A Filtering Technique for the Railway Scheduling Problem

Marlene Arangú and **Miguel A. Salido** and **Federico Barber**
Instituto de Automática e Informática Industrial
Universidad Politécnica de Valencia.
Valencia, Spain

Abstract

Railway scheduling has been a significant issue in the railway industry. Building the schedule of trains is a difficult and time-consuming task, particularly in the case of real networks, where the number and the complexity of constraints grow dramatically. Railway scheduling can be modeled as a Constraint Satisfaction Problem (CSP) and it can also be solved using constraint programming techniques. Arc-Consistency algorithms are the most commonly used filtering techniques to prune the search space of CSPs. 2-consistency guarantees that any instantiation of a value to a variable can be consistently extended to any second variable, and as a consequence 2-consistency can be stronger than arc-consistency in binary CSPs. In this work we present a new algorithm, called 2-C3OPL, a reformulation 2-C3 algorithm that is able to reduce unnecessary checking and prune more search space than AC3. Although these algorithms are for general purpose, thay have been mainly developed to detect inconsistencies in the railway scheduling problem. In the experimental result section, we evaluate the behaviour of our techniques on random instances and a empirical evaluation was performed using real data provided by the Spanish Manager of Railway Infrastructure (ADIF).

Introduction

Railway transportation has a major role in many countries over the last few years. Railway traffic has increased considerably [1], which has created the need for optimizing both the use of railway infrastructures and the methods and tools to perform it. A feasible schedule should specify the departure and arrival time of each train to each location of its journey, taking into account the line capacity and other operational constraints. One of the goals set for 2020 in Europe is to achieve a 20% increase in passenger transport and 70% in goods (Salido et al. 2005). This implies a need to improve management of infrastructure capacity. Building the schedule of trains is a difficult and time-consuming task, particularly in the case of real networks, where the number of constraints and the complexity of constraints grow dramatically. Thus, numerous approaches and tools have been developed to compute railway scheduling (see survey in (Barber et al. 2007)).

In this work, the railway scheduling problem is modeled as a Constraint Satisfaction Problems (CSPs), as (Ingolotti 2007; Walker, Snowdon, and Ryan 2005; Silva de Oliveira 2001). For simplicity an empty network is assumed so that no previous trains are scheduled in the network. Since the number of variables and constraints generated in the problem is high, our goal is developing filtering techniques which reduce the search space, so that a solution can efficiently be found.

There exist many levels of consistency depending on the number of variables involved: node-consistency involves only one variable; arc-consistency involves two variables; path-consistency involves three variables; k-consistency involves k variables. More information can be seen in (Barták 2001; Dechter 2003).

Arc-consistency is the basic propagation mechanism that is probably used in all solvers (Bessiere 2006). Proposing efficient algorithms for enforcing arc-consistency has always been considered as a central question in the constraint reasoning community, so there exist many arc-consistency algorithms such as AC1 .. AC8, AC2001/3.1.. AC3rm; and more.

Algorithms that perform arc-consistency have focused their improvements on time-complexity and space-complexity. Main improvements have been achieved by: changing the way of propagation: from arcs (AC3 (Mackworth 1977)) to values (AC6 (Bessiere and Cordier 1993; Bessiere 1994)), (i.e., changing the granularity: coarse-grained to fine-grained); appending new structures; performing bidirectional searches (AC7 (Bessiere, Freuder, and Régin 1999)); changing the support search: searching for all supports (AC4 (Mohr and Henderson 1986)) or searching for only the necessary supports (AC6, AC7, AC2001/3.1 (Bessiere et al. 2005)); improving the propagation (i.e., it performs propagation only when necessary, AC7 and AC2001/3.1); etc. However, AC3(Mackworth 1977) and AC2001/3.1(Bessiere et al. 2005) are the most often used (Lecoutre and Hemery 2007).

The concept of consistency was generalized to k-consistency by (Freuder 1978) and an optimal k-consistency algorithm for labeling problem was proposed by (Cooper 1994). Nevertheless, it is only based in normalized CSPs

[1] See statistics in http://epp.eurostat.ec.europa.eu and http://www.ine.es

(two different constraints do not involve exactly the same variables). If the constraints have two variables in k-consistency (k=2) then we talking about to 2-consistency. 2-consistency guarantees that any instantiation of a value to a variable can be consistently extended to any second variable.

Figure 1 left shows a binary CSP with two variables X_1 and X_2, $D_1 = D_2 = \{1, 2, 3\}$ and a block of two constraints $C_{12} : \{R_{12}, R'_{12}\} : R_{12}(X_1 \leq X_2)$, $R'_{12}(X_1 \neq X_2)$ presented in (Rossi, Van Beek, and Walsh 2006). It can be observed that this CSP is arc-consistent due to the fact that every value of every variable has a support for constraints R_{12} and R'_{12}. In this case, arc-consistency does not prune any value of the domain of variables X_1 and X_2. However, (as authors say in (Rossi, Van Beek, and Walsh 2006)) this CSP is not 2-consistent because the instantiation $X_1 = 3$ can not be extended to X_2 and the instantiation $X_2 = 1$ can not be extended to X_1. Thus, Figure 1 right presents the resultant CSP filtered by arc-consistency and 2-consistency. It can be observed that 2-consistency is at least as strong as arc-consistency.

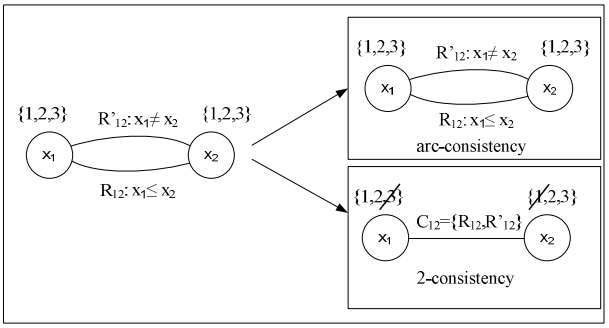

Figure 1: Example of Binary CSP.

Our CSP formulation on the railway scheduling problem contains non-normalized constraints (different constraints may involve exactly the same variables). The CSP formulation described in this work has been defined together with the Spanish Manager of Railway Infrastructure (ADIF) in such a way that the resulting scheduling was feasible and practicable. The railway scheduling problem is characterized by having many variables with large domains. Besides, a non-normalized CSP can be transformed into normalized using the intersection of valid tuples (Arangú, Salido, and Barber 2009b); however it can be very costly in large domains (Arangú, Salido, and Barber 2009a).

