Faculty of Computer Science Institute of Artificial Intelligence Knowledge Representation and Reasoning

Polynomial Reduction from PESP to SAT

Peter Großmann

KRR Report 11-05

Großer Beleg

Polynomial Reduction from PESP to SAT

Peter Großmann

October 21, 2011

Supervised by:
Prof. Dr. rer. nat. habil. Steffen Hölldobler
Dipl-Inf. Peter Steinke

Abstract

SAT solvers have been already successfully applied in several industrial fields, which are not directly related to propositional logic. In this work, the periodic event scheduling problem (PESP) will be presented and, furthermore, the order encoding from a PESP instance to a SAT instance. The $\mathcal{NP}$-complete PESP is particularly important in several traffic scenarios with periodic properties. Most native domain solvers cannot solve large instances within a given time frame. This will be omitted by a rather short time consuming conversion from PESP to SAT and a fast state-of-the-art SAT solver, in order to achieve a fast calculated solution of a PESP instance.

Contents

1. Introduction

Modelling and automatic solving of large periodic event networks, like a timetable for the railway network of Germany, is still an open scientific field. The model, the so called periodic event scheduling problem (PESP), is well defined, but lacks in efficient runtime behaviour for all state-of-the-art PESP solvers, if the instances become very large. Since the PESP is a $\mathcal{NP}$-complete problem [Odi94], there will always be instances, which cannot be solved in any reasonably given time frame, unless $\mathcal{P} = \mathcal{NP}$. However, for a lot of $\mathcal{NP}$-hard problems, there exist efficient algorithms to solve very hard and huge instances in an acceptable solving time.

One example is the solving of propositional formulas – the so called SAT solving. Several industrial problems have already been reformulated as propositional formulas and been solved by SAT solvers [CDE08, CBRZ01, MZ06]. Hence, these are generic solvers to solve different domains.

This work tries to follow this approach: reformulating a model of the periodic event network domain to a propositional formula and solving this instance with a state-of-the-art SAT solver. The transition, or reduction, to this formula is called encoding. In this work two different encodings will be presented to reduce a PESP instance into a SAT instance.

First of all, the two domains, periodic event networks and propositional logic, will be introduced, in order to get a good understanding of the core of this work itself. Having a good knowledge about these fields, the two encodings itself will be presented. On the one hand, the direct encoding approach will be introduced, which will yield a rather large propositional formula in contrast to the second approach. On the other hand, knowing the disadvantages of the previous method, the order encoding tries to keep the structure and behaviour of the periodic event network and being rather small.

The core idea of the order encoding, with respect to the PESP domain, is the utilization of the constraint's definition, which is strictly linear – even the identity function. This information can be greatly used for a domain like the propositional formulas, which just have $true$ and $false$ as domain of its variables.

Another effort in this work will be done by proofing both methods being sound and correct. Consequently, the use of these methods is valid and can replace a traditional PESP solver.

After comparing the two encodings itself, with respect to different size types, all approaches will be applied, evaluated and compared to real industrial examples with respect to creating valid timetables for railway networks. The instances range from rather small instances to very large instances. Thus, it gives an interesting perspective for the usage of the presented new methods.

In the end, the work will be concluded by remaining questions and open scientific areas, which could result in even better runtime behaviours than evaluated in the used examples.

2. Preliminaries

In this section will be introduced both propositional logic and periodic event networks. Because both themes are important for this work, it is essential to introduce them elementarily. This section ends in a short overview about one state-of-the-art PESP solver.

2.1. Propositional Logic

Propositional logic is a fundamental topic in computer science and other areas. More and more industrial problems are encoded into propositional logic [CDE08, CBRZ01, MZ06]. Hence, the need of fast solvers, which can solve these problems in short time, increases. In this work, propositional logic will be introduced as binary logic, which values $true$ and $false$ are very well known. An in depth overview of this topic can be found in [Höl09]. The definitions in the next sections are following the definitions as of [Höl09].

2.1.1. Syntax

Definition 2.1 (Alphabet of Propositional Logic)**.** *The* alphabet of propositional logic *Σ_{SAT} consists of an countably infinite set of propositional variables $\mathcal{R} = \{p_1, p_2, \ldots\}$, the brackets "(" and ")", as well as the set of connectives $\{\wedge, \vee, \neg\}$. Both the $\wedge$ and $\vee$ connectives are binary, whereas the $\neg$ connective is unary.*

These connectives are called conjunction, disjunction and negation respectively.

Definition 2.2 (Propositional formula)**.** *A string F, that consists only letters of the alphabet Σ_{SAT} is called* propositional formula, *if and only if it fulfills one of the properties*

1. *if $F = p$, then $p \in \mathcal{R}$ or*
2. *if $F = \neg G$, then G is a propositional formula or*
3. *if $\circ$ is a binary connective and $F = (G \circ H)$, then G and H are propositional formulas.*

Let $\mathcal{L}(\Sigma_{SAT})$ be the smallest set, that contains each propositional formula under the alphabet Σ_{SAT}.

Because Σ_{SAT} is an alphabet, $\mathcal{L}(\Sigma_{SAT})$ is also called the language (syntax) of the propositional logic.

Example 2.1. Let $p, q, r \in \mathcal{R}$. The propositional formula

$$F = ((((p \vee q) \vee r) \wedge (\neg p \vee \neg r)) \wedge (\neg p \vee \neg q)) \in \mathcal{L}(\Sigma_{\mathsf{SAT}})$$

is constructed as conjunction of disjunctions of variables or negated variables. This special form will be precisely defined in the following.

This example will be used throughout this section.

Definition 2.3 (Literal). *A propositional formula $L \in \mathcal{L}(\Sigma_{SAT})$ is called* literal*, if and only if $L = p$ or $L = \neg p$, with $p \in \mathcal{R}$.*

Definition 2.4 (Clause). *A propositional formula $C \in \mathcal{L}(\Sigma_{SAT})$ is called a* clause*, if it is a disjunction of literals. Hence, with $n \geq 0$*

$$C = (\ldots(L_1 \vee L_2) \vee \ldots) \vee L_n),$$

where L_i $(i \in \{1, \ldots, n\})$ are literals. For convenience the clause C can be written as

$$C = [L_1, \ldots, L_n].$$

Definition 2.5 (Conjunctive Normal Form). *A propositional formula $F \in \mathcal{L}(\Sigma_{SAT})$ is in* conjunctive normal form *(denoted as CNF), if it is a conjunction of clauses. Thus, with $m \geq 0$*

$$F = (\ldots(C_1 \wedge C_2) \wedge \ldots) \wedge C_m),$$

where C_i $(i \in \{1, \ldots, m\})$ are clauses. For convenience the formula F is denoted as

$$\begin{aligned} F &= \langle C_1, \ldots, C_m \rangle \\ &= \langle [L_{1,1}, \ldots, L_{1,n_1}], \ldots, [L_{m,1}, \ldots, L_{m,n_m}] \rangle, \end{aligned} \tag{1}$$

with literals $L_{i,j}$ $\forall i \in \{1, \ldots, m\}$, $j \in \{1, \ldots, n_i\}$, $n_i \geq 0$.

Example 2.2. Let F be the propositional formula from Example 2.1. Since it is in conjunctive normal form, it can be written as

$$F = \langle [p, q, r], [\neg p, \neg r], [\neg p, \neg q] \rangle$$

2.1.2. Semantic

In order to decide, whether a propositional formula is $true$ or $false$ under a certain variable assignment, the formula must be evaluated by a so called interpretation. Since this work contains only propositional formulas in conjunctive normal form, it is sufficient to regard only this subset of $\mathcal{L}(\Sigma_{\text{SAT}})$.

Definition 2.6 (Interpretation). *Let $F \in \mathcal{L}(\Sigma_{SAT})$ a propositional formula. Then the mapping*

$$I : \mathcal{L}(\Sigma_{SAT}) \rightarrow \{true, false\}$$
$$F^I = w$$

is called Interpretation *with $w \in \{true, false\}$.*

It is sufficient to assign each propositional variable $p \in \mathcal{R}$ a truth value [Höl09], such that $p^I = w$, since all evaluations of the connectives $\{\wedge, \vee, \neg\}$ under I are known in beforehand. Thus, we can easily evaluate F under I with

$$F^I = \begin{cases} \neg(G^I) & \text{if } F = \neg G \\ (G^I \circ H^I) & \text{if } F = G \circ H \end{cases}$$

with $G, H \in \mathcal{L}(\Sigma_{\text{SAT}})$ and $\circ \in \{\wedge, \vee\}$.

The evaluation of an interpretation will be shown in the following definitions.

Definition 2.7. *A propositional formula F is satisfiable, if and only if there exists an interpretation I, such that $F^I = true$.*

A propositional formula F is unsatisfiable, if and only if $F^I = false$ for all interpretations I.

Definition 2.8. *A literal L is satisfied under an interpretation I, if*

1. *$L = p \in \mathcal{R}$ and $p^I = true$ or*
2. *$L = \neg p$, $p \in \mathcal{R}$ and $p^I = false$.*

Definition 2.9. *A clause $C = [L_1, \dots, L_n]$ is satisfied under an interpretation I, if at least one literal L_i $(i \in \{1, \dots, n\})$ is satisfied under I.*

Definition 2.10. *A propositional formula $F = \langle C_1, \dots, C_m \rangle$ in CNF is satisfied under an interpretation I, if all clauses C_i $(i \in \{1, \dots, m\})$ are satisfied under I.*

Definition 2.11 (Model)**.** *Let $F \in \mathcal{L}(\Sigma_{SAT})$ be a propositional formula and I an interpretation. I is called* model, *if and only if $F^I = true$. F is then satisfied with model I.*

If I is a model for F, it can be denoted as

$$I \models F.$$

Example 2.3. Let F be the propositional formula in CNF from Example 2.2 with

$$F = \langle [p, q, r], [\neg p, \neg r], [\neg p, \neg q] \rangle.$$

Given the interpretation I with $p^I = true, r^I = false, q^I = false$, F can be evaluated under I:

$$p^I = true,\ p \text{ is a literal, } p \text{ occurs in } [p, q, r] \overset{\text{Def 2.8,2.9}}{\Longrightarrow} [p, q, r]^I = true \quad (2)$$

$$r^I = false,\ \neg r \text{ is a literal, } \neg r \text{ occurs in } [\neg p, \neg r] \overset{\text{Def 2.8,2.9}}{\Longrightarrow} [\neg p, \neg r]^I = true \quad (3)$$

$$q^I = false,\ \neg q \text{ is a literal, } \neg q \text{ occurs in } [\neg p, \neg q] \overset{\text{Def 2.8,2.9}}{\Longrightarrow} [\neg p, \neg q]^I = true \quad (4)$$

With (2), (3), (4) and Definition 2.10 it follows, that $F^I = true$.

Thus, I is a model for F, which can be denoted as well with $I \models F$.

Definition 2.12 (Semantic Equivalence)**.** *Let F and G be propositional formulas. F and G are* semantically equivalent, *denoted as*

$$F \equiv G,$$

if and only if for all interpretations I

$$F^I = G^I.$$

Since the propositional logic is introduced as binary ($true$ and $false$) logic in this work, semantically equivalence holds as well for F and G, if and only if for all interpretations I

$$I \models F \Leftrightarrow I \models G$$

Example 2.4 (without proofs)**.** Let F, G and H be propositional formulas. Then

$$\begin{aligned} \neg\neg F &\equiv F & \\ \neg(F \wedge G) &\equiv (\neg F \vee \neg G) & \text{(de Morgan)} \\ \neg(F \vee G) &\equiv (\neg F \wedge \neg G) & \text{(de Morgan)} \\ ((F \vee G) \vee H) &\equiv (F \vee (G \vee H)) & \text{(associativity)} \\ ((F \wedge G) \wedge H) &\equiv (F \wedge (G \wedge H)) & \text{(associativity)} \\ (F \vee G) &\equiv (G \vee F) & \text{(commutativity)} \\ (F \wedge G) &\equiv (G \wedge F) & \text{(commutativity)} \end{aligned}$$

The presented examples will be widely used in Section 3. Since associativity holds for propositional formulas, the parentheses will often be skipped in the remaining work.

2.1.3. SAT Problem

There exist several classifications of the SAT problem [Höl09]. This work will concentrate on SAT being a decision problem, which either says, that the given formula is satisfiable under a certain interpretation or that the formula is unsatisfiable. This decision can be computationally calculated by so called SAT solvers.

Definition 2.13 (SAT Problem)**.** *Let $F \in \mathcal{L}(\Sigma_{SAT})$ be a propositional formula in CNF. It shall be decided, whether*

1. *F is unsatisfiable or*
2. *F is satisfiable, if $\exists I : I \models F$, return I.*

Example 2.5. Let F be the propositional formula in CNF from Example 2.2. Since there exists an interpretation I, such that I is a model for F (Example 2.3), the solution of the SAT problem would return this evaluation.

Because it is well known, that the SAT problem is $\mathcal{NP}$-complete [Coo71], it is not tractable to solve this decision problem. Unless there does exist an algorithm with polynomial complexity that solves a $\mathcal{NP}$-complete problem, it will stay hard to solve such a SAT instance, which is a common open problem in computer science, that is known as $\mathcal{P} \stackrel{?}{=} \mathcal{NP}$.

