SAT. Tech. rep., Knowledge Representation and Reasoning Group, TU Dresden

Großmann P, Hölldobler S, Manthey N, Nachtigall K, Opitz J, Steinke P (2012a) Solving Public Railway Transport Networks with SAT. In: Jiang H, Ding W,

Ali M, Wu X (eds) 25th International Conference on Industrial Engineering and Other Applications of Applied Intelligent Systems, Springer, Lecture Notes in Computer Science, vol 7345, pp 166–175

Großmann P, Weiss R, Opitz J, Nachtigall K (2012b) Automated Generation and Optimization of Public Railway and Rail Freight Transport Time Tables. MTM (5):23–26

Großmann P, Opitz J, Wei R (2015) On Resolving Infeasible Periodic Event Networks. In: Proceedings of 13th Conference on Advanced Systems in Public Transport (CASPT)

Ingolotti L, Lova A, Barber F, Tormos P, Salido MA, Abril M (2006) New heuristics to solve the "CSOP" railway timetabling problem. In: Ali M, Dapoigny R (eds) Advances in Applied Artificial Intelligence, Springer, pp 400–409, DOI 10.1007/11779568_44

Kümmling M, Großmann P, Nachtigall K, Opitz J, Weiß R (2015) A state-of-the-art realization of cyclic railway timetable computation. Public Transport 7(3):281–293, DOI 10.1007/s12469-015-0108-5

Lin LJ (1992) Self-improving reactive agents based on reinforcement learning, planning and teaching. Machine learning 8:69–97

Lin LJ (1993) Reinforcement learning for robots using neural networks. Phd thesis, Carnegie-Mellon Univ, Pittsburgh, PA

Manthey N, Steinke P (2012) npSolver - A SAT based solver for optimization problems. Pragmatics of SAT

Martins R, Manquinho V, Lynce I (2014) Open-WBO: A modular MaxSAT solver. In: Sinz C, Egly U (eds) International Conference on Theory and Applications of Satisfiability Testing – SAT 2014, Springer, pp 438–445, URL `http://sat.inesc-id.pt/open-wbo/`

Matos GP (2018) Optimisation of Periodic Train Timetables. Msc thesis, Instituto Superior Técnico

Matos GP, Albino L, Saldanha RL, Morgado EM (2017) Optimising Cyclic Timetables with a SAT Approach. In: Portuguese Conference on Artificial Intelligence, Springer, pp 343—-354

Matos GP, Albino L, Saldanha RL, Morgado EM (2018) Solving periodic timetabling problems with SAT and machine learning. In: Proceedings of 14th Conference on Advanced Systems in Public Transport (CASPT)

Nachtigall K, Voget S (1997) Minimizing waiting times in integrated fixed interval timetables by upgrading railway tracks. European journal of operational research 103(3):610–627

Peeters LWP (2003) Cyclic railway timetable optimization. Phd thesis, Erasmus Universiteit Rotterdam

Serafini P, Ukovich W (1989) A mathematical model for periodic event scheduling problems. SIAM Journal on Discrete Mathematics 2(4):550–581, DOI 10.1137/0402049

Soos M (2016) The CryptoMiniSat 5 set of solvers at SAT Competition 2016. In: Balyo T, Heule MJH, Järvisalo M (eds) SAT Competition 2016, p 28, URL `https://github.com/msoos/cryptominisat`

Staereling I (2015) High-Potential Heuristics for Railway Timetabling. In: Proceedings of 13th Conference on Advanced Systems in Public Transport (CASPT)

de Waal E (2005) Generating periodic timetables using a cutting plane algorithm. Msc thesis, Vrije Universiteit Amsterdam

Watkins CJ, Dayan P (1992) Q-learning. Machine learning 8:279–292
Zhang W, Dietterich TG (2000) Solving Combinatorial Optimization Tasks by Reinforcement Learning: A general Methodology Applied to Resource-constrained Scheduling. Journal of Artificial Intelligence Research 53(9):1689–1699, DOI 10.1017/CBO9781107415324.004, arXiv:1011.1669v3