In this paper, we propose a new filtering technique: the 2-C3OPL algorithm, that reaches 2-consistency for binary and non-normalized CSPs. Our proposed technique is domain-independent and it performs 2-consistency efficiently (it saves checks and running time) In this work the consistency technics are used before search in pre-process step. This paper is organized as follows: in the following sections we explain in detail the railway scheduling problem and we provide both: the necessary definitions and notations. Then, we explain the algorithms AC3 and 2C3 and, we present our 2-C3OPL algorithm. In experimental results section, we evaluate AC3, 2-C3 and 2-C3OPL empirically using both random problems and real data of ADIF in railway benchmark problem, and finally, we present our conclusions and further works.

Definitions and Notations of CSP

By following standard notations and definitions in the literature (Bessiere 2006); (Barták 2001), (Dechter 2003); We have summarized the basic definitions that we will use in this rest of the paper:

Constraint Satisfaction Problem (CSP) is a triple $P = \langle X, D, R\rangle$ where, X is the finite set of variables $\{X_1, X_2, ..., X_n\}$. D is a set of domains $D = D_1, D_2, ..., D_n$ such that for each variable $X_i \in X$ there is a finite set of values that variable can take. R is a finite set of constraints $R = \{R_1, R_2, ..., R_m\}$ which restrict the values that the variables can simultaneously take. R_{ij} is used to indicate the existence of a constraint between the variables X_i and X_j. A block of constraints C_{ij} is a set of binary constraints that involve the same variables X_i and X_j. We denote $C_{ij} \equiv (C_{ij}, 1) \vee (C_{ij}, 3)$ as the block of direct constraints defined over the variables X_i and X_j and $C_{ji} \equiv (C_{ij}, 2)$ as the same block of constraints in the inverse direction over the variables X_i and X_j (inverse block of constraints).

Instantiation is a pair $(X_i = a)$, that represents an assignment of the value a to the variable X_i, and a is in D_i. A constraint R_{ij} is satisfied if the instantiation of $X_i = a$ and $X_j = b$ hold in R_{ij}. **Symmetry of the constraint.** If the value $b \in D_j$ supports a value $a \in D_i$, then a supports b as well.

Arc-consistency: A **value** $a \in D_i$ is arc-consistent relative to X_j, iff there exists a value $b \in D_j$ such that (X_i, a) and (X_j, b) satisfies the constraint R_{ij} ($(X_i = a, X_j = b) \in R_{ij}$). A **variable** X_i is arc-consistent relative to X_j iff all values in D_i are arc-consistent. A **CSP** is arc-consistent iff all the variables are arc-consistent, e.g., all the constraints R_{ij} and R_{ji} are arc-consistent. (Note: here we are talking about full arc-consistency).

2-consistency: A **value** $a \in D_i$ is 2-consistent relative to X_j, iff there exists a value $b \in D_j$ such that (X_i, a) and (X_j, b) satisfies all the constraints R^k_{ij} ($\forall k : (X_i = a, X_j = b) \in R^k_{ij}$). A **variable** X_i is 2-consistent relative to X_j iff all values in D_i are 2-consistent. A **CSP** is 2-consistent iff all the variables are 2-consistent, e.g., any instantiation of a value to a variable can be consistently extended to any second variable.

Railway Scheduling Problem

The main objective of the railway scheduling problem is to minimize the journey time of a set of trains. A *railway network* is basically composed of locations of one-way or two-way tracks. A *location* can be:

- *Station*: is a place for trains to park, stop or pass through. There are two or more tracks in a station where crossings or overtaking can be performed. Each station is associated with a unique station identifier.

- *Halt*: is a place for trains to stop, pass through, but not park. Each halt is associated with a unique halt identifier.
- *Junction*: is a place where two different tracks fork. There is no stop time. Each junction is associated with a unique junction identifier.

On a rail network, the user needs to schedule the paths of n trains going in one direction (up) and m trains going in the opposite direction (down). These trains are of a given type and a scheduling frequency is required. The type of trains to be scheduled determines the time assigned for travel between two locations on the path. The path selected by the user for a train trip determines which stations are used and the stop time required at each station for commercial purposes. In order to perform crossing in a section with a one-way track, one of the trains should wait in a station. This is called a technical stop. One of the trains is detoured from the main track so that the other train can cross or continue.

A *running map* contains information regarding the topology of the railways (stations, tracks, distances between stations, traffic control features, etc.) and the schedules of the trains that use this topology (arrival and departure times of trains at each station, frequency, stops, crossing, overtaking, etc.). We assume that two connected locations have only one line connecting them.

The Figure 2 shows a running map where the names of the stations are presented on the left side and the vertical line represents the number of tracks between stations (one-way or two-way). Horizontal dotted lines represent halts or junctions, while solid lines represent stations. The objective is to obtain a correct and optimized running map taking into account: (i) traffic rules, (ii) user requirements and (iii) the railway infrastructure topology.

Notation of railway scheduling problem

The notation used to describe the railway scheduling problem is based on the works of (Ingolotti 2007; Tormos et al. 2008). The notation is the following:

- T: finite set of trains t considered in the problem. $T = \{t_1, t_2, ..., t_k\}$. T_D: set of trains traveling in the *down* direction. T_U: set of trains traveling in the *up* direction. Thus, $T = T_D \cup T_U$ and $T_D \cap T_U = \emptyset$.
- $L = \{l_0, l_1, ..., l_m\}$: railway line that is composed by an ordered sequence of locations (stations, halts, junctions) that may be visited by trains $t \in T$. The contiguous locations l_i and l_{i+1} are linked by a single or double track section.
- J_t: journey of train t. It is described by an ordered sequence of locations to be visited by a train t such that $\forall t \in T, \exists J_t : J_t \subseteq L$. The journey J_t shows the order that is used by train t to visit a given set of locations. Thus, l_0^t and $l_{n_t}^t$ represent the *fist* and *last* location visited by train t, respectively.
- C_i^t minimum time required for train t to perform commercial operations (such as boarding or leaving passengers) at station i (commercial stop).
- $\Delta_{i\rightarrow(i+1)}^t$: journey time for train t from location l_i^t to $l_{(i+1)}^t$.
- F: Frequency of trains $t \in T$ considered in the problem. F_D: Frequency of trains traveling in the *down* direction. F_U: Frequency of of trains traveling in the *up* direction.
- λ: delay time allowed for train t with frequency F.