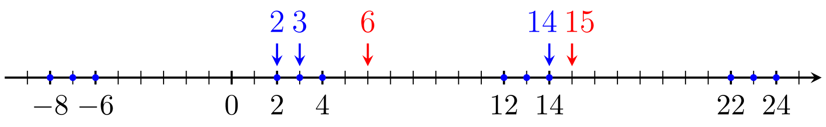


Figure 2.1: Visualization of Example 2.6

2.2. Periodic Event Networks

Several time scheduling problems have periodic properties, like the one presented in Section 4. Hence, a time event does not only happen once in time, but periodically often modulo a time bound. In order to restrict these events in some way, there will be introduced two different kind of constraints. Connecting events with constraints will result in a graph of a so called periodic event network. If all constraints hold for a certain assignment of all the events' potentials, then this assignment is called valid schedule. More in depth information can be found in [Opi09].

2.2.1. Periodic Event Network

Definition 2.14 (Interval). *Let $a, b \in \mathbb{Z}$ with $a \leq b$.*

$$[a, b] := \{a, a+1, \dots, b-1, b\}$$

is the interval *from a, the lower bound, to b, the upper bound.*

Further knowledge of interval arithmetic and linear subspaces can be gained in [Lau07].

Definition 2.15 (Modulo Interval). *Let $a, b \in \mathbb{Z}$ and $t \in \mathbb{N}_+$. With a and b being the lower and upper bound, respectively,*

$$[a, b]_t := \bigcup_{z \in \mathbb{Z}} [a + z \cdot t, b + z \cdot t] \subseteq \mathbb{Z}$$

is called interval modulo t.

Example 2.6. Let

$$\begin{aligned} I &= [2, 4]_{10} \\ &= \dots \cup [-8, -6] \cup [2, 4] \cup [12, 14] \cup [22, 24] \cup \dots \\ &= \{\dots, -8, -7, -6, 2, 3, 4, 12, 13, 14, 22, 23, 24, \dots\} \subset \mathbb{Z} \end{aligned}$$

be an interval modulo 10. Then

$$\begin{aligned} 2 &\in [2, 4] \subset I \\ 3 &\in [2, 4] \subset I \\ 6 &\notin I \\ 14 &\in [12, 14] \subset I \\ 15 &\notin I. \end{aligned}$$

This can be seen visualized in Figure 2.1.

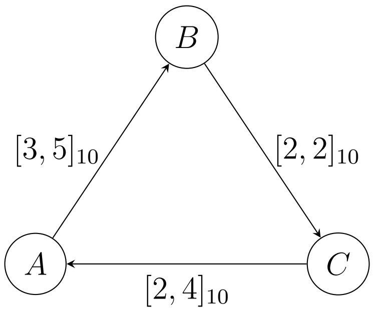


Figure 2.2: Periodic event network of Example 2.7

Definition 2.16 (Periodic Event Network)**.** *A periodic event network $\mathcal{N} = (\mathcal{V}, \mathcal{A}, t_T)$ with respect to period $t_T \in \mathbb{N}$ consists of periodic events $\mathcal{V}$ (nodes) and a set of constraints $\mathcal{A}$ (edges). Each constraint $a \in \mathcal{A}$ connects two periodic events, which is denoted as*

$$a = ((i,j), [l_a, u_a]_{t_T}) \in (\mathcal{V} \times \mathcal{V}) \times 2^{\mathbb{Z}},$$

with $l_a, u_a \in \mathbb{Z}$ being the lower and upper bound, respectively, t_T the period and $2^{\mathbb{Z}}$ the power set of $\mathbb{Z}$.

$\mathcal{A}$ is a union of $\mathcal{S}$ and $\mathcal{C}$ with $\mathcal{S} \cap \mathcal{C} = \emptyset$, where $\mathcal{S}$ and $\mathcal{C}$ are the sets of symmetry constraints and time consuming constraints, respectively.

Note that each $a \in \mathcal{A}$ is contained exactly once in one of the sets $\mathcal{C}$ and $\mathcal{S}$, since both are disjoint.

Definition 2.17. *$\mathcal{V}^+$ denotes the countably infinite set, which contains all possible nodes, a periodic event network could have.*

Likewise $\mathcal{A}^+_{t_T}$ denotes the countably infinite set set, which contains all possible constraints with respect to period t_T.

This technical definition will be needed in some function definitions.

The following example demonstrates a periodic event network.

Example 2.7. Let $\mathcal{N} = (\{A, B, C\}, \mathcal{A}, 10)$ be a periodic event network. With

$$\begin{aligned} \mathcal{A} = \mathcal{C} = \{&((A,B), [3,5]_{10}), \\ &((B,C), [2,2]_{10}), \\ &((C,A), [2,4]_{10})\}. \end{aligned}$$

This can be displayed as graph, which is shown in Figure 2.2

The exact definition of both constraint types will be clarified in Definition 2.20 and 2.21. Beforehand, there will be introduced the semantic part of the periodic event networks. In order to receive a schedule, it is needed to assign each event a point in time, when it happens.

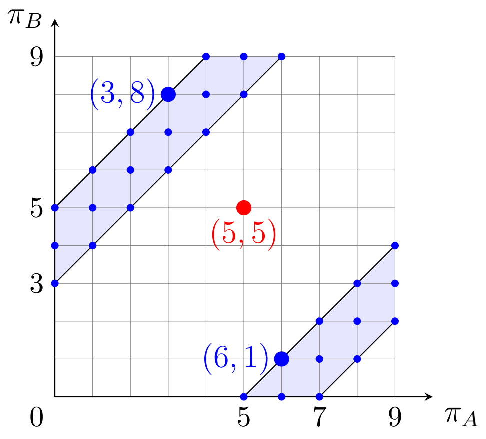


Figure 2.3: Feasible regions of the time consuming constraint for Example 2.8

Definition 2.18 (Event's Potential)**.** *Let $\mathcal{N} = (\mathcal{V}, \mathcal{A}, t_T)$ be a periodic event network. $\pi_n \in \mathbb{Z}$ is called* potential of event $n \in \mathcal{V}$.

Definition 2.19 (Schedule)**.** *Let $\mathcal{N} = (\mathcal{V}, \mathcal{A}, t_T)$ be a periodic event network. The mapping*

$$\begin{aligned} \Pi_{\mathcal{V}} : \mathcal{V} &\to \mathbb{Z} \\ n &\mapsto \pi_n \end{aligned}$$

is called schedule *of $\mathcal{N}$.*

The values $\pi_n = \Pi_{\mathcal{V}}(n)$ are the potentials of Definition 2.18.

Note, that for convenience, it is often only used π_n instead of introducing the schedule itself, since the exact schedule is not known beforehand, thus π_n can be used as variable.

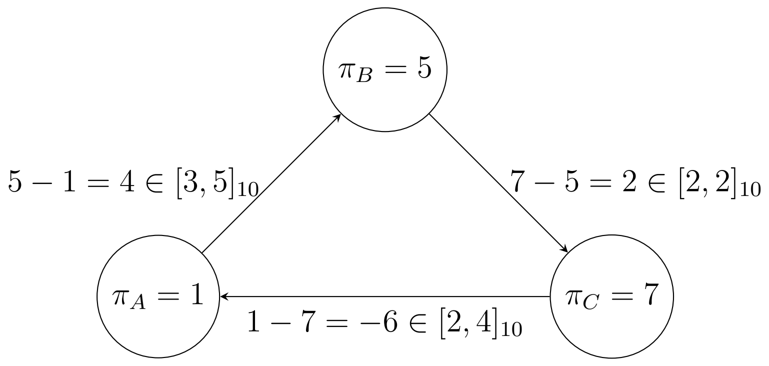
Definition 2.20 (Time Consuming Constraint)**.** *Let $\mathcal{N} = (\mathcal{V}, \mathcal{A}, t_T)$ be a periodic event network with $\mathcal{A} = \mathcal{S} \cup \mathcal{C}$ (see Definition 2.16) and $a = ((i,j), [l_a, u_a]_{t_T}) \in \mathcal{C}$ with $i, j \in \mathcal{V}$. For two potentials π_i, π_j the time consuming constraint a* holds, *if and only if*

$$\pi_j - \pi_i \in [l_a, u_a]_{t_T}.$$

This constraint describes a time consuming process, which describes, how much time within $[l_a, u_a]$ modulo t_T is needed from an event i to another event j, in order to hold with respect to a.

Definition 2.21 (Symmetry Constraint)**.** *Let $\mathcal{N} = (\mathcal{V}, \mathcal{A}, t_T)$ be a periodic event network with $\mathcal{A} = \mathcal{S} \cup \mathcal{C}$ (see Definition 2.16) and $a = ((i,j), [l_a, u_a]_{t_T}) \in \mathcal{S}$ with $i, j \in \mathcal{V}$. For two potentials π_i, π_j the symmetry constraint a* holds, *if and only if*

$$\pi_j + \pi_i \in [l_a, u_a]_{t_T}.$$



$\pi_B = 5$
$5 - 1 = 4 \in [3,5]_{10}$
$7 - 5 = 2 \in [2,2]_{10}$
$\pi_A = 1$
$\pi_C = 7$
$1 - 7 = -6 \in [2,4]_{10}$

Figure 2.4: Valid schedule for periodic event network of Example 2.9

Example 2.8. Let A, B be events and $a = ((A,B),[3,5]_{10})$ be a time consuming constraint (Definition 2.20).
If $\pi_A = 3$ and $\pi_B = 8$, then a holds, because $8 - 3 = 5 \in [3,5]_{10}$.
If $\pi_A = 5$ and $\pi_B = 5$, then a does not hold, because $5 - 5 = 0 \notin [3,5]_{10}$.
If $\pi_A = 6$ and $\pi_B = 1$, then a holds, because $1 - 6 = -5 \in [3,5]_{10}$.
In Figure 2.3 there are all feasible assignments with respect to π_A and π_B marked as blue circles, if a holds. The reason, why each domain of π_A and π_B are finitely displayed ($\{0,\ldots,9\}$) will be clarified in Corollary 2.2.

Definition 2.22. *Let $\mathcal{N} = (\mathcal{V},\mathcal{A},t_T)$ be a periodic event network and $\Pi_\mathcal{V}$ be a schedule of $\mathcal{N}$. A constraint $a \in \mathcal{A}$ holds* under $\Pi_\mathcal{V}$, *if and only if $a = (i,j,I)$ holds for $\pi_i = \Pi_\mathcal{V}(i)$ and $\pi_j = \Pi_\mathcal{V}(j)$.*

Definition 2.23 (Valid Schedule)**.** *Let $\mathcal{N} = (\mathcal{V},\mathcal{A},t_T)$ be a periodic event network and $\Pi_\mathcal{V}$ be a schedule of $\mathcal{N}$. $\Pi_\mathcal{V}$ is* valid *with respect to $\mathcal{A}$, if and only if all constraints $a \in \mathcal{A}$ hold under $\Pi_\mathcal{V}$.*

Because it is only interesting to look for valid schedules, it will be often called "schedule" in the remaining work.

Example 2.9. Let $\mathcal{N} = (\mathcal{V},\mathcal{A},10)$ be the periodic event network of Example 2.7 and $\Pi_\mathcal{V}$ a schedule of $\mathcal{N}$ with $\pi_A = 1, \pi_B = 5, \pi_C = 7.$. Then $\Pi_\mathcal{V}$ is valid, because

$$\begin{aligned}\pi_B - \pi_A &= 5 - 1 = 4 \in [3,5]_{10}\\ \pi_C - \pi_B &= 7 - 5 = 2 \in [2,2]_{10}\\ \pi_A - \pi_C &= 1 - 7 = -6 \in [2,4]_{10}.\end{aligned}$$

This is visualized in Figure 2.4.

Definition 2.24 (Equivalent Schedules)**.** *Let $\mathcal{N} = (\mathcal{V},\mathcal{A},t_T)$ be a periodic event network and $\Pi_\mathcal{V}$, $\Phi_\mathcal{V}$ be schedules of $\mathcal{N}$. $\Pi_\mathcal{V}$ and $\Phi_\mathcal{V}$ are called* equivalent, *denoted as*

$$\Pi_\mathcal{V} \equiv \Phi_\mathcal{V},$$

if and only if

$$\forall n \in \mathcal{V} : \Pi_{\mathcal{V}}(n) \bmod t_T = \Phi_{\mathcal{V}}(n) \bmod t_T.$$

Example 2.10. Let $\mathcal{N} = (\mathcal{V}, \mathcal{A}, 10)$ be the periodic event network with $\mathcal{V} = \{A, B, C\}$. The schedules $\Pi_{\mathcal{V}}$ and $\Phi_{\mathcal{V}}$ with

$$\pi_n = \Pi_{\mathcal{V}}(n), \varphi_n = \Phi_{\mathcal{V}}(n) \quad \forall n \in \mathcal{V}$$

and

$$\begin{aligned} &\pi_A = 2, \pi_B = 16, \pi_C = -3, \\ &\varphi_A = 12, \varphi_B = 6, \varphi_C = 7 \end{aligned}$$

are equivalent, because

$$\begin{aligned} \pi_A \bmod 10 = 2 = \varphi_A \bmod 10, \\ \pi_B \bmod 10 = 6 = \varphi_B \bmod 10, \\ \pi_C \bmod 10 = 7 = \varphi_C \bmod 10. \end{aligned}$$

Proposition 2.1 (Validity of Equivalent Schedules)**.** *Let $\mathcal{N} = (\mathcal{V}, \mathcal{A}, t_T)$ be a periodic event network and $\Pi_{\mathcal{V}}$, $\Phi_{\mathcal{V}}$ be schedules of $\mathcal{N}$ with $\Pi_{\mathcal{V}} \equiv \Phi_{\mathcal{V}}$. Then*

$$\Pi_{\mathcal{V}} \textit{ is valid} \Leftrightarrow \Phi_{\mathcal{V}} \textit{ is valid}$$

with respect to $\mathcal{A}$.