CSP formulation of railway scheduling problem

A CSP formulation consists of the tuple $\langle$ X, D, R $\rangle$, where X is a set of variables, D is a set of domains and R is a set of binary constraints.

Variables

Each train travelling in each location will generate two different variables (arrival and departure time):

- dep_i^t departure time of train $t \in T$ from the location i, where $i \in J_t \setminus \{l_{n_t}^t\}$.
- arr_i^t arrival time of train $t \in T$ to the location i, where $i \in J_t \setminus \{l_0^t\}$.

Domains

The domain of each variable (dep_i^t or arr_i^t) is an interval $[min_V, max_V]$, where $min_V \geq 0$ and $max_V > min_V$. This interval is obtained from data of ADIF, according to the journey.

Constraints

There are three groups of scheduling rules in our railway system: traffic rules, user requirements rules and topological rules. A valid running map must satisfy and optimize the above rules. These scheduling rules can be modeled using the following constraints:

1. Traffic rules guarantee crossing and overtaking operations. The main constraints to take into account are:
   - Crossing constraint: Any two trains going in opposite directions must not simultaneously use the same one-way track: $dep_{i+1}^{t'} > arr_{i+1}^t \vee dep_i^t > arr_i^{t'}$.
   - Expedition time constraint. There exists a given time to put a detoured train back on the main track and exit from a station: $|dep_l^{t'} - arr_l^t| \geq E_t$, where E_t is the expedition time specified for t.
   - Reception time constraint. There exists a given time to detour a train from the main track so that crossing or overtaking can be performed: $arr_l^{t'} \geq arr_l^t \rightarrow arr_l^{t'} - arr_l^t \geq Recep_t$, where $Recep_t$ is the reception time specified for the train that arrives to l first.
   - Journey Time constraint. $arr_{i+1}^t = dep_i^t + \Delta_{i\rightarrow(i+1)}^t$. For each train t and each track section, a Journey time is given by $\Delta_{i\rightarrow(i+1)}^t$, which represents the time the train t should employ to go from location l_i^t to location l_{i+1}^t.
2. User Requirements: The main constraints due to user requirements are:
   - Number of trains going in each direction n (down) and m (up) to be scheduled. $T = T_D \cup T_U$, where:

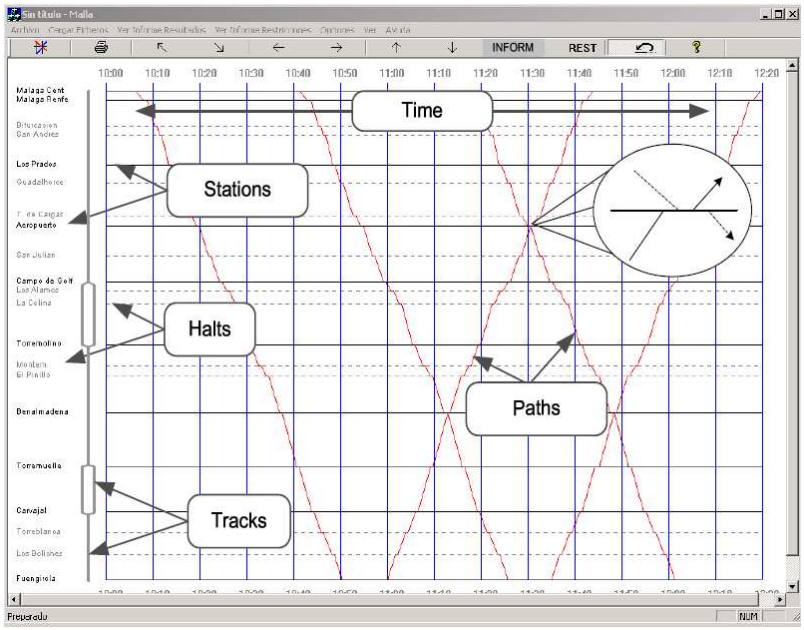

Figure 2: Example of running map for railway scheduling problem.

- $t \in T_D \leftrightarrow (\forall l_i^t : 0 \leq i < n_t, \exists l_j \in \{L \setminus \{l_m\}\} : l_i^t = l_j \wedge l_{i+1}^t = l_{j+1})$, and
- $t \in T_U \leftrightarrow (\forall l_i^t : 0 \leq i < n_t, \exists l_j \in \{L \setminus \{l_0\}\} : l_i^t = l_j \wedge l_{i+1}^t = l_{j-1})$.

- Journal: Locations used and Stop time for commercial purposes in each direction for each train $t \in T$: $J_t = \{l_0^t, l_1^t, ..., l_{n_t}^t\}$
- Scheduling frequency. The frequency requirements F of the departure of trains in both directions: $dep_{i+1}^t - dep_i^t = F + \lambda_{i+1}$. This constraint is very restrictive because, when crossing are performed, trains must wait for a certain time interval at stations. This interval must be propagated to all trains going in the same direction in order to maintain the established scheduling frequency. The user can require a fixed frequency, a frequency within a minimum and maximum interval, or multiple frequencies.

  A frequency within a minimum and maximum interval was chosen in this work:
  - For $t \in T_D : dep_i^t + F_D < dep_{i+1}^t$ and $dep_i^t + F_D + \lambda_{i+1} > dep_{i+1}^t$.
  - For $t \in T_U : dep_i^t + F_U < dep_{i+1}^t$ and $dep_i^t + F_U + \lambda_{i+1} > dep_{i+1}^t$.

3. Topological railway infrastructure and type of trains to be scheduled give rise other constraints to be taken into account. Some of them are:

   - Number of tracks in stations (to perform technical and/or commercial operations) and the number of tracks between two locations (one-way or two-way). No crossing or overtaking is allowed on a one-way track,
   - Added Station time constraints for technical and/or commercial purposes. Each train $t \in T$ is required to remain in a station l_i^t at least Com_i^t time units (Commercial stop). $dep_i^t \geq arr_i^t + Com_i^t$

In accordance with ADIF requirements, the system should obtain a solution so that all the above constraints (traffic, user requirements and topological) are satisfied. The source of difficulties underlying the Railway Scheduling are: a) the problem of the railway scheduling each location generates two variables whose domain sizes are large (a combinatorial problem). b) The increase of trains increases disjunctive constraints (that generates the search tree branches). c) The increased frequency of trains, makes the problem more restrictive (often no solution). d) Finding a solution using a search algorithm as FC for instances proposed in this paper can take more than two hours.

AC3 algorithm

The AC3 algorithm (Mackworth 1977) is one of most popular algorithms for arc-consistency. AC3 is a coarse grained algorithm (because it propagates arcs) and it is very easy of implements. The main algorithm (see Algorithm 1) is a simple loop that selects and revises the constraints stored in Q until either no change occurs (Q is empty) or the domain of a variable becomes empty. The first case ensures that all values of domains are consistent with all constraints, and second case returns that the problem has no solution.

To avoid many useless calls to the *ReviseAC3* procedure, AC3 keeps all the constraints R_{ij} that do not guarantee that

Algorithm 1: Procedure AC3

Data: A CSP, $P = \langle X, D, R \rangle$
Result: **true** and P' (which is arc consistent) or **false** and P' (which is arc inconsistent because some domain remains empty)

```
begin
1   for every arc R_ij ∈ R do
2       Append (Q, (R_ij)) and Append (Q, (R'_ji))
3   while Q ≠ φ do
4       select and delete R_ij from queue Q
5       if ReviseAC3(R_ij) = true then
6           if D_i ≠ φ then
7               Append (Q, (R_ki)) with k ≠ i, k ≠ j
            else
8               return false /*empty domain*/
9   return true
end
```

Algorithm 2: Procedure ReviseAC3

Data: A $CSP\ P'$ defined by two variables $X = (X_i, X_j)$, domains D_i and D_j, and constraint R_{ij}.
Result: D_i, such that X_i is arc consistent relative X_j and the boolean variable $change$

```
begin
1   change ← false
2   for each a ∈ D_i do
3       if ∄b ∈ D_j such that (X_i = a, X_j = b) ∈ R_ij then
4           remove a from D_i
5           change ← true
6   return change
end
```

D_i is arc-consistent with the constraints in a queue Q. Also, Q is updated by adding constraints R_{ki}, which were, previously assessed where D_k can be inconsistent because D_i was pruned. AC3 achieves arc-consistency on binary networks in $O(md^3)$ time and $O(m)$ space, where d is the domain size and m is the number of binary constraints in the problem.

The time complexity of AC3 is not optimal because the *ReviseAC3* procedure (see Algorithm 2) does not remember anything about its computations to find supports for values, which leads AC3 to perform the same constraint checks many times.

2-C3 algorithm

The 2-C3 algorithm (Arangú, Salido, and Barber 2009a) is a coarse-grained algorithm that achieves 2-consistency in binary and non-normalized CSPs. This algorithm is a reformulation of the well-known AC3 algorithm. The main algorithm is a simple loop that selects and revises the block of constraints stored in a queue Q until no change occurs (Q is empty), or until the domain of a variable becomes empty. The first case ensures that all values of domains are consistent with all block of constraints, and the second case returns that the problem has no solution.

Algorithm 3: Procedure 2-C3

Data: A CSP, $P = \langle X, D, R \rangle$
Result: **true** and P' (which is 2-consistent) or **false** and P' (which is 2-inconsistent because some domain remains empty)

```
begin
1   for every i, j do
2       C_ij = ∅
3   for every arc R_ij ∈ R do
4       C_ij ← C_ij ∪ R_ij
5   for every set C_ij do
6       Q ← Q ∪ {C_ij, C'_ji}
7   while Q ≠ φ do
8       select and delete C_ij from queue Q
9       if Revise2C3(C_ij) = true then
10          if D_i ≠ φ then
11              Q ← Q ∪ {C_ki | k ≠ i, k ≠ j}
12          else
13              return false /*empty domain*/
14  return true
end
```

The Revise procedure of 2-C3 is very close to the Revise procedure of AC3. The only difference is that the instantiation $(X_i = a, X_j = b)$ must be checked with the block of constraints C_{ij} instead of with only one constraint. This set of constraints C_{ij} could also be ordered in order to avoid unnecessary checks. If we order this set from the tightest constraint to the loosest constraint, the constraint checking will find inconsistency constraints sooner, in which case no further constraint checks must be carried out.

Algorithm 4: Procedure Revise2C3

Data: A $CSPP'$ defined by two variables $X = (X_i, X_j)$, domains D_i and D_j, and constraint set C_{ij}.
Result: D_i, such that X_i is 2-consistent relative X_j and the boolean variable $change$

```
begin
1   change ← false
2   for each a ∈ D_i do
3       if ∄b ∈ D_j such that (X_i = a, X_j = b) ∈ C_ij then
4           remove a from D_i
5           change ← true
6   return change
end
```

The 2-C3 algorithm performs a stronger than AC3 prune, however, as with AC3, the time complexity of 2-C3 is not optimal. This is because the *Revise2C3* procedure does not remember anything about its computations to find supports for values, which leads 2-C3 to perform the same constraint block checks many times.

2-C3OPL algorithm

2-C3OPL is a new coarse grained algorithm that achieves 2-consistency in binary and non-normalized CSPs (see Al-

gorithm 5). This algorithm deals with block of constraints as 2-C3 but it only requires to keep half of the block of constraints in Q. Furthermore, 2-C3OPL avoids ineffective checks by storing the last support founded (as AC2001/3.1 (Bessiere et al. 2005)). Thus, performance gain 2-C3OPL in general is due: 1) the 2-C3OPL algorithm performs bidirectional checks; 2) it stores bidirectionally the support values for each block of constraints and 3) it performs inference to avoid unnecessary checks. However, inference is done by using: a) the structures ($suppInv$, $minSupp$ and t) that are shared by all the constraints, and b) a matrix ($Last$) where the values are shared by all constraints in the block.