Proof. Without loss of generality it is sufficient to show "$\Rightarrow$".
$\Pi_{\mathcal{V}}$ is valid with respect to $\mathcal{A} \Rightarrow \forall a \in \mathcal{A}$:

$$a \text{ holds under } \Pi_{\mathcal{V}}. \tag{5}$$

Let $\mathcal{A} = \mathcal{S} \cup \mathcal{C}$. To show:

1. $\forall a \in \mathcal{C} : a$ holds under $\Phi_{\mathcal{V}}$

2. $\forall a \in \mathcal{S} : a$ holds under $\Phi_{\mathcal{V}}$

1. Let $a = ((i,j), [l_a, u_a]_{t_T}) \in \mathcal{C}$ be an arbitrary but fixed time consuming constraint with $i, j \in \mathcal{V}$, $\pi_i = \Pi_\mathcal{V}(i)$, $\pi_j = \Pi_\mathcal{V}(j)$ and $\varphi_i = \Phi_\mathcal{V}(i)$, $\varphi_j = \Phi_\mathcal{V}(j)$

$$\begin{aligned}
&\overset{(5)}{\Longrightarrow} && \pi_j - \pi_i \in [l_a, u_a]_{t_T} \\
&\overset{Def\ 2.15}{\Longrightarrow} && \pi_j - \pi_i \in \{[l_a + z \cdot t_T, u_a + z \cdot t_T] \mid z \in \mathbb{Z}\} \\
&\overset{z \in \mathbb{Z}}{\Longrightarrow} && \forall w \in \mathbb{Z} : \pi_j - \pi_i + w \cdot t_T \in \{[l_a + z \cdot t_T, u_a + z \cdot t_T] \mid z \in \mathbb{Z}\} \\
&\overset{z \in \mathbb{Z}}{\Longrightarrow} && \forall w, v \in \mathbb{Z} : \pi_j - \pi_i + w \cdot t_T - v \cdot t_T \\
& && \qquad \in \{[l_a + z \cdot t_T, u_a + z \cdot t_T] \mid z \in \mathbb{Z}\} \\
&\Longrightarrow && \forall w, v \in \mathbb{Z} : (\pi_j + w \cdot t_T) - (\pi_i + v \cdot t_T) \\
& && \qquad \in \{[l_a + z \cdot t_T, u_a + z \cdot t_T] \mid z \in \mathbb{Z}\} \\
&\Longrightarrow && (\pi_j \bmod t_T) - (\pi_i \bmod t_T) \in \{[l_a + z \cdot t_T, u_a + z \cdot t_T] \mid z \in \mathbb{Z}\} \\
&\overset{Def\ 2.15}{\Longrightarrow} && (\pi_j \bmod t_T) - (\pi_i \bmod t_T) \in [l_a, u_a]_{t_T} \\
&\overset{Def\ 2.24,\ \Pi_\mathcal{V} \equiv \Phi_\mathcal{V}}{\Longrightarrow} && (\varphi_j \bmod t_T) - (\varphi_i \bmod t_T) \in [l_a, u_a]_{t_T} \\
&\overset{\text{analog. } \pi_j - \pi_i}{\Longrightarrow} && \varphi_j - \varphi_i \in [l_a, u_a]_{t_T} \\
&\overset{Def\ 2.20}{\Longrightarrow} && a \text{ holds under } \Phi_\mathcal{V}
\end{aligned}$$

Since a is arbitrary, it holds for all $a \in \mathcal{C}$.
2. analogous. □

The following corollary has been already introduced, allthough slightly different, in [LM07].

Corollary 2.1. *It can be easily shown, that $\equiv$ forms an equivalence relation. Then*

$$\Pi_\mathcal{V} \textit{ is valid} \Rightarrow \forall \Phi_\mathcal{V} \in [\Pi_\mathcal{V}]_\equiv : \Phi_\mathcal{V} \textit{ is valid}$$

with $[\Pi_\mathcal{V}]_\equiv := \{\Phi_\mathcal{V} \mid \Phi_\mathcal{V} \equiv \Pi_\mathcal{V}\}$ being the equivalence class of $\Pi_\mathcal{V}$ with respect to $\equiv$.

Proof. Follows directly by Proposition 2.1. □

Corollary 2.2 (Periodicity of Potentials)**.** *Let $\mathcal{N} = (\mathcal{V}, \mathcal{A}, t_T)$ be a periodic event network and $\Pi_\mathcal{V}$ be a valid schedule with respect to $\mathcal{A}$. Then there exists a valid schedule $\Phi_\mathcal{V} \in [\Pi_\mathcal{V}]_\equiv$ with*

$$\forall n \in \mathcal{V} : \Phi_\mathcal{V}(n) \in [0, t_T - 1]$$

Proof. Follows directly by Proposition 2.1 and Corollary 2.1. □

With Corollary 2.2 the finite domain's choice of Example 2.8 has been clarified. In the remaining work it will always be tried to find the equivalence class' member presented in Corollary 2.2. Of course, it is not known beforehand, which equivalence classes represent valid schedules for its members. The sufficiency of only searching for the member presented in Corollary 2.2 is shown in the following theorem.

Theorem 2.1. *Let $\mathcal{N} = (\mathcal{V}, \mathcal{A}, t_T)$ be a periodic event network and*

$$\Omega := \{\Phi_\mathcal{V} \mid \Phi_\mathcal{V} \textit{ valid schedule with respect to } \mathcal{A}\}$$

the set of all valid schedules for $\mathcal{N}$. Then it is sufficient to search for a valid schedule $\Pi_\mathcal{V} \in \Omega$ with

$$\forall n \in \mathcal{V} : \Pi_\mathcal{V}(n) \in [0, t_T - 1]$$

with respect to not cutting the solution space Ω.

Proof.

$$\begin{aligned}\Omega &= \{\Phi_\mathcal{V} \mid \Phi_\mathcal{V} \text{ valid schedule with respect to } \mathcal{A}\}\\ &= \bigcup_{\Phi_\mathcal{V} \text{ valid}} [\Phi_\mathcal{V}]_\equiv \end{aligned} \tag{6}$$

With Corollary 2.1 and Corollary 2.2 it follows that

$$\exists \Pi_\mathcal{V} : \forall n \in \mathcal{V} : \Pi_\mathcal{V}(n) \in [0, t_T - 1] \tag{7}$$

in each $[\Phi_\mathcal{V}]_\equiv$ of (6). Hence,

$$\Omega = \bigcup_{\Pi_\mathcal{V} \text{ valid}} [\Pi_\mathcal{V}]_\equiv \quad \text{with } \Pi_\mathcal{V} \text{ of form (7)}$$

Thus, it is sufficient to look for valid schedules of form (7). □

In this section it has been shown, that if a valid schedule exists for a periodic event network $\mathcal{N} = (\mathcal{V}, \mathcal{A}, t_T)$, an event's potential π_n, with $n \in \mathcal{V}$, happens periodically often modulo t_T in time, since they are all in the same equivalence class.

2.2.2. Periodic Event Scheduling Problem

Like the SAT problem has been introduced in Definition 2.13 the periodic event scheduling problem (PESP) will be introduced as decision problem, which either says, that there does no valid schedule exist, or returns any valid schedule, if one exists.

Definition 2.25 (PESP). *Let $\mathcal{N} = (\mathcal{V}, \mathcal{A}, t_T)$ be a periodic event network. It shall be decided, whether*

1. *$\nexists$ schedule $\Pi_\mathcal{V} : \Pi_\mathcal{V}$ valid for $\mathcal{N}$ or*

2. *$\exists$ schedule $\Pi_\mathcal{V} : \Pi_\mathcal{V}$ valid for $\mathcal{N}$, return $\Pi_\mathcal{V}$.*

Solving such a PESP can be done in several ways. One way is to solve it with a PESP solver presented in the next section or with the method described in section 3. It has been shown in [Odi94], that PESP is $\mathcal{NP}$-complete.

2.2.3. A State-of-the-Art PESP Solver

As supposed and described in [Opi09] and [Nac96], it seems that the current best way to solve a PESP is using a constraint propagation technique. It basically propagates current probably valid assignments of events across the network. Once it finds out, that the valid domain of a potential is empty, there is no need to investigate further assignments for other events' potentials. Combining this with a decision tree method with respect to the events' potentials, it will be called in the remaining work as "the" PESP solver.

3. Polynomial Reduction from PESP to SAT

In this chapter, it will be shown, how a PESP instance can be encoded as a SAT instance. This means, given a periodic event network $\mathcal{N}$, we need to find a valid schedule or a proof, that there does not exist such a solution. These properties have to be encoded as SAT instance, meaning a propositional formula in conjunctive normal form, and then be proofed by a SAT solver.

If the SAT solver returns the SAT instance' unsatisfiability, we know, that there does not exist a valid schedule for the encoded periodic event network $\mathcal{N}$. On the other hand, getting a model for the propositional formula, it is ensured, that there exists a valid schedule for $\mathcal{N}$. The questions, how we can correctly encode such a periodic event network into a propositional formula in conjunctive normal form and how we can extract the schedule from the interpretation, computed by the SAT solver, will be answered in the following sections.

There will be two different kinds of encodings introduced for a PESP instance reduced into a SAT instance. Firstly, the direct encoding for variables of finite domains will be shown in Section 3.2 and the specific implementation for a PESP in Section 3.3. Secondly, the order encoding for variables with finite ordered domains will be defined in Section 3.4 and how it is used to encode a PESP as a SAT instance in Section 3.5.

In the end of this chapter, the two approaches will be compared, in order to get an indication and impression, how much faster an order encoded PESP can be solved, than a direct encoded one and how much smaller it is.

In the next chapter, these approaches will be tested in real world examples, compared to a traditional PESP solver.

3.1. Encoding Variables of Finite Domains

In order to encode problems like PESP, it is necessary to encode variables with finite domains.

The conditions to such an encoding are

1. the value of the variable must be within the given domain and
2. the value of the variable is arbitrary, but fixed.

This will be emphasized in the following examples.

Example 3.1. Let $x \in D$ be a variable and $D = \{a, b\}$. The domain D has two elements. Hence, it is sufficient to use a single propositional variable $p \in \mathcal{R}$, thus the propositional formula p, for x, in order to fulfill the conditions.

But it is important to note, that there must be a convention, like:

$$p^I = true \Rightarrow x = a$$
$$p^I = false \Rightarrow x = b$$

for an interpretation I.

Example 3.2. Let $x \in D$ be a variable and $D = \{a, b, c, d\}$. The domain D has four elements. It is obvious, that a single propositional variable is not sufficient to encode the variable x as propositional formula, because a propositional variable can only have two different values under an interpretation.

The previous example has shown, that we need to introduce more variables and connect them somehow in order to encode a variable x correctly as a propositional formula.

In the following sections the semantic and the syntax of a variable will be sometimes mixed, but finally clear by using them: A variable can be assigned to some special value of its domain, but it is as well element of a certain variable space. For example a propositional variable p is element of the variable space of the propositional variables $\mathcal{R}$, but it can be assigned two values of the set $\{true, false\}$ under a certain interpretation. The distinction in the following sections will always be clearly given by its embedded context.

Definition 3.1. *Let x be a variable. Then $dom(x)$ is the variable's domain and x is as well element in a variable space $\mathcal{X}$.*

3.2. Direct Encoding for Variables of Finite Domains

In this section, there will be introduced a special encoding for variables of finite domains, which does not assume, that the domain is ordered. It is called direct encoding. A more in depth view for this encoding can be read in [TTB11].

Definition 3.2 (Direct Encoding Function for Variables of Finite Domains)**.** *Let $x \in \mathcal{X}$ be a variable with $dom(x) = D$, $|D| < \infty$ and $\mathcal{X}$ be the variable space. Then the function*

$$encode\_direct : \mathcal{X} \to \mathcal{L}(\Sigma_{SAT})$$

is the direct encoding function for a variable *of the variable space $\mathcal{X}$ with*

$$encode\_direct : x \mapsto (\left(\bigvee_{a \in dom(x)} p_{x,a}\right) \wedge \left(\bigwedge_{a \in dom(x)} \bigwedge_{b \in dom(x), b \neq a} (\neg p_{x,a} \vee \neg p_{x,b})\right))$$

with $\forall a \in D : p_{x,a} \in \mathcal{R}$.

This encoding forces each except one propositional variable $p_{x,a}$ to be $false$ under a certain interpretation. This means with interpretation I and $a \in dom(x)$

$$p^I_{x,a} = true \Rightarrow x = a,$$

which is the previous announced convention for this encoding. This will be ensured by the following lemma.