Algorithm 5: Procedure 2-C3OPL

Data: A CSP, $P = \langle X, D, R\rangle$
Result: **true** and P' (which is 2-consistent) or **false** and P' (which is 2-inconsistent because some domain remains empty)
begin
1 for *every* i, j **do**
2 $C_{ij} = \emptyset$
3 for *every arc* $R_{ij} \in R$ **do**
4 $C_{ij} \leftarrow C_{ij} \cup R_{ij}$
5 for *every set* C_{ij} **do**
6 $Q \leftarrow Q \cup \{(C_{ij}, t) : t = 1\}$
$Last[C_{ij}, Xi, a] = dummyvalue$;
$Last[C_{ij}, Xj, b] = dummyvalue; \forall a \in D_i; \forall b \in D_j$
7 for *each* $d \in D_{max}$ **do**
8 $suppInv[d] = 0$
9 **while** $Q \neq \emptyset$ **do**
10 select and delete (C_{ij}, t) from queue Q with $t = \{1, 2, 3\}$
11 $change = ReviseOPL((C_{ij}, t))$
12 **if** $change > 0$ **then**
13 **If** $change \geq 1 \wedge change \leq 3$ **then**
$Q \leftarrow Q \cup AddQ(change, (C_{ij}, t))$
else return false /*empty domain*/
14 **return** true
end

$suppInv$ is a vector whose size is the maximum size of all domains ($maxD$). It stores the value 1 when the value of X_j is supported. $minSupp$ is an integer variable that stores the first value $b \in D_j$ that supports any $a \in D_i$. t is an integer parameter which takes values $t = \{1, 2, 3\}$. This value is used during the Revise procedure in order to determine whether to check or not a constraint C_{ij} (direct or inverse order) and to decide between imposing or not bidirectionality.

Initially 2-C3OPL procedure stores in queue Q the constraint blocks $(C_{ij}, t) : t = 1$ and initializes both the vector $suppInv$ to zero and the matrix $Last$ to a dummy value. Then, a simple loop is performed to select and revise the block of constraints stored in Q, until no change occurs (Q is empty), or the domain of a variable remains empty. The first case ensures that every value of every domain is 2-consistent and the second case returns that the problem is not consistent.

The *ReviseOPL* procedure (see Algorithm 6) requires two internal variables $change_i$ and $change_j$. They are initialized to zero and are used to remember which domains were pruned. For instance, if D_i was pruned then $change_i = 1$ and if D_j was pruned then $change_j = 2$. However, if both D_i and D_j were pruned then $change = 3$ (because $change = change_i + change_j$). Also, $minorSupp$ variable is initialized in a dummy value. During the loop of steps 3-11, each value in D_i is checked[2]. Firstly, the $Last$ matrix is checked. If the stored value in $Last[C_{ij}, X_i, a]$ belongs to D_j, no constraint checking is needed because this value $a \in D_i$ has a valid support in D_j (it was founded in previous iteration) and furthermore, $minSupp$ is updated. If the stored value in $Last[C_{ij}, X_i, a]$ does not belong to D_j, a support $b \in D_j$ is searched. If the value $b \in D_j$ supports the value $a \in D_i$ then $suppInv[b] = a$, due to the symmetry of the constraint (the support is bidirectional). Furthermore the first value $b \in D_j$ (which supports a value in D_i) is stored in $minSupp$ and the $Last$ matrix is bidirectionally updated.

The second part of Algorithm 6 is carried out in function of values t and $change_i$. If $t = 2$ or $t = 3$, and $change_i = 0$

Algorithm 6: Procedure ReviseOPL

Data: A CSP P' defined by two variables $X = (X_i, X_j)$, domains D_i and D_j, tuple (C_{ij}, t), vector $Last$ and vector $suppInv$.
Result: D_i, such that X_i is 2-consistent relative X_j and D_j, such that X_j is 2-consistent relative X_i and integer variable $change$
begin
1 $change_i = 0; change_j = 0$
2 $minSupp$ =*dummyvalue*
3 **for** *each* $a \in D_i$ **do**
4 **if** *value stored in* $Last[C_{ij}, X_i, a] \in D_j$ **then**
5 Check and update($minSupp$) with value stored in $Last[C_{ij}, X_i, a]$
6 **next** a
7 **if** $\nexists b \in D_j$ *such that* $(X_i = a, X_j = b) \in (C_{ij}, t)$ **then**
8 remove a from D_i; $change_i = 1$
9 **else**
10 $Last[C_{ij}, X_i, a] = b$; $Last[C_{ij}, X_j, b] = a$; $suppInv[b] = 1$
11 Check and update($minSupp$) with value b
12 **if** $([(t = 2 \vee t = 3) \wedge change_i = 1] \vee t = 1)$ **then**
13 **for** *each* $b \in D_j$ **do**
14 **if** $b < minSupp$ **then**
15 remove b from D_j; $change_j = 2$
16 **else**
17 **if** $suppInv[b] > 0$ **then**
18 $suppInv[b] = 0$
19 **else**
20 **if** $\nexists a \in D_i$ *such that* $(X_i = a, X_j = b) \in (C_{ij}, t)$ **then**
21 remove b from D_j; $change_j = 2$
else $Last[C_{ij}, X_j, b] = a$
22 $change = change_i + change_j$
23 **return** $change$
end

[2] if t=2 the inverse operator is used

then C_{ij} is not needed to be checked due to the fact that in the previous loop, the constraint has not generated any prune. However, if $t = 1$ then C_{ij} requires full bidirectional evaluation. If $t = 2$ or $t = 3$, and $change_i = 1$ then C_{ij} also requires full bidirectional evaluation.

Experimental Results

As it can be observed, the developed algorithm can be used for general purposes, Thus, in this section we compare the behavior of 2-C3OPL in two different types of problems: random problems and benchmark problems (the railway scheduling problem).

Determining which algorithms are superior to others remains difficult. Algorithms take often been compared by observing its performance on benchmark problems or on suites of random instances generated from a simple, uniform distribution. On the one hand, the advantage of using benchmark problems is that if they are an interesting problem (to someone), then information about which algorithm works well on these problems is also interesting. However, although an algorithm outperforms to any other algorithm in its application to a concrete benchmark problem, it is difficult to extrapolate this feature to general problems. On the other hand, an advantage of using random problems is that there are many of them, and researchers can design carefully controlled experiments and report averages and other statistics. However, a drawback of random problems is that they may not reflect real life situations. Here, we analyzed the number of constraint checks, the number of propagations and the running time as a measure of efficiency. All algorithms were written in C. The experiments were conducted on a PC Intel Core 2 Quad (2.83 GHz processor and 3 GB RAM).

Random problems

The experiments performed on random and non-normalized instances were characterized by the 5-tuple $< n, d, m, c, p >$, where n was the number of variables, d the domain size, m the number of binary constraints, c the number of constraints in each block and p the percentage of non-normalized constraints. The instances were randomly generated in intensionally form. We evaluated 50 test cases for each type of problem. It must be taken into account that the random instances represent large problems with hundreds of variables with large domains, and thousands of constraints.