Lemma 3.1. *Let $x \in \mathcal{X}$ be a variable with $dom(x) = D$, $|D| < \infty$ and $\mathcal{X}$ be the variable space. Then with*

$$S := \{p_{x,k} \mid \forall k \in D : p^I_{x,k} = true\}$$

being the set of propositional variables, that are $true$ under an interpretation I:

$$|S| = 1 \Leftrightarrow encode\_direct(x)^I = true,$$

Proof. "$\Rightarrow$":

$$|\{p_{x,k} \mid \forall k \in D : p_{x,k}^I = true\}| = 1 \tag{8}$$
$$\Rightarrow \exists! k \in D : I \models p_{x,k} \tag{9}$$

To show:

$$encode\_direct(x)^I \overset{Def\ 3.2}{=} ((\bigvee_{a \in D} p_{x,a}) \wedge (\bigwedge_{a \in D} \bigwedge_{b \in D, b \neq a} (\neg p_{x,a} \vee \neg p_{x,b})))^I = true$$

1. With (9) it follows directly
$$(\bigvee_{a \in D} p_{x,a})^I = true$$
2. Proof by contradiction: assume
$$(\bigwedge_{a \in D} \bigwedge_{b \in D, b \neq a} (\neg p_{x,a} \vee \neg p_{x,b})))^I = false$$
$$\Rightarrow \exists i, j \in D, i \neq j : (\neg p_{x,i} \vee \neg p_{x,j})^I = false$$
$$\Rightarrow \exists i, j \in D, i \neq j : p_{x,i}^I = true \wedge p_{x,j}^I = true$$

Contradiction to (8)

$$\Rightarrow (\bigwedge_{a \in D} \bigwedge_{b \in D, b \neq a} (\neg p_{x,a} \vee \neg p_{x,b})))^I = true$$
$$\overset{1.}{\Rightarrow} ((\bigvee_{a \in D} p_{x,a}) \wedge (\bigwedge_{a \in D} \bigwedge_{b \in D, b \neq a} (\neg p_{x,a} \vee \neg p_{x,b})))^I = true$$
$$\overset{Def\ 3.2}{\Rightarrow} encode\_direct(x)^I = true$$

"$\Leftarrow$":

$$encode\_direct(x)^I = true$$
$$\overset{Def\ 3.2}{\Rightarrow} ((\bigvee_{a \in D} p_{x,a}) \wedge (\bigwedge_{a \in D} \bigwedge_{b \in D, b \neq a} (\neg p_{x,a} \vee \neg p_{x,b})))^I = true$$
$$\Rightarrow (\bigvee_{a \in D} p_{x,a})^I = true \Rightarrow |S| \geq 1, \tag{10}$$
$$(\bigwedge_{a \in D} \bigwedge_{b \in D, b \neq a} (\neg p_{x,a} \vee \neg p_{x,b}))^I = true \tag{11}$$

To show: $S \leq 1$.
Proof by contradiction: assume $|S| \geq 2$

$$\Rightarrow \exists i, j \in D, i \neq j : p_{x,i}^I = true \wedge p_{x,j}^I = true$$
$$\Rightarrow \exists i, j \in D, i \neq j : (\neg p_{x,i} \vee \neg p_{x,j})^I = false$$
$$\Rightarrow (\bigwedge_{a \in D} \bigwedge_{b \in D, b \neq a} (\neg p_{x,a} \vee \neg p_{x,b}))^I = false$$

Contradiction to (11)

$$\begin{aligned} &\Rightarrow |S| \leq 1 \\ &\overset{(10)}{\Rightarrow} |S| = 1 \end{aligned}$$

□

The large disjunction, which represents the first part of the formula, ensures, that at least one value of the domain will be assigned to x. Consequently, it fulfills the first condition of 3.1. The second part ensures, that the second condition holds, which will be emphasized in the following example.

Because the propositional formula, that will be yielded by $encode\_direct$, is a conjunction of disjunctions of literals, it is already in conjunctive normal form and can be directly used for SAT solvers, since most SAT solvers need the formula in CNF.

Example 3.3. Let x be a variable with $dom(x) = \{1, 2, 3\}$. Then

$$\begin{aligned} F := encode\_direct(x) \quad &= ((p_{x,1} \vee p_{x,2} \vee p_{x,3}) \wedge (\neg p_{x,1} \vee \neg p_{x,2}) \\ &\quad \wedge (\neg p_{x,1} \vee \neg p_{x,3}) \wedge (\neg p_{x,2} \vee \neg p_{x,3})) \\ &= \langle [p_{x,1}, p_{x,2}, p_{x,3}], [\neg p_{x,1}, \neg p_{x,2}], \\ &\quad [\neg p_{x,1}, \neg p_{x,3}], [\neg p_{x,2}, \neg p_{x,3}] \rangle. \end{aligned}$$

Since we are opting for interpretations, which are models for F, let us see some examples:

$$I = \{p_{x,1}, p_{x,2}\} \Rightarrow F^I = false,$$

because $[\neg p_{x,1}, \neg p_{x,2}]^I = false$.

$$J = \{p_{x,1}\} \Rightarrow F^J = true \Rightarrow x = 1,$$

because in each clause contains an propositional variable, which is not in J and at least one propositional variable is $true$ under J, which validates the first clause.

Now, we are able to encode variables with finite domains and this will be used in the following section.

3.3. Reducing PESP to SAT Using Direct Encoding

Given a periodic event network $\mathcal{N} = (\mathcal{V}, \mathcal{A}, t_T)$ and Theorem 2.1 it is sufficient to search for potentials within the interval $[0, t_T - 1]$. This leads to a finite domain for each node's potential, which implies, that it can be encoded with the function $encode\_direct$ introduced in Definition 3.2.

After the encoding for each potential has been introduced, it will be followed by the encoding for the constraints. This can be concluded to encode a PESP instance.

3.3.1. Encoding of Event Variables (Potentials)

As stated above, we know, that we can encode each potential π_n with $n \in \mathcal{V}$ of a schedule of the form (7) for $\mathcal{N}$. The variable space $\mathcal{X}$ of Definition 3.2 is

$$\mathcal{X} := \{\pi_n \mid \forall n \in \mathcal{V}\}.$$

Hence, we can apply for each $n \in \mathcal{V}$ the encoding function $encode\_direct$, because we know, that $dom(\pi_n) \in [0, t_T - 1]$.

This yields to the encoding of the potentials

$$\Omega^{\mathcal{V}}_{direct} := \bigwedge_{n \in \mathcal{V}} encode\_direct(\pi_n), \tag{12}$$

which is a conjunction of each node's potential, since every potential must obey the conditions of 3.1.

Because $encode\_direct$ returns a propositional formula in CNF and $\Omega^{\mathcal{V}}_{direct}$ is a conjunction of these, it remains in CNF.

In order to extract the schedule $\Pi_{\mathcal{V}}$ from an interpretation I, which satisfies the given propositional formula, we do the following

$$\forall n \in \mathcal{V} : \Pi_{\mathcal{V}}(n) = i, \text{ if and only if } p^I_{\pi_n, i} = true.$$

Furthermore, with Lemma 3.1 we know, that there exists exactly one $i \in dom(\pi_n)$ for all $n \in \mathcal{V}$. Thus, the schedule is well-defined and bijective to I.

3.3.2. Encoding Constraints of Periodic Event Networks

There are several ways of encoding constraints of Definition 2.20 and Definition 2.21. The presented way in this work is to exclude the not feasible region of each constraint of the periodic event network.

The sufficiency and possibility in order to get valid schedules can be easily shown: since the search space $X := \{\Pi_{\mathcal{V}} \mid \Pi_{\mathcal{V}} \text{ of form (7)}\}$ and the solution space $S := \{\Pi_{\mathcal{V}} \in X \mid \Pi_{\mathcal{V}} \text{ is valid}\}$ are finite, because $dom(\pi_n) \in [0, t_T - 1]$, $\forall n \in \mathcal{V}$, the not feasible region $X \setminus S$ is finite. Thus, the encoding of a constraint $a \in \mathcal{A}$ can be defined in a finite manner.

3.3.3. Direct Encoding of Constraints

As proposed in 3.3.2, we will encode each constraint by excluding the not feasible region with the following function.

Definition 3.3 (Direct Encoding Function for Constraints)**.** *Let $\mathcal{A}$ be a set of constraints. Then the function*

$$encode\_direct\_con : \mathcal{A} \rightarrow \mathcal{L}(\Sigma_{\mathsf{SAT}})$$

is the direct encoding function of a constraint $a = ((n_1, n_2), [l_a, u_a]_{t_T}) \in \mathcal{A}$ *with*

$$encode\_direct\_con : a \mapsto \bigwedge_{(i,j) \in P_a} (\neg p_{n_1,i} \vee \neg p_{n_2,j})$$

with $\{p_{n,i} \mid i \in [0, t_T - 1], n \in \{n_1, n_2\}\}$ *being the set direct encoding propositional variables of the potentials* π_{n_1}*,* π_{n_2} *and* $P_a := \{(\pi_{n_1}, \pi_{n_2}) \mid a$ *does not hold for* $\pi_{n_1}, \pi_{n_2}\}$ *be the set of pairs, which are part of the not feasible region of* a*.*

Now, we are able to encode each constraint $a \in \mathcal{A}$ and can add it to the resulting propositional formula. The following example demonstrates, how such an encoding works

Example 3.4. Let A, B be events and $a = ((A, B), [3, 5]_{10})$ be a time consuming constraint of Example 2.8.

As seen in Figure 2.3 we need to exclude each pair (π_A, π_B), such that a is not valid under these potentials, which means not part of the feasible region.

E.g., the example's pair $(5, 5)$ must be excluded. Which means, by having the variables π_A and π_B direct encoded with the propositional variables $p_{A,i}$ and $p_{B,i}$, respectively, $i \in [0, 9]$, the assignments $\pi_A = 5$ and $\pi_B = 5$ must not happen, in order to receive a valid schedule. Hence,

$$\begin{aligned} & \nexists (\pi_A, \pi_B) : \pi_A = 5 \text{ and } \pi_B = 5 \\ \Leftrightarrow \quad & \neg(\pi_A = 5 \wedge \pi_B = 5) \text{ must hold} \\ \Leftrightarrow \quad & \neg(p_{A,5} \wedge p_{B,5})^I = true \\ \overset{Example\ 2.4}{\Leftrightarrow} \quad & (\neg p_{A,5} \vee \neg p_{B,5})^I = true \\ \overset{Defition\ 2.5}{\Leftrightarrow} \quad & [\neg p_{A,5}, \neg p_{B,5}]^I = true, \end{aligned}$$

with I being the interpretation, such that π_A, π_B can be extracted from I, because they are direct encoded.

Since the last expression is in conjunctive normal form, we can add it to the resulting formula, which shall encode the constraint a.

Since $encode\_direct\_con$ yields a large conjunction of a disjunction of two literals, it maps to a propositional formula in conjunctive normal form.

Applying $encode\_direct\_con$ to every constraint $a \in \mathcal{A}$ of a periodic event network $\mathcal{N} = (\mathcal{V}, \mathcal{A}, t_T)$ and requiring, that each constraint must hold, we can formulate this as

$$\Psi^{\mathcal{A}}_{direct} := \bigwedge_{a \in \mathcal{A}} encode\_direct\_con(a). \tag{13}$$

However, it is necessary, that a node $n \in \mathcal{V}$ for two constraints $a_1, a_2 \in \mathcal{A}$ is encoded with the same propositional variables $p_{n,i}$ in order to get valid schedules for $\mathcal{N}$.

3.3.4. Encoding of the PESP

Having $encode\_direct$ and $encode\_direct\_con$, respectively $\Omega^{\mathcal{V}}_{direct}$ and $\Psi^{\mathcal{A}}_{direct}$, we are able to encode a periodic event network respectively a PESP. This means, that we need to encode the potentials as well as the constraints in a single propositional formula, which is done in the following definition.

Definition 3.4 (Direct Encoding of PESP)**.** *The function*

$$encode\_direct\_pesp : 2^{\mathcal{V}^+} \times 2^{\mathcal{A}^+_{t_T}} \times 2^{\mathbb{N}} \to \mathcal{L}(\Sigma_{SAT})$$

is the direct encoding function of a PESP *with respect to an periodic event network $\mathcal{N} = (\mathcal{V}, \mathcal{A}, t_T)$ with*

$$encode\_direct\_pesp : (\mathcal{V}, \mathcal{A}, t_T) \mapsto (\Omega^{\mathcal{V}}_{direct} \wedge \Psi^{\mathcal{A}}_{direct}),$$

$2^{\mathcal{V}^+}$ and $2^{\mathcal{A}^+_{t_T}}$ being the power sets of all possible events and edges, respectively (see Definition 2.17). $\Omega^{\mathcal{V}}_{direct}$ and $\Psi^{\mathcal{A}}_{direct}$ are of the form of (12) and (13), respectively.

$\Omega^{\mathcal{V}}_{direct}$ is said to be in CNF. Likewise, this can be said about $\Psi^{\mathcal{A}}_{direct}$, because the encoding function $direct\_encode\_con$ yields a propositional formula in CNF and $\Psi^{\mathcal{A}}_{direct}$ is a conjunction of these. As the previous defined encoding functions, $encode\_direct\_pesp$ is a conjunction of two propositional formulas, which are in CNF. Hence, the mapping $encode\_direct\_pesp$ yields a propositional formula in CNF, which can be handled by most modern SAT solvers. With $encode\_direct\_pesp$ we are now able to encode a whole PESP with respect to its periodic event network. The following example shows a brief overview of executing this function.