Table 1 shows the behavior of the three techniques in both consistent and inconsistent instances. Regarding consistent instances, Table 1 shows that both 2-C3 and 2-C3OPL were able to prune more search space (80%) than AC3 in all instances. This is due to the fact that AC3 analyzed each constraint individually meanwhile 2-C3 and 2-C3OPL studied the block of constraints (see Example of Figure 1). Furthermore, 2-C3OPL needs performing less constraints checks than 2-C3 and AC3, because 2-C3OPL remembering supports founded in $Last$ structure. As the number of prunes is very important in filtering techniques, we use two ratios to compare the behavior of these algorithms: checks/prunes and prunes/time. In both ratios, 2-C3OPL was more efficient than both AC3 and 2-C3. For instance, in problems $< 800, 200, 8000, 3, 0.8 >$, 2-C3OPL carried out 12480 prunes per second, while AC3 carried out 7706 prunes per second. The behavior of the approach was also improved in non-consistent instances. 2-C3OPL was able to detect inconsistency quickly. 2-C3OPL carried out both a better runtime and a less effort (in prunes and checks) than both AC3 and 2-C3 in all instances. In large instances, for instances in problems $< 800, 200, 3000, 4, 0.8 >$ the improvement in runtime is one order of magnitude.

Benchmark problems of the railway scheduling problem

As we have pointed out in the introduction, the railway scheduling problem proposed here is a simpler instance of the railway scheduling problem proposed in (Ingolotti 2007) due to the fact that the we initially assume an empty network so that no previous trains are previously scheduled in the network. The generated CSP for the railway scheduling problem contains all the variables and constraints mentioned above. All constraints are binary and they were presented in intensional form. Each problem has non-normalized constraints. The number of variables and constraints in our CSP formulation are determined by the number of trains T and the number of locations L. If L or T increases then both the number of variables and the number of constraint increases as well. The number of constraints does not change when the frequency values F_D, F_U or the delay value λ are modified. However these variations influence on the tightness of the constraints. The combinations of trains and frequencies, used in our evaluation, are shown in Tables 2 and 3, respectively.

Table 2: Combination of values for frequency F and delay λ used in the railway scheduling problem.

Combination name	F		delay
	F_D	F_U	λ
F1	100	120	2
F2	100	120	5
F3	100	120	10
F4	150	170	2
F5	150	170	5
F6	150	170	10
F7	100	150	30

Our instances were carried out on a real railway infrastructure that joins Spanish locations, using data of the Administrator of Railway Infrastructures of Spain (ADIF). We consider two journeys:

- *Zaragoza - Caset*: This journey consist of 7 locations. In all these test cases the number of locations (5 stations and 2 halts) is fixed, however the number of trains, the frequency and the delay, are changed for each each test case.
- *Zaragoza - Calat*: This journey consist of 25 locations. In all these test cases the number of trains is fixed to 6, however the number of locations (any intermedia location between Zaragoza and Calat), the frequency and the delay, are changed for each each test case.

Table 1: Number of pruning, runtime (sec.) and checks by using AC3, 2-C3 and 2-C3OPL in random non-normalized instances.

tuple	**Arc-consistency**			**2-consistency**						instance
	AC3			**2-C3**			**2-C3OPL**			
	prune	*time*	*checks*	*prune*	*time*	*checks*	*prune*	*time*	*checks*	
$\langle 600, 200, 8000, 4, 0.8\rangle$	12600	2.01	1.0×10^8	60600	9.59	8.8×10^7	60600	5.9	4.7×10^7	consistent
$\langle 600, 200, 8000, 3, 0.8\rangle$	12600	2.25	1.2×10^8	60000	9.65	9.0×10^7	60000	5.91	4.8×10^7	consistent
$\langle 800, 200, 8000, 4, 0.8\rangle$	16800	2.14	1.1×10^8	80800	9.89	9.0×10^7	80800	6.48	4.9×10^7	consistent
$\langle 800, 200, 8000, 3, 0.8\rangle$	16800	2.18	1.2×10^8	80000	9.84	9.2×10^7	80000	6.41	5.0×10^7	consistent
$\langle 600, 100, 3000, 3, 0.6\rangle$	46745	8.38	1.0×10^8	43180	6.75	3.4×10^7	34906	1.13	4.8×10^6	inconsistent
$\langle 600, 100, 3000, 4, 0.6\rangle$	56176	13.41	9.1×10^7	48011	6.78	2.8×10^7	34141	1.25	4.8×10^6	inconsistent
$\langle 800, 200, 8000, 3, 0.8\rangle$	148227	133.81	1.6×10^9	127351	9.38	5.9×10^8	77513	7.4	4.6×10^7	inconsistent
$\langle 800, 200, 8000, 4, 0.8\rangle$	125141	106.99	1.4×10^9	115302	8.34	6.1×10^8	89712	6.4	4.5×10^7	inconsistent

For each journey (Zaragoza-Caset and Zaragoza-Calat), we assign parameters L, $T_i, (1 \leq i \leq 4)$, and $F_j, (1 \leq j \leq 7)$. For each assignment, a new instance is obtained and the corresponding CSP $\langle n, d, m\rangle$. It must be taken into account that although two different instances generate the same CSP tuple $\langle n, d, m\rangle$, they represent two different CSP instances with different tightness.

Table 3: Combination of trains T used in the railway scheduling problem.

Combination name	**T**		Number of Trains
	T_D	T_U	
T1	3	3	6
T2	4	4	8
T3	5	5	10
T4	6	6	12

Table 4 shows the pruned values, the running time and the number of constraint checks in the railway scheduling problem *Zaragoza - Caset* for 9 different instances. Each instance was defined by: the number of locations ($L = 7$), the number of trains ($T = 6; T_D = 3, T_U = 3$) and one of the six frequency combinations (F1 to F6). The CSP tuple generated by the 9 instances defined above was $\langle 86, 3600, 413\rangle$. The results of Table 4 shows that in instances from F1 to F3, AC3 could not detect inconsistency, while both: 2-C3OPL and 2-C3 (2-consistency techniques) detected them quickly. This is due to the fact that AC3 analyzed each constraint individually meanwhile 2-C3 and 2-C3OPL studied the block of constraints. Thus, both 2-C3OPL and 2-C3 were more efficient than AC3 because they detected inconsistencies earlier and generate a small number of prunes and checks.