Example 3.5. Let $\mathcal{N} = (\mathcal{V}, \mathcal{A}, 10)$ be the periodic event network of Example 2.7, which means

$$\begin{aligned}\mathcal{V} &= \{A, B, C\}\\ \mathcal{A} = \mathcal{C} &= \{a_1, a_2, a_3\},\end{aligned}$$

with

$$\begin{aligned}a_1 &:= ((A, B), [3, 5]_{10})\\ a_2 &:= ((B, C), [2, 2]_{10})\\ a_3 &:= ((C, A), [2, 4]_{10}).\end{aligned}$$

In order to encode this periodic event network, respectively the PESP, we need to apply the function $encode\_direct\_pesp$. This yields

$$\begin{aligned} encode\_direct\_pesp(\mathcal{N}) = \;& (\Omega^{\mathcal{V}}_{direct} \wedge \Psi^{\mathcal{A}}_{direct}) \\ = \;& \left(\bigwedge_{n \in \mathcal{V}} encode\_direct(\pi_n)\right) \\ & \wedge \left(\bigwedge_{a \in \mathcal{A}} encode\_direct\_con(a)\right) \\ = \;& (encode\_direct(\pi_A) \\ & \wedge encode\_direct(\pi_B) \wedge encode\_direct(\pi_C) \\ & \wedge encode\_direct\_con(a_1) \wedge encode\_direct\_con(a_2) \\ & \wedge encode\_direct\_con(a_3)) \end{aligned}$$

Applying function $encode\_direct$, we get for event's potential π_A

$$\begin{aligned} encode\_direct(\pi_A) = \langle & [p_{\pi_A,0}, p_{\pi_A,1}, \ldots, p_{\pi_A,9}], \\ & [\neg p_{\pi_A,0}, \neg p_{\pi_A,1}], [\neg p_{\pi_A,0}, \neg p_{\pi_A,2}], \ldots, [\neg p_{\pi_A,8}, \neg p_{\pi_A,9}]\rangle. \end{aligned}$$

Likewise, we use this function for π_B and π_C.

In order to evaluate the constraints a_i $(i \in \{1,2,3\})$, we first need to calculate the sets P_{a_i} $(i \in \{1,2,3\})$ of Definition 3.3. This will be demonstrated for the time consuming constraint a_1:

$$\begin{aligned} P_{a_1} &= \{(\pi_A, \pi_B) \mid \forall \pi_A, \pi_B \in \{0,\ldots,9\} : a_1 \text{ does not hold for } \pi_A, \pi_B\} \\ &= \{(0,0),(0,1),(0,2),\ldots\} \end{aligned}$$

Afterwards, applying $encode\_direct\_con$ yields

$$\begin{aligned} encode\_direct\_con(a_1) &= \bigwedge_{(i,j) \in P_{a_1}} (\neg p_{\pi_A,i} \vee \neg p_{\pi_B,j}) \\ &= \langle [\neg p_{\pi_A,0}, \neg p_{\pi_B,0}], [\neg p_{\pi_A,0}, \neg p_{\pi_B,1}], [\neg p_{\pi_A,0}, \neg p_{\pi_B,2}], \ldots\rangle. \end{aligned}$$

The whole encoding can be found as SAT instance, with the respective PESP instance, on CD as of Section A, which can be used for most state of the art SAT solvers.

Furthermore, we can show soundness and completeness of the direct encoding method in the following theorem, in order to ensure the correct possibility to use the given approach.

Theorem 3.1 (Soundness and Completeness Direct Encoding). *Let $\mathcal{N} = (\mathcal{V}, \mathcal{A}, t_T)$ be a periodic event network and*

$$F := encode\_direct\_pesp(\mathcal{N}) \in \mathcal{L}(\Sigma_{SAT})$$

be the direct encoded propositional formula of $\mathcal{N}$. Then

$$\exists I : I \models F \Leftrightarrow \exists \Pi_{\mathcal{V}} : \Pi_{\mathcal{V}} \text{ is valid}$$

with I being an interpretation and $\Pi_{\mathcal{V}}$ a schedule of $\mathcal{N}$.

Proof. "⇒":

$$\exists I : I \models F$$

We extract $\Pi_{\mathcal{V}}$ from I as stated in 3.3.1.

Proof by contradiction: assume an arbitrary, but fixed $a = ((n_1, n_2), [l_a, u_a]_{t_T}) \in \mathcal{A}$, without loss of generality a time consuming constraint, such that a does not hold under $\Pi_{\mathcal{V}}$ with $i := \Pi_{\mathcal{V}}(n_1)$, $j := \Pi_{\mathcal{V}}(n_2)$.

$$\Rightarrow j - i \notin [l_a, u_a]_{t_T} \tag{14}$$

$$p^I_{\pi_{n_1},i} = true \wedge p^I_{\pi_{n_2},j} = true$$

$$\Rightarrow [\neg p_{\pi_{n_1},i}, \neg p_{\pi_{n_2},j}] \notin F \tag{15}$$

but with (14) and Definition 3.3 we know

$$(i, j) \in P_a \overset{Def\ 3.3}{\Longrightarrow} [\neg p_{\pi_{n_1},i}, \neg p_{\pi_{n_2},j}] \in F$$

contradiction to (15)

$$\Rightarrow a \text{ must hold under } \Pi_{\mathcal{V}}.$$

Since a is arbitrary, it holds for all $a \in \mathcal{A}$.

$$\overset{Def\ 2.23}{\Rightarrow} \exists \Pi_{\mathcal{V}} : \Pi_{\mathcal{V}} \text{ is valid}$$

"⇐":

$$\begin{aligned} & \exists \Pi_{\mathcal{V}} : \Pi_{\mathcal{V}} \text{ is valid} \\ \overset{Def\ 2.23}{\Rightarrow} & \forall a \in \mathcal{A} : a \text{ holds under } \Pi_{\mathcal{V}} \\ \overset{Def\ 2.22}{\Rightarrow} & \forall a = ((n_1, n_2), [l_a, u_a]_{t_T}) \in \mathcal{A} : a \text{ holds for } \pi_{n_1}, \pi_{n_2} \\ \overset{Def\ 3.3}{\Rightarrow} & \forall a = ((n_1, n_2), [l_a, u_a]_{t_T}) \in \mathcal{A}, i := \Pi_{\mathcal{V}}(n_1), j := \Pi_{\mathcal{V}}(n_2) : (i, j) \notin P_a. \end{aligned} \tag{16}$$

Construct interpretation I, such that

$$\forall n \in \mathcal{V}\ \forall k \in \{0, \ldots, t_T - 1\} : p^I_{n,k} = \begin{cases} true & \text{if } k = \Pi_{\mathcal{V}}(n) \\ false & \text{if } k \neq \Pi_{\mathcal{V}}(n) \end{cases}$$

$$\Rightarrow \forall n \in \mathcal{V} : |\{p_{n,k} \mid \forall k \in \{0, \ldots, t_T - 1\} : p^I_{n,k} = true\}| = 1$$

$$\overset{Lem\ 3.1}{\Rightarrow} \forall n \in \mathcal{V} : encode\_direct(\pi_n)^I = true$$

$$\overset{(12)}{\Rightarrow} I \models \Omega^{\mathcal{V}}_{direct}. \tag{17}$$

Let $a = ((n_1, n_2), [l_a, u_a]_{t_T}) \in \mathcal{A}$ be an arbitrary, but fixed constraint.

To show: $encode\_direct\_con(a)^I = true$.

Proof by contradiction: assume $encode\_direct\_con(a)^I = false$

$$\overset{Def\ 3.3}{\Rightarrow} \exists (k, l) \in P_a : [\neg p_{\pi_{n_1},k}, \neg p_{\pi_{n_2},l}]^I = false$$

contradiction to (16), because $\forall(k,l) \in P_a : p^I_{\pi_{n_1},k} = false \vee p^I_{\pi_{n_2},l} = false$

$$\Rightarrow encode\_direct\_con(a)^I = true$$

Because a is arbitrary, it holds for all $a \in \mathcal{A}$.

$$\begin{aligned}
&\Rightarrow \forall a \in \mathcal{A} : encode\_direct\_con(a)^I = true \\
&\overset{(13)}{\Rightarrow} I \models \Psi^{\mathcal{A}}_{direct} \\
&\overset{(17)}{\Rightarrow} (\Omega^{\mathcal{V}}_{direct} \wedge \Psi^{\mathcal{A}}_{direct})^I = true \\
&\overset{Def\ 3.3}{\Rightarrow} F^I = true \\
&\Rightarrow \exists I : I \models F
\end{aligned}$$

□

We can directly conclude the following corollary.

Corollary 3.1 (Soundness and Completeness Direct Encoding). *Let $\mathcal{N} = (\mathcal{V}, \mathcal{A}, t_T)$ be a periodic event network and*

$$F := encode\_direct\_pesp(\mathcal{N}) \in \mathcal{L}(\Sigma_{SAT})$$

be the direct encoded propositional formula of $\mathcal{N}$. Then

$$\nexists I : I \models F \Leftrightarrow \nexists \Pi_{\mathcal{V}} : \Pi_{\mathcal{V}} \textit{ is valid}$$

with I being an interpretation and $\Pi_{\mathcal{V}}$ a schedule of $\mathcal{N}$.

Proof. "⇒":

$$\nexists I : I \models F$$

Proof by contradiction: assume

$$\begin{aligned}
&\exists \Pi_{\mathcal{V}} : \Pi_{\mathcal{V}} \text{ is valid for } \mathcal{N} \\
&\overset{Thrm\ 3.1}{\Rightarrow} \exists I : I \models F, \qquad (18)
\end{aligned}$$

but

$$\nexists I : I \models F \Rightarrow \forall I : I \not\models F.$$

contradiction to (18).

$$\Rightarrow \nexists \Pi_{\mathcal{V}} : \Pi_{\mathcal{V}} \text{ is valid}$$

"⇐": analogous □

This method will be applied in Section 4 and compared to the order encoding approach, which will be presented in the following section.

3.4. Order Encoding for Variables of Finite Ordered Domains

In contrast to 3.2, this section assumes an ordered finite domain. The best example is a proper subset of the natural numbers $\mathbb{N}$. These are always ordered with respect to $<$. In the following, it will be discussed how this property can be efficiently encoded into a propositional formula. Enhancing knowledge about this topic can be found in [TTB11].

Because in this work we only regard variables having the domain as a subset, especially an interval, of the natural numbers, we know, how to apply its order relation "$<$". In general, it could be assumed, that it is any set, which can be related to an specific order relation.

Example 3.6. Let x be a variable with $dom(x) = \{1, 2, 3, 4, 5\} = [1, 5] \subset \mathbb{N}$. Then we know

$$x \geq 1 \Rightarrow x \not\leq 0 \Rightarrow \neg(x \leq 0) \tag{19}$$
$$x \leq 5 \tag{20}$$
$$\forall i \in \{1, \ldots, 5\} : (x \leq i - 1) \rightarrow (x \leq i)$$
$$\Leftrightarrow \forall i \in \{1, \ldots, 5\} : \neg(x \leq i - 1) \vee (x \leq i). \tag{21}$$

With facts (19) and (20), we can conclude

$$(21) \overset{(19),(20)}{\Rightarrow} \forall i \in \{2, \ldots, 4\} : \neg(x \leq i - 1) \vee (x \leq i).$$

The previous example shows the intension of the order encoding and results in the following definition.

Definition 3.5 (Order Encoding Function for Variables of Finite Domains). *Let $x \in \mathcal{X}$ be a variable with $dom(x) = D = [l, u] \subset \mathbb{N}$ and $\mathcal{X}$ be the variable space. Then the function*

$$encode\_ordered : \mathcal{X} \rightarrow \mathcal{L}(\Sigma_{SAT})$$

is the order encoding function for a variable *of the variable space $\mathcal{X}$ with*

$$encode\_ordered : x \mapsto \bigwedge_{i \in [l+1, u-1]} (\neg q_{x,i-1}, q_{x,i})$$

with $\forall i \in [l, u - 1] : q_{x,i} \in \mathcal{R}$.

Since $encode\_ordered$ maps to a propositional formula, which is a conjunction of disjunctions of literals, it is in conjunctive normal form. Unlike in the direct encoding approach, $q_{x,i}$ has the meaning

$$q_{x,i} \Leftrightarrow x \leq i.$$

This results in two different cases for a given interpretation I:

$$q^I_{x,i} = true \Leftrightarrow x \leq i, \tag{22}$$
$$q^I_{x,i} = false \Leftrightarrow x \not\leq i. \tag{23}$$

Extracting the value $x = i$ of the given interpretation is not that obvious, as in the direct encoding method. Before giving the definition for extracting the value for x, we need the following lemma to show the interpretation's structure.

Lemma 3.2. *Let x be a variable with $dom(x) = D = [l, u] \subset \mathbb{N}$ and I an interpretation. Then*

$$\begin{aligned} I \models encode\_ordered(x) \Leftrightarrow \exists! k \in [l, u] : \forall i \in [l, k-1] : I \not\models q_{x,i}, \\ \forall j \in [k, u-1] : I \models q_{x,j} \end{aligned}$$

Proof (sketch). "$\Rightarrow$": Let $I \models encode\_ordered(x)$. To show:

1. $\exists k \in [l, u] : \forall i \in [l, k-1] : I \not\models q_{x,i}, \forall j \in [k, u-1] : I \models q_{x,j} \quad \wedge$
2. $\nexists h \in [l, u], h \neq k : \forall i \in [l, h-1] : I \not\models q_{x,i}, \forall j \in [h, u-1] : I \models q_{x,j}$

1. is simply shown with mathematical induction.
2. is simply shown without loss of generality $h > k \Rightarrow q^I_{x,k} = false$, which is a contradiction to 1.
"$\Leftarrow$": can be simply shown with mathematical induction as in "$\Rightarrow$". $\square$

The previous lemma shows the interpretation's structure. It can be concluded for an interpretation I, which satisfies the propositional formula $encode\_ordered(x)$:

$$\begin{aligned} & \exists k \in [l, u-1] : q^I_{x,k-1} = false, q^I_{x,k} = true \\ \overset{(22),(23)}{\Rightarrow} \; & \exists k \in [l, u] : x \not\leq k-1, x \leq k \\ \overset{dom(x)\subset\mathbb{N}}{\Rightarrow} \; & \exists k \in [l, u] : x \geq k, x \leq k \\ \Rightarrow \; & \exists k \in [l, u] : x = k \end{aligned}$$

This will be summarized in the following definition.