Table 4 also shows that the number of constraint checks in 2-C3OPL was smaller than both AC3 and 2-C3 when the instances were consistent. 2-C3OPL needs performing less constraints checks than both 2-C3 and AC3, because 2-C3OPL storing supports in the $Last$ structure. The checks avoided by 2-C3OPL improved its running time in 70% with relation to 2-C3 for consistent instances. However the additional storage structures required by 2-C3OPL spend time meanwhile AC3 saves this time.

Table 5 shows the pruned values, the running time and the number of constraint checks in the railway scheduling problem *Zaragoza - Caset* for 6 different instances. Each instance was defined by: $L = 7, T = 12; T_D = 6, T_U = 6$ and one of the six frequency combinations (F1 to F6). The CSP tuple generated by the six instances defined above was $\langle 170, 3600, 1376\rangle$. The results of Table 5 are similar to those obtained in Table 4. Our tecniques are able to detect inconsistencies quickly.

Table 6 shows the pruned values, the running time and the number of constraint checks in the railway scheduling problem *Zaragoza - Calat* for 6 different instances. Each instance was defined by: $L = 5$, $T = 6$ ($T_D = 3, T_U = 3$) and one of the six frequency combinations (F1 to F6). The domain size was increased to 5000 seconds. The CSP tuple generated by the 6 instances defined above was $\langle 62, 5000, 299\rangle$. With these instances, the improvement in number of checks was one order of magnitude for consistent problems, although the running time was worse.

Finally, Table 7 shows the pruned values, the running time and the number of constraint checks in the railway scheduling problem *Zaragoza-Caset*, where $L = 7$, $F = F7$ (see Table 2) and the number of trains was increased from 6 to 12 (see Table 3). Due to the parameter T is increased, both the number of variables and constraints were increased for the resultant CSP. The results show that our 2-consistency algorithms achieved 2% more pruning than AC3. In the ratio checks/prunes, 2-C3OPL was more efficient than both: AC3 and 2-C3. For instance, in the problem $< 86, 3600, 413 >$, 2-C3OPL carried out 10364 checks per prune, while 2-C3 and AC3 carried out 41068 and 34774 checks per prune, respectively.

Conclusions and Further Work

Railway scheduling is a real world problem that can be modeled as a Constraint Satisfaction Problem (CSP). The formulation of the railway scheduling problem generates a large amount of variables with broad domains. Therefore, consistency techniques become an important issue in order to prune the search space and to improve the search process.

AC3 is one of the most well-known arc consistency algorithms and different versions have improved the efficiency of the original one. In this paper, we have presented 2-C3OPL algorithm to achieve 2-consistency in binary and non-normalized CSPs. 2-C3OPL is an optimized and reformulated version of 2-C3 that improves the efficiency of

Table 4: Number of pruning, runtime (sec.) and checks for the railway scheduling problem with $T = T1$ and $L = 7$. The CSP formulation is $\langle 86, 3600, 413 \rangle$ and F was increased from F1 to F6 by using AC3, 2-C3 and 2-C3OPL.

Frequency	**Arc-consistency**			**2-consistency**						Observation
	AC3			**2-C3**			**2-C3OPL**			
	prune	*time*	*checks*	*prune*	*time*	*checks*	*prune*	*time*	*checks*	
F1	65300	31.6	2.3×10^9	I 3600	4.7	3.8×10^7	I 3600	4.8	3.8×10^7	Inconsistency is not detected by AC3
F2	65300	31.8	2.3×10^9	I 3600	4.7	3.8×10^7	I 3600	4.8	3.8×10^7	Inconsistency is not detected by AC3
F3	65300	31.6	2.3×10^9	I 3600	4.7	3.8×10^7	I 3600	4.8	3.8×10^7	Inconsistency is not detected by AC3
F4	70300	30.1	2.2×10^9	70300	289.3	2.6×10^9	70300	77.9	6.8×10^8	
F5	70300	30.3	2.2×10^9	70300	289.6	2.6×10^9	70300	77.8	6.8×10^8	
F6	70300	30.0	2.2×10^9	70300	289.4	2.6×10^9	70300	77.7	6.8×10^8	

Table 5: Number of pruning, runtime (sec.) and checks for the railway scheduling problem with $T = T4$ and $L = 7$. The CSP formulation is $\langle 170, 3600, 1376 \rangle$ and F was increased from F1 to F6 by using AC3, 2-C3 and 2-C3OPL.

Frequency	**Arc-consistency**			**2-consistency**						Observation
	AC3			**2-C3**			**2-C3OPL**			
	prune	*time*	*checks*	*prune*	*time*	*checks*	*prune*	*time*	*checks*	
F1	161490	62.9	4.5×10^9	I 3600	4.8	3.8×10^7	I 3600	5.1	3.8×10^7	Inconsistency is not detected by AC3
F2	161490	63.0	4.5×10^9	I 3600	4.8	3.8×10^7	I 3600	5.1	3.8×10^7	Inconsistency is not detected by AC3
F3	161490	62.8	4.5×10^9	I 3600	4.9	3.8×10^7	I 3600	5.1	3.8×10^7	Inconsistency is not detected by AC3
F4	185990	57.3	4.1×10^9	185990	512.3	4.6×10^9	185990	151.0	1.3×10^9	
F5	185990	57.1	4.1×10^9	185990	512.5	4.6×10^9	185990	151.0	1.3×10^9	
F6	185990	57.3	4.1×10^9	185990	512.5	4.6×10^9	185990	151.0	1.3×10^9	

Table 6: Number of pruning, runtime (sec.) and checks for the railway scheduling problem with $T = T1$ and $L = 5$. The CSP formulation is $\langle 62, 5000, 299 \rangle$ and F was increased from F1 to F6 by using AC3, 2-C3 and 2-C3OPL.

Frequency	**Arc-consistency**			**2-consistency**						Observation
	AC3			**2-C3**			**2-C3OPL**			
	prune	*time*	*checks*	*prune*	*time*	*checks*	*prune*	*time*	*checks*	
F1	36356	39.3	2.8×10^9	I 5000	9.1	7.4×10^7	I 5000	9.2	7.4×10^7	Inconsistency is not detected by AC3
F2	36356	39.3	2.8×10^9	I 5000	9.1	7.4×10^7	I 5000	9.2	7.4×10^7	Inconsistency is not detected by AC3
F3	36356	39.3	2.8×10^9	I 5000	9.2	7.4×10^7	I 5000	9.3	7.4×10^7	Inconsistency is not detected by AC3
F4	40156	37.3	2.7×10^9	40156	380.0	3.3×10^9	40156	110.7	9.9×10^8	
F5	40156	37.3	2.7×10^9	40156	379.2	3.3×10^9	40156	111.0	9.9×10^8	
F6	40156	37.4	2.7×10^9	40156	379.8	3.3×10^9	40156	110.9	9.9×10^8	

Table 7: Number of pruning, runtime (sec.) and checks for the railway scheduling problem with $L = 7$, $F = F7$, domain size $D = 3600$ and the number of trains was increased from T1 to T4 by using AC3, 2-C3 and 2-C3OPL.