Definition 3.6 (Extracting Value of Intepretation of Order Encoded Variable)**.** *Let x be a variable with $dom(x) = D = [l, u] \subset \mathbb{N}$ and I an interpretation, such that $I \models encode\_ordered(x)$. Then there exists a $k \in [l, u]$ with*

$$x = \xi_x(I) = \begin{cases} k & k \in [l, u-1] : q^I_{x,k-1} = false \wedge q^I_{x,k} = true \\ u & k = u : q^I_{x,u-1} = false \end{cases}$$

We are now able to extract the value of x of an satisfying interpretation I, with respect to $encode\_ordered(x)$. The following example demonstrates this behaviour.

Example 3.7. Let x be the variable with $dom(x) = [1, 5] \subset \mathbb{N}$ of Example 3.6. Applying $encode\_ordered$ yields

$$\begin{aligned} encode\_ordered(x) &= (\neg q_{x,1} \vee q_{x,2}) \wedge (\neg q_{x,2} \vee q_{x,3}) \wedge (\neg q_{x,3} \vee q_{x,4}) \\ &= \langle [\neg q_{x,1}, q_{x,2}], [\neg q_{x,2}, q_{x,3}], [\neg q_{x,3}, q_{x,4}] \rangle \end{aligned}$$

Let I be an interpretation with

$$\begin{aligned} q^I_{x,1} &= false \\ q^I_{x,2} &= false \\ q^I_{x,3} &= false \\ q^I_{x,4} &= true. \end{aligned}$$

Then we can extract the value for x by applying Definition 3.6, which yields

$$x = 4,$$

because $x \not\leq 3$ ($q^I_{x,3} = false$) and $x \leq 4$ ($q^I_{x,4} = true$).

3.5. Reducing PESP to SAT Using Order Encoding

The procedure of order encoding a PESP instance into a SAT instance will be equivalent to the direct encoding approach. First of all, it will be shown, how the set of the potentials can be encoded. Followed by that, both constraint type encoding functions will be defined. In the end, it will be summarized to the encoding of the whole PESP with respect to a certain periodic event network $\mathcal{N}$.

3.5.1. Encoding of Event Variables (Potentials)

As proposed and applied in Section 3.3.1, the node's potential domain will be within $[0, t_T - 1]$ in order to receive schedules of the form (7). Thus, $encode\_ordered$ will be appled to each node's potential π_n of all nodes in the periodic event network.

Given a periodic event network $\mathcal{N} = (\mathcal{V}, \mathcal{A}, t_T)$

$$\Omega^{\mathcal{V}}_{ordered} := \bigwedge_{n \in \mathcal{V}} encode\_ordered(\pi_n), \tag{24}$$

is the propositional formula in conjunctive normal form, which encodes all node's potentials of $\mathcal{V}$.

Extracting a schedule $\Pi_{\mathcal{V}}$ from an interpretation I, which satisfies $\Omega^{\mathcal{V}}_{ordered}$ is given on a per-element basis by

$$\forall n \in \mathcal{V} : \Pi_{\mathcal{V}}(n) = \xi_{\pi_n}(I), \tag{25}$$

with ξ_{π_n} being the mapping given in Definition 3.6. The bijective and well-defined behavior as proposed in Section 3.3.1 is given in this approach as well with Lemma 3.2.

3.5.2. Encoding of Time Consuming Constraints

This section will thoroughly deal with the order encoding of time consuming constraints by a given periodic event network $\mathcal{N}$. In the direct encoding approach in Section 3.3.3 the goal is effectively achieved by excluding all not feasible regions for each constraint. Likewise, this method will be applied in this context as well.

With two potentials being order encoded, we are able to not just exclude not feasible pairs, as used in the direct encoding method, but exclude larger regions, which must be rectangles. The base idea is shown in the following example.

Example 3.8. Let A, B be two events and $a = ((A, B), [3, 5]_{10})$ be a time consuming constraint of Example 2.8 with π_A and π_B the potential of A and B, respectively.

An not feasible region would be for example

$$r = ([4, 7] \times [3, 6])$$

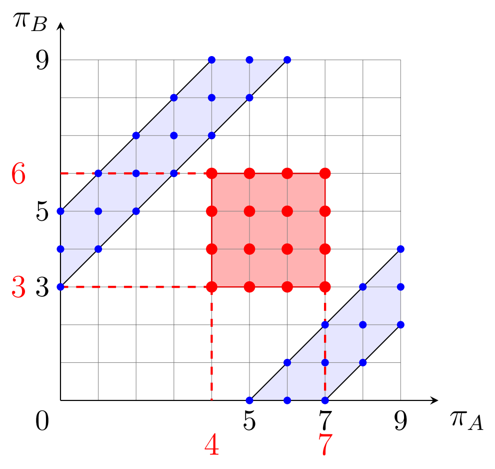


Figure 3.1: Not feasible rectangle r of the time consuming constraint for Example 3.8

with $(\pi_A, \pi_B) \notin r$. The to be excluded rectangle r is visualized in Figure 3.1.

Thus, all pairs, which are not feasible is the set

$$\{(i,j) \mid i \in [4,7], j \in [3,6]\} = r.$$

Then, we know, that for each pair $(\pi_A, \pi_B) \notin r$, such that a holds for π_A, π_B. With $q_{A,i}$, $i \in [0,8]$ and $q_{B,j}$, $j \in [0,8]$ being the propositional variables of Definition 3.5, it follows

$$\begin{aligned}
&\nexists(\pi_A,\pi_B) : \pi_A \leq 7, \pi_A \geq 4, \pi_B \leq 6, \pi_B \geq 3\\
\Leftrightarrow\ &\neg((\pi_A \leq 7) \wedge (\pi_A \geq 4) \wedge (\pi_B \leq 6) \wedge (\pi_B \geq 3))\\
\Leftrightarrow\ &\neg((\pi_A \leq 7) \wedge \neg(\pi_A \leq 3) \wedge (\pi_B \leq 6) \wedge \neg(\pi_B \leq 2))\\
\Leftrightarrow\ &\neg(q_{\pi_A,7} \wedge \neg q_{\pi_A,3} \wedge q_{\pi_B,6} \wedge \neg q_{\pi_B,2})\\
\Leftrightarrow\ &(\neg q_{\pi_A,7} \vee q_{\pi_A,3} \vee \neg q_{\pi_B,6} \vee q_{\pi_B,2})\\
&= [\neg q_{\pi_A,7}, q_{\pi_A,3}, \neg q_{\pi_B,6}, q_{\pi_B,2}] =: F
\end{aligned}$$

Since F is a clause, we can directly add this to the resulting formula.

The previous example shows, how we can exclude a single rectangle. Before giving the function, which maps each constraint to a propositional formula in conjunctive normal form, we need some preliminary definitions, in order to give the concrete mapping itself.

First of all, we need some helper functions. These helper functions will be concluded in a function, which calculates as few rectangles as possible to cover the whole not feasible region of a constraint. This results in the lowest possible number of clauses, which should help a SAT solver to find a solution faster having less redundancy.

Reconsidering Figure 3.1 displays the two feasible regions. Each of which has a lower and an upper bound. In order to describe the not feasible region in between, we need the next lower bound, which represents the upper feasible region and the previous lower bound, which represents the lower feasible region with respect to π_B. With respect to

Figure 3.1 and the displayed not feasible region, we would have an upper bound $u = -5$ and a lower bound $l = 3$. It can be directly concluded, that $u < l$.

Definition 3.7. *Let $u, l \in \mathbb{Z}$ be two integers with $u < l$. Then*

$$\begin{aligned} \delta \colon \mathbb{Z} \times \mathbb{Z} &\to \mathbb{Z} \\ (l, u) &\mapsto l - u - 1 \end{aligned}$$

describes the interior distance *between u and l.*

Definition 3.8. *Let $u, l \in \mathbb{Z}$ two integers with $u < l$. Then*

$$\begin{aligned} \delta y \colon \mathbb{Z} \times \mathbb{Z} &\to \mathbb{Z} \\ (l, u) &\mapsto \left\lfloor \frac{\delta(l, u)}{2} \right\rfloor \end{aligned}$$

is the height *for an rectangle between u and l, with $\lfloor \cdot \rfloor$ being the round down function.*

Definition 3.9. *Let $u, l \in \mathbb{Z}$ be two integers with $u < l$. Then*

$$\begin{aligned} \delta x \colon \mathbb{Z} \times \mathbb{Z} &\to \mathbb{Z} \\ (l, u) &\mapsto \left\lceil \frac{\delta(l, u)}{2} \right\rceil - 1 \end{aligned}$$

is the width *for an rectangle between u and l, with $\lceil \cdot \rceil$ being the round up function.*

With these definitions, we are able to determine a rectangle between u and l, such that it has maximum area having minimum perimeter. These functions are demonstrated in the following example.

Example 3.9. Let A, B be two events and $a = ((A, B), [3, 5]_{10})$ be a time consuming constraint of Example 2.8 with π_A and π_B the potential of A and B, respectively.
For example, we choose $l = 3$, $u = -5$. Then

$$\begin{aligned} \delta(3, -5) &= 3 - (-5) - 1 = 7 \\ \delta y(3, -5) &= \left\lfloor \frac{\delta(3, -5)}{2} \right\rfloor = \left\lfloor \frac{7}{2} \right\rfloor = 3 \\ \delta x(3, -5) &= \left\lceil \frac{\delta(3, -5)}{2} \right\rceil - 1 = \left\lceil \frac{7}{2} \right\rceil - 1 = 3 \end{aligned}$$

The evaluated values are displayed in Figure 3.2.

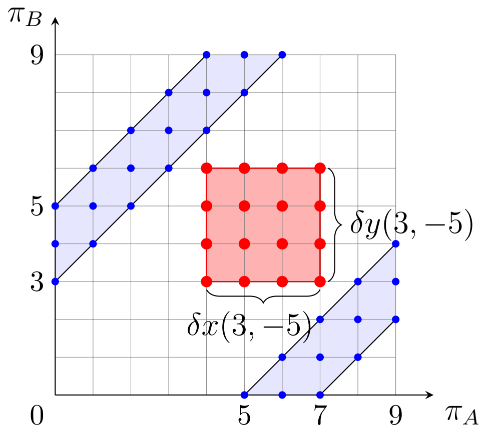


Figure 3.2: Evaluated function values $\delta y(3,-5)$, $\delta x(3,-5)$ of Example 3.9

The previous example shows the meaning of the helper functions δx and δy. Basically each rectangle will have a width of $\delta x(l,u)$ and a height of $\delta y(l,u)$.

In order to cover each not feasible pair in the area between u and l, we need approximately t_T rectangles. The rectangle's shape does not influence the count t_T of the needed rectangles. The δx, δy approach seems to be the best, since most conflicts can be faster detected by a SAT solver, because we have each not feasible pair more often encoded, without having more clauses in the resulting propositional formula. The worse runtime behaviour of other shapes will not be verified in this work.

Definition 3.10. *Let $u, l \in \mathbb{Z}$ two integers with $u < l$ and $t_T \in \mathbb{N}$. Then*

$$R_{l,u,t_T} := \{(a,b) \in [0, t_T - 1] \times [0, t_T - 1] \mid b - a \in [u+1, l-1]\}$$

is the set of not feasible pairs *between u and l.*

Definition 3.11. *Let $u, l \in \mathbb{Z}$ be two integers with $u < l$ and $t_T \in \mathbb{N}$. Then*

$$\begin{aligned} \varphi_{t_T} : \mathbb{Z} \times \mathbb{Z} &\to 2^{(2^{\mathbb{Z}} \times 2^{\mathbb{Z}})} \\ (l,u) &\mapsto \{([x_1, x_1 + \delta x(l,u)] \times [x_2, x_2 + \delta y(l,u)]) \\ &\qquad \mid \forall x_2 \in \{-\delta y(l,u), \ldots, t_T - 1\} : \\ &\qquad\quad x_1 + \delta x(l,u) \geq 0 \land x_1 \leq t_T - 1 : \\ &\qquad\quad x_1 = x_2 - u - 1 - \delta x(l,u)\} \end{aligned}$$

maps (l,u) to the set of rectangles *between u and l.*

In order to get all not feasible pairs, we need several regions between different u and l as exposed in the following example.

Example 3.10. Let A, B be two events and $a = ((A,B), [3,5]_{10})$ be a time consuming constraint of Example 2.8 with π_A and π_B the potential of A and B, respectively.

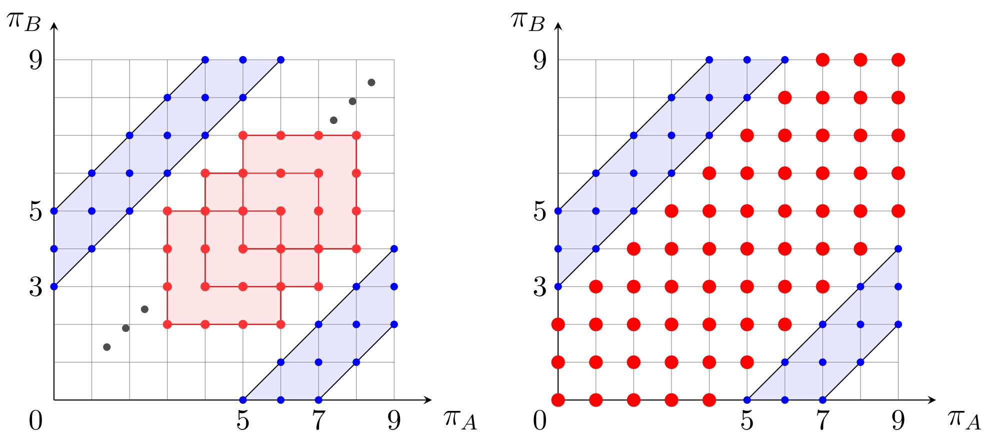


Figure 3.3: Evaluation of subset of $\varphi_{10}(3,-5)$ (left) and $R_{3,-5,10}$ (right) of Example 3.10

Then, as evaluated in Example 3.9

$$\begin{aligned}\delta(3,-5) &= 7\\ \delta y(3,-5) &= 3\\ \delta x(3,-5) &= 3.\end{aligned}$$

Applying φ_{t_T} of Definition 3.11 for $l = 3$, $u = -5$ and $t_T = 10$, we get

$$\begin{aligned}\varphi_{10}(3,-5) = \{&([x_1, x_1 + \delta x(l,u)] \times [x_2, x_2 + \delta y(l,u)])\\ &\mid \forall x_2 \in [-3,9] : x_1 \geq -3 \wedge x_1 \leq 9 : x_1 = x_2 + 1\}\\ = \{&([-2,1] \times [-3,0]), ([-1,2] \times [-2,1]), ([0,3] \times [-1,2]),\\ &([1,4] \times [0,3]), ([2,5] \times [1,4]), ([3,6] \times [2,5]),\\ &([4,7] \times [3,6]), ([5,8] \times [4,7]), ([6,9] \times [5,8]),\\ &([7,10] \times [6,9]), ([8,11] \times [7,10])\}\\ K := \{&([3,6] \times [2,5]), ([4,7] \times [3,6]),\\ &([5,8] \times [4,7])\} \subset \varphi_{10}(3,-5)\end{aligned}$$

To get all not feasible pairs, we calculate R_{l,u,t_T} of Definition 3.10:

$$\begin{aligned}R_{3,-5,10} &= \{(a,b) \in [0,9] \times [0,9] \mid b - a \in [-4,2]\}\\ &= \{(0,0), (0,1), (0,2), (1,0), (1,1), (1,2), (1,3), (2,0), \ldots\}\end{aligned}$$

K and $R_{3,-5,10}$ are displayed in Figure 3.3.

In the previous example, it can be visually seen, that φ_{t_T} covers all not feasible pairs of R_{l,u,t_T}, but does not influence any feasible pair. This will be verified by the following lemma.

Definition 3.12. *Let a be a constraint with period $t_T \in \mathbb{N}$ and P_a be the set of not feasible pairs of Definition 3.3. Then*

$$S_a := [0, t_T - 1] \times [0, t_T - 1] \setminus P_a$$

is the set of feasible pairs of a.

Lemma 3.3. *Let $u, l \in \mathbb{Z}$ with $u + 1 < l$ and $t_T \in \mathbb{N}$ and a time consuming constraint $a = ((n_1, n_2), [l_a, u_a]_{t_T})$, such that $(l, u) \in \{(l_a + i \cdot t_T, u_a + (i-1) \cdot t_T) \mid i \in \mathbb{Z}\}$. Then, it holds*

$$\text{(i) } R_{l,u,t_T} \subseteq \bigcup_{A \in \varphi_{t_t}(l,u)} A$$

$$\text{(ii) } S_a \cap \bigcup_{A \in \varphi_{t_t}(l,u)} A = \emptyset$$

Proof. (i): $(a, b) \in R_{l,u,t_T}$. To show:

$$\begin{aligned} &(a,b) \in \bigcup_{A \in \varphi_{t_t}(l,u)} A \\ \Leftrightarrow\ &\exists A \in \varphi_{t_T}(l,u) : (a,b) \in A \end{aligned}$$

With Definition 3.10, it follows

$$\begin{aligned} &(a,b) \in R_{l,u,t_T} \\ \Rightarrow\ &a, b \in [0, t_T - 1] \wedge b - a \in [u+1, l-1] \\ \Leftrightarrow\ &a, b \in [0, t_T - 1] \wedge a \in [-l+1-b, -u-1-b] \\ \Leftrightarrow\ &a, b \in [0, t_T - 1] \wedge a \in [-l+1, -u-1] - b \end{aligned}$$

To show:

$$\begin{aligned} &\forall b \in [-, t_T - 1] : \forall a \in ([-l+1, -u-1] - b) \wedge a \in [0, t_T - 1] : \\ &\quad \exists x_2 \in [-\delta y(l,u), t_T - 1], x1 = x2 - u - 1 - \delta x(l,u) : (a,b) \in I \times J \end{aligned} \tag{26}$$

with

$$\begin{aligned} I &:= [x_1, x_1 + \delta x(l,u)] \\ J &:= [x_2, x_2 + \delta y(l,u)], \end{aligned}$$

which is basically Definition 3.11.

$$\forall b \in [0, t_T - 1] : b \in J,$$

because $[0, t_T - 1] \subseteq J$, since $\delta y(l.u) \geq 0$ and $x_2 \in [-\delta y(l,u), t_T - 1]$.

We can conclude, that for an arbitrary, but fixed $b \in [0, t_T - 1]$

$$x_2 \in [b - \delta y(l,u), b]. \tag{27}$$

It remains to show:

$$a \in I$$

We know, that

$$\begin{aligned} a &\in ([-l+1, -u-1] + b) \cap [0, t_T - 1] \\ &= ([-l+1, -u-1] + b) \cap ([-b, t_T - 1 - b] + b) \\ &= b + ([-l+1, -u-1] \cap [-b, t_T - 1 - b]) \\ &=: L \end{aligned}$$

and further

$$\begin{aligned} I &= [x_1, x_1 + \delta x(l,u)] \\ &\overset{(26)}{=} [x_2 - u - 1 - \delta x(l,u), x_2 - u - 1 - \delta x(l,u) + \delta x(l,u)] \\ &= [x_2 - u - 1 - \delta x(l,u), x_2 - u - 1] \\ &= x_2 - u - 1 + [-\delta x(l,u), 0] \\ &\overset{(27)}{\subseteq} [b - \delta y(l,u), b] - u - 1 + [-\delta x(l,u), 0] \\ &= b + [-\delta y(l,u), 0] - u - 1 + [\delta x(l,u), 0] \\ &= b - u - 1 + [-\delta x(l,u) - \delta y(l,u), 0] \\ &=: Z \end{aligned}$$

To show:

$$\begin{aligned} & & L &\subseteq Z \\ &\Leftrightarrow & b + ([-l+1, -u-1] \cap [-b, t_T - 1 - b]) &\subseteq b - u - 1 + [-\delta x(l,u) - \delta y(l,u), 0] \\ &\Leftrightarrow & [-l+1, -u-1] \cap [-b, t_T - 1 - b] &\subseteq -u - 1 + [-\delta x(l,u) - \delta y(l,u), 0] \end{aligned}$$

Because $[-l+1, -u-1] \cap [-b, t_T - 1 - b] \subseteq [-l+1, -u-1]$, it is sufficient to show

$$\begin{aligned} & & [-l+1, -u-1] &\subseteq -u - 1 + [-\delta x(l,u) - \delta y(l,u), 0] \\ &\Leftrightarrow & [-l+1, -u-1] + u + 1 &\subseteq [-\delta x(l,u) - \delta y(l,u), 0] \\ &\Leftrightarrow & [-l+1+u+1, 0] &\subseteq [-\delta x(l,u) - \delta y(l,u), 0] \\ &\Leftrightarrow & -[0, l-u-2] &\subseteq -[0, \delta x(l,u) + \delta y(l,u)] \\ &\Leftrightarrow & [0, l-u-2] &\subseteq [0, \delta x(l,u) + \delta y(l,u)] \end{aligned}$$

To show:

$$l - u - 2 \le \delta x(l,u) + \delta y(l,u)$$

$$\begin{aligned}\delta x(l,u) + \delta y(l,u) &= \left\lceil \frac{\delta(l,u)}{2} \right\rceil - 1 + \left\lfloor \frac{\delta(l,u)}{2} \right\rfloor \\ &= \delta(l,u) - 1 = l - u - 1 - 1 = l - u - 2 \geq l - u - 2\end{aligned}$$

(ii): Let $(a,b) \in S_a$. With the assumption $(l,u) \in \{(l_a + i \cdot t_T, u_a + (i-1) \cdot t_T) \mid i \in \mathbb{Z}\}$ we know, that

$$R_{l,u,t_T} \subseteq P_a, \tag{28}$$

because $R_{l,u.t_T}$ is the complement set between u, l with respect to Definition 2.20 with arbitrary potentials.

With

$$S_a \cap P_a \overset{(28)}{\subseteq} S_a \cap R_{l,u,t_T} = \emptyset$$

and (i) it follows, that

$$(a,b) \notin \bigcup_{A \in \varphi_{t_t}(l,u)} A$$

□

As seen in Figure 3.3, we need to cover as well the not feasible regions in the upper left and lower right corners. To identify all not feasible regions, the following definition is needed. In order to proof the sufficiency, that all not feasible pairs of a constraint are excluded, it will be followed by an appropriate lemma.

Definition 3.13. *Let $a = ((n_1, n_2), [l,u]_{t_T})$ be a time consuming constraint. Then with*

$$k := \begin{cases} -1 & 0 \notin [l,u]_{t_T} \\ 0 & 0 \in [l,u]_{t_T} \end{cases}$$

$$\begin{aligned}\zeta_{t_T} \colon \mathbb{Z} \times \mathbb{Z} &\to 2^{(\mathbb{Z}\times\mathbb{Z})\times(\mathbb{Z}\times\mathbb{Z})} \\ (l,u) &\mapsto \bigcup_{i \in [k,1]} \varphi_{t_T}(l + i \cdot t_T, u + (i-1) \cdot t_T)\end{aligned}$$

maps to the set of all not feasible rectangles of a.

Lemma 3.4. *Let $a = ((n_1, n_2), [l,u]_{t_T})$ be a time consuming constraint. Then, it holds*

(i) $P_a \subseteq \bigcup_{A \in \zeta_{t_t}(l,u)} A$

(ii) $S_a \cap \bigcup_{A \in \zeta_{t_t}(l,u)} A = \emptyset$

Proof. (i): Two cases

1. $0 \notin [l,u]_{t_T}$
2. $0 \in [l,u]_{t_T}$

1. Without loss of generality

$$l \in [1, t_T - 1], u \in [l, t_T - 1] \tag{29}$$

$0 \notin [l, u]_{t_T} \Rightarrow k = -1$ with k of Definition 3.13. Hence,

$$\zeta_{t_t}(l, u) = \varphi_{t_T}(l - t_T, u - 2 \cdot t_T) \cup \varphi_{t_T}(l, u - t_T) \cup \varphi_{t_T}(l + t_T, u)$$

It is sufficient to show, that

$$P_a \subseteq R_{l-t_T,u-2\cdot t_T,t_T} \cup R_{l,u-t_T,t_T} \cup R_{l+t_T,u,t_T} =: Z, \tag{30}$$

because, with Lemma 3.3, we know

$$\begin{aligned} & R_{l-t_T,u-2\cdot t_T,t_T} \subseteq \bigcup_{A \in \varphi_{t_T}(l-t_T,u-2\cdot t_T)} A \\ \wedge \quad & R_{l,u-t_T,t_T} \subseteq \bigcup_{A \in \varphi_{t_T}(l,u-t_T)} A \\ \wedge \quad & R_{l+t_T,u,t_T} \subseteq \bigcup_{A \in \varphi_{t_T}(l+t_T,u)} A \end{aligned}$$

$$\begin{aligned} & \Rightarrow Z \subseteq \bigcup_{A \in \zeta_{t_t}(l,u)} A \\ & \overset{(30)}{\Rightarrow} P_a \subseteq \bigcup_{A \in \zeta_{t_t}(l,u)} A \end{aligned}$$

It remains to show (30):

$$\begin{aligned} (30) & \Leftrightarrow \forall (x, y) \in P_a : (x, y) \in Z \\ & \Leftrightarrow \forall (x, y) \in \{(x, y) \mid (x, y) \text{ does not hold for } a\} : (x, y) \in Z \end{aligned}$$

Let $(x, y) \in P_a$ arbitrary, but fixed. We know

$$\begin{aligned} & (x, y) \in [0, t_T - 1] \times [0, t_T - 1] =: H \\ & \wedge \quad y - x \notin [l, u]_{t_T} \end{aligned} \tag{31}$$

$$\begin{aligned} & \Leftrightarrow (x, y) \in H \wedge y - x \in \mathbb{Z} \setminus [l, u]_{t_T} \\ & \Leftrightarrow (x, y) \in H \wedge y - x \in [u + 1, l - 1]_{t_T} \\ & \overset{(29),(30)}{\Leftrightarrow} (x, y) \in H \wedge y - x \in [u + 1 - 2 \cdot t_T, l - t_T - 1] \\ & \qquad \cup [u + 1 - t_T, l - 1] \cup [u + 1, l + t_T - 1] \\ & \overset{Def\ 3.10}{\Rightarrow} (x, y) \in R_{l-t_T,u-2\cdot t_T,t_T} \vee (x, y) \in R_{l,u-t_T,t_T} \vee (x, y) \in R_{l+t_T,u,t_T} \\ & \Rightarrow (x, y) \in Z \end{aligned}$$

2. analogous to 1.
(ii): follows directly by (i) and Lemma 3.3, because ζ_{t_T} is constructed by a union of φ_{t_T}. □

Now, we are able to collect all not feasible rectangles via ζ_{t_T} for a time consuming constraint. This leads to the remaining question, how a rectangle can be encoded, such that it uses only the propositional variables given in Definition 3.5. The strategy will basically the same as in Example 3.8.