CSP tuple	**Arc-consistency**			**2-consistency**						Observation
	AC3			**2-C3**			**2-C3OPL**			
	prune	*time*	*checks*	*prune*	*time*	*checks*	*prune*	*time*	*checks*	
$\langle 86, 3600, 413 \rangle$	65300	31.6	2.3×10^9	66340	311.3	2.7×10^9	66340	78.4	6.7×10^8	lower pruning in AC3
$\langle 114, 3600, 720 \rangle$	93798	42.4	3.0×10^9	95838	415.7	3.7×10^9	95838	105.9	9.3×10^8	lower pruning in AC3
$\langle 142, 3600, 1048 \rangle$	125868	52.8	3.8×10^9	129208	487.2	4.4×10^9	129208	130.4	1.1×10^9	lower pruning in AC3
$\langle 170, 3600, 1376 \rangle$	161490	62.7	4.5×10^9	166450	560.3	5.0×10^9	166450	154.3	1.3×10^9	lower pruning in AC3

previous one by reducing the number of propagations, the number of constraint checks and the running time. The proposed technique is domain-independent. The 2-C3OPL was applied to the railway scheduling problem because it is a non-normalized problem and this problem can benefit from a more strong consistency like 2-consistency. The evaluation section shows that 2-C3OPL had a better behavior than AC3 and 2-C3 in both consistent and inconsistent random instances. This technique is also being applied to the railway scheduling problem, which has a set of non-normalized constraints and 2-consistency remains more efficient than arc-consistency. The evaluation section shows that 2-C3OPL had a better behavior than AC3 in inconsistent instances and 2-C3OPL had a better behavior than 2-C3 in consistent in-

stances. Furthermore, AC3 was unable to detect some inconsistencies (see evaluation section) while both 2-C3OPL and 2-C3 detected the inconsistency efficiently. Railway operators are very interesting on detecting inconsistencies in preliminary/tentative timetables so that our techniques remain useful to determine inconsistencies in the first steps of a timetable generation. Also shows that the running time is reduced up to 75% and the number of constraint checks is reduced up to 50% in relation to 2-C3 for consistent instances.

In further work, we will focus our attention to apply these filtering techniques to MAC2-C in order to improve the efficiency during search.

Acknowledgments

This work has been partially supported by the research projects TIN2007-67943-C02-01 (Min. de Educación y Ciencia, Spain-FEDER) and P19/08 (Min. de Fomento, Spain-FEDER)

References

Arangú, M.; Salido, M.; and Barber, F. 2009a. 2-C3: From arc-consistency to 2-consistency. In *SARA 2009*.

Arangú, M.; Salido, M.; and Barber, F. 2009b. Normalizando CSP no-normalizados: un enfoque híbrido. In *CAEPIA-TTIA 2009. Workshop on Planning, Scheduling and Constraint Satisfaction.*, 57–68.

Barber, F.; Abril, M.; Salido, M. A.; Ingolotti, L.; Tormos, P.; and Lova, A. 2007. Survey of automated systems for railway management. Technical report, DSIC-II/01/07.UPV.

Barták, R. 2001. Theory and practice of constraint propagation. In Figwer, J., ed., *Proceedings of the 3rd Workshop on Constraint Programming in Decision and Control*.

Bessiere, C., and Cordier, M. 1993. Arc-consistency and arc-consistency again. In *Proc. of the AAAI'93*, 108–113.

Bessiere, C.; Régin, J. C.; Yap, R.; and Zhang, Y. 2005. An optimal coarse-grained arc-consistency algorithm. *Artificial Intelligence* 165:165–185.

Bessiere, C.; Freuder, E.; and Régin, J. C. 1999. Using constraint metaknowledge to reduce arc consistency computation. *Artificial Intelligence* 107:125–148.

Bessiere, C. 1994. Arc-consistency and arc-consistency again. *Artificial Intelligence* 65:179–190.

Bessiere, C. 2006. Constraint propagation. Technical report, CNRS.

Cooper, M. 1994. An optimal k-consistency algorithm. *Artificial Intelligence* 41:89–95.

Dechter, R. 2003. *Constraint Processing*. Morgan Kaufmann.

Freuder, E. 1978. Synthesizing constraint expressions. *Communications of the ACM* 21:958–966.

Ingolotti, L. 2007. *Modelos y métodos para la optimización y eficiencia de la programación de horarios ferroviarios*. Ph.D. Dissertation, Universidad Politécnica de Valencia.

Lecoutre, C., and Hemery, F. 2007. A study of residual supports in arc consistency. In *proceedings IJCAI 2007*, 125–130.

Mackworth, A. K. 1977. Consistency in networks of relations. *Artificial Intelligence* 8:99–118.

Mohr, R., and Henderson, T. 1986. Arc and path consistency revised. *Artificial Intelligence* 28:225–233.

Rossi, F.; Van Beek, P.; and Walsh, T. 2006. *Handbook of constraint programming*. Elsevier.

Salido, M.; Barber, F.; Abril, M.; Ingolotti, L.; Lova, A.; Tormos, P.; and Estarda, J. 2005. Técnicas de inteligencia artificial en planificación ferroviaria. In Thomson., ed., *VI Session on Artificial Intelligence Technology Transfer (TTIA '2005). Proceedings of TTIA'2005*, 11–18.

Silva de Oliveira, E. 2001. *Solving Single-Track Railway Scheduling Problem Using Constraint Programming*. Ph.D. Dissertation, University of Leeds, School of Computing.

Tormos, P.; Lova, A.; Barber, F.; Ingolotti, L.; Abril, M.; and Salido, M. 2008. A genetic algorithm for railway scheduling problems. *Metaheuristics for Scheduling In Industrial and Manufacturing Applications, ISBN: 978-3-540-78984-0* 255–276.

Walker, C.; Snowdon, J.; and Ryan, D. 2005. Simultaneous disruption recovery of a train timetable and crew roster in real time. *Comput. Oper. Res* 2077–2094.