Definition 3.14. *Let $A = ([x_1, x_2] \times [y_1, y_2]) \in 2^{\mathbb{Z}} \times 2^{\mathbb{Z}}$ be a rectangle with $(x, y) \in A$. Then*

$$\begin{aligned} encode\_ordered\_rec : 2^{\mathbb{Z}} \times 2^{\mathbb{Z}} &\to \mathcal{L}(\Sigma_{\mathsf{SAT}}) \\ [x_1, x_2] \times [y_1, y_2] &\mapsto (\neg q_{x,x_2} \vee q_{x,x_1-1} \vee \neg q_{y,y_2} \vee q_{y,y_1-1}) \end{aligned}$$

is the order encoding function for the to be excluded rectangle A with $q_{l,k}$ ($l \in \{x, y\}$, $k \in dom(l)$) being the propositional variables of Definition 3.5.

Since $encode\_ordered\_rec$ maps to a disjunction of literals, it is a clause. The following lemma ensures, that all pairs within an encoded rectangle are really not feasible.

Lemma 3.5. *Let $A = ([x_1, x_2] \times [y_1, y_2]) \in 2^{\mathbb{Z}} \times 2^{\mathbb{Z}}$ be a rectangle with $(x, y) \in A$ and $q_{l,k} \in \mathcal{R}$ ($l \in \{x, y\}$, $k \in dom(l)$) being the propositional variables of Definition 3.5. Then*

$$I \models encode\_ordered\_rec(A) \Leftrightarrow (\xi_x(I), \xi_y(I)) \notin A$$

with I being an interpretation and $\xi_n(I)$ being the extracted potential of Definition 3.6.

Proof. "$\Rightarrow$":

$$I \models encode\_ordered\_rec(A) \Rightarrow [\neg q_{x,x_2}, q_{x,x_1-1}, \neg q_{y,y_2}, q_{y,y_1-1}]^I = true \tag{32}$$

Proof by contradiction: assume $(\xi_x(I), \xi_y(I)) \in A = ([x_1, x_2] \times [y_1, y_2])$. Then

$$x_1 \leq \xi_x(I) \wedge x_2 \geq \xi_x(I) \wedge y_1 \leq \xi_y(I) \wedge y_2 \geq \xi_y(I)$$

which results in

$$\begin{aligned} &q^I_{x,x_2} = true \wedge q^I_{x,x_1-1} = false \wedge \neg q^I_{y,y_2} = true \wedge q^I_{y,y_1-1} = false \\ \Rightarrow\ &[\neg q_{x,x_2}, q_{x,x_1-1}, \neg q_{y,y_2}, q_{y,y_1-1}]^I = false \end{aligned}$$

This is a contradiction to (32). Hence,

$$(\xi_x(I), \xi_y(I)) \notin A$$

"$\Leftarrow$": analogous to "$\Rightarrow$". □

Definition 3.15. *Let $a = ((n_1, n_2), [l, u]_{t_T}) \in \mathcal{C}$ be a time consuming constraint. Then*

$$\begin{aligned} encode\_ordered\_time\_con : \mathcal{C} &\to \mathcal{L}(\Sigma_{\mathsf{SAT}}) \\ ((n_1, n_2), [l, u]_{t_T}) &\to \bigwedge_{A \in \zeta_{t_T}(l,u)} encode\_ordered\_rec(A) \end{aligned}$$

is the order encoding function of a time consuming constraint.

Because $encode\_ordered\_time\_con$ is a conjunction of clauses, it is in conjunctive normal form.

3.5.3. Encoding of Symmetry Constraints

The order encoding of symmetry constraints follows exactly the same strategy as for time consuming constraints. A lot of functions and definitions can be directly used of Section 3.5.2.

Definition 3.16. *Let $u, l \in \mathbb{Z}$ be two integers with $u < l$ and $t_T \in \mathbb{N}$. Then*

$$K_{l,u,t_T} := \{(a, b) \in [0, t_T - 1] \times [0, t_T - 1] \mid b + a \in [u + 1, l - 1]\}$$

is the set of not feasible pairs *between u and l.*

Definition 3.17. *Let $u, l \in \mathbb{Z}$ be two integers with $u < l$ and $t_T \in \mathbb{N}$. Then*

$$\begin{aligned} \psi_{t_T} \colon \mathbb{Z} \times \mathbb{Z} &\to 2^{(2^{\mathbb{Z}} \times 2^{\mathbb{Z}})} \\ (l, u) &\mapsto \{([x_1, x_1 + \delta x(l, u)] \times [x_2, x_2 + \delta y(l, u)]) \\ &\qquad \mid \forall x_2 \in \{-\delta y(l, u), \ldots, t_T - 1\} : \\ &\qquad\ \ x_1 + \delta x(l, u) \geq 0 \wedge x_1 \leq t_T - 1 : \\ &\qquad\ \ x_1 = -x_2 + l - 1 - \delta x(l, u)\} \end{aligned}$$

maps (l, u) to the set of rectangles *between u and l.*

ζ_{t_T} mapped to all not feasible rectangles for a time consuming constraint. Likewise, we need such a function for a symmetry constraint.

Definition 3.18. *Let $a = ((n_1, n_2), [l, u]_{t_T})$ be a symmetry constraint. Then with*

$$k := \begin{cases} 2 & 0 \notin [l, u]_{t_T} \\ 1 & 0 \in [l, u]_{t_T} \end{cases}$$

$$\begin{aligned} \Lambda_{t_T} \colon \mathbb{Z} \times \mathbb{Z} &\to 2^{(\mathbb{Z} \times \mathbb{Z}) \times (\mathbb{Z} \times \mathbb{Z})} \\ (l, u) &\mapsto \bigcup_{i \in [0,k]} \psi_{t_T}(l + i \cdot t_T, u + (i - 1) \cdot t_T) \end{aligned}$$

maps to the set of all not feasible rectangles of a.

Furthermore, all lemmas of the previous sections can be applied here as well. This leads to the following needed lemma.

Lemma 3.6. *Let $a = ((n_1, n_2), [l, u]_{t_T})$ be a symmetry constraint. Then, it holds*

(i) $(x, y) \in P_a \Rightarrow (x, y) \in \bigcup_{A \in \Lambda_{t_t}(l,u)} A$

(ii) $(x, y) \in S_a \Rightarrow (x, y) \notin \bigcup_{A \in \Lambda_{t_t}(l,u)} A$

with P_a and S_a of Definition 3.12

Proof. Analogous to proof of Lemma 3.4. □

In order to encode a single rectangle, we use the function $encode\_ordered\_rec$ of Definition 3.14. Finally, we need the following definition to encode a symmetry constraint.

Definition 3.19. *Let $a = ((n_1, n_2), [l, u]_{t_T}) \in \mathcal{S}$ be a symmetry constraint. Then*

$$\begin{aligned} encode\_ordered\_sym\_con : \mathcal{S} &\to \mathcal{L}(\Sigma_{SAT}) \\ ((n_1, n_2), [l, u]_{t_T}) &\mapsto \bigwedge_{A \in \Lambda_{t_T}(l,u)} encode\_ordered\_rec(A) \end{aligned}$$

is the order encoding function of a symmetry constraint.

Since $encode\_ordered\_sym\_con$ is a conjunction of clauses, it is in conjunctive normal form.

3.5.4. Encoding of the PESP

Having the gathered information of the previous sections, we are able to order encode a whole PESP with respect to a periodic event network. Besides the defined $\Omega^{\mathcal{V}}_{ordered}$, we need the conjunction for the constraints.

Definition 3.20. *Let $a \in \mathcal{A}$ be a constraint and $\mathcal{A} = \mathcal{C} \cup \mathcal{S}$ according to Definition 2.16. Then*

$$\begin{aligned} encode\_ordered\_con : \mathcal{A} &\to \mathcal{L}(\Sigma_{SAT}) \\ a &\mapsto \begin{cases} encode\_ordered\_time\_con(a) & a \in \mathcal{C} \\ encode\_ordered\_sym\_con(a) & a \in \mathcal{S} \end{cases} \end{aligned}$$

maps a to its corresponding order encoding.

Definition 3.21. *Let $\mathcal{A}$ be a set of constraints. Then*

$$\Psi^{\mathcal{A}}_{ordered} := \bigwedge_{a \in \mathcal{A}} encode\_ordered\_con(a).$$

is the propositional formula in conjunctive normal form of all constraints $a \in \mathcal{A}$.

Furthermore, because we know, that both $\Psi^{\mathcal{A}}_{ordered}$ and $\Omega^{\mathcal{A}}_{ordered}$ must hold, we simply conjunct them. This results in the following definition.

Definition 3.22. *Let $\mathcal{N} = (\mathcal{V}, \mathcal{A}, t_T)$ be periodic event network. Then the function*

$$\begin{aligned} encode\_direct\_pesp : 2^{\mathcal{V}^+} \times 2^{\mathcal{A}^+_{t_T}} \times 2^{\mathbb{N}} &\to \mathcal{L}(\Sigma_{SAT}) \\ (\mathcal{V}, \mathcal{A}, t_T) &\mapsto (\Omega^{\mathcal{V}}_{ordered} \wedge \Psi^{\mathcal{A}}_{ordered}) \end{aligned}$$

is the order encoding function of a PESP *with respect to $\mathcal{N}$.*

The usage of this function will be demonstrated in the following example.

Example 3.11. Let $\mathcal{N} = (\mathcal{V}, \mathcal{A}, 10)$ be the periodic event network of Example 2.7. That is

$$\mathcal{V} = \{A, B, C\}$$
$$\mathcal{A} = \mathcal{C} = \{a_1, a_2, a_3\},$$

with

$$a_1 := ((A, B), [3, 5]_{10})$$
$$a_2 := ((B, C), [2, 2]_{10})$$
$$a_3 := ((C, A), [2, 4]_{10}).$$

Order encoding this function means to apply $encode\_direct\_pesp$. This yields

$$\begin{aligned} encode\_ordered\_pesp(\mathcal{N}) = \;& (\Omega^{\mathcal{V}}_{ordered} \wedge \Psi^{\mathcal{A}}_{ordered}) \\ = \;& \left(\bigwedge_{n \in \mathcal{V}} encode\_ordered(\pi_n)\right) \\ & \wedge \left(\bigwedge_{a \in \mathcal{A}} encode\_ordered\_con(a)\right) \\ = \;& (encode\_ordered(\pi_A) \wedge encode\_ordered(\pi_B) \\ & \wedge encode\_ordered(\pi_C) \wedge encode\_ordered\_con(a_1) \\ & \wedge encode\_ordered\_con(a_2) \\ & \wedge encode\_ordered\_con(a_3)) \end{aligned}$$

Applying function $encode\_ordered$, we get for event's potential π_A

$$\begin{aligned} encode\_ordered(\pi_A) = \;& \langle [\neg q_{\pi_A,0}, q_{\pi_A,1}], [\neg q_{\pi_A,1}, q_{\pi_A,2}], [\neg q_{\pi_A,2}, q_{\pi_A,3}], [\neg q_{\pi_A,3}, q_{\pi_A,4}], \\ & [\neg q_{\pi_A,4}, q_{\pi_A,5}], [\neg q_{\pi_A,5}, q_{\pi_A,6}], [\neg q_{\pi_A,6}, q_{\pi_A,7}], [\neg q_{\pi_A,7}, q_{\pi_A,8}] \rangle. \end{aligned}$$

Likewise, we use this function for π_B and π_C.

In order to encode the constraints a_i $(i \in \{1, 2, 3\})$, we first need to calculate the sets $\zeta_{10}(a_i)$ $(i \in \{1, 2, 3\})$ of Definition 3.13. This will be demonstrated for the time consuming constraint a_1:

$$\begin{aligned} \zeta_{10}(a_1) \overset{0 \notin [3,5]_{10}}{=} \;& \varphi_{10}(3 - 10, 5 - 20) \cup \varphi_{10}(3, 5 - 10) \cup \varphi_{10}(3 + 10, 5) \\ = \;& \varphi_{10}(-7, -15) \cup \varphi_{10}(3, -5) \cup \varphi_{10}(13, 5) \\ \overset{Ex\ 3.10}{=} \;& \varphi_{10}(-7, -15) \cup \varphi_{10}(13, 5) \\ & \cup \{([-2, 1] \times [-3, 0]), ([-1, 2] \times [-2, 1]), ([0, 3] \times [-1, 2]), \\ & ([1, 4] \times [0, 3]), ([2, 5] \times [1, 4]), ([3, 6] \times [2, 5]), \\ & ([4, 7] \times [3, 6]), ([5, 8] \times [4, 7]), ([6, 9] \times [5, 8]), \\ & ([7, 10] \times [6, 9]), ([8, 11] \times [7, 10])\} \end{aligned}$$

Afterwards, applying $encode\_ordered\_con$ yields

$$\begin{aligned}
encode\_ordered\_con(a_1) &= encode\_ordered\_time\_con(a_1) \\
&= encode\_ordered\_time\_con(\,((A,B),[3,5]_{10})\,) \\
&= \bigwedge_{R \in \zeta_{10}(3,-5)} encode\_ordered\_rec(R) \\
&= (\ldots \wedge encode\_ordered\_rec([3,6] \times [2,5]) \\
&\quad \wedge encode\_ordered\_rec([4,7] \times [3,6]) \wedge \ldots) \\
&= (\ldots \wedge (\neg q_{\pi_A,6} \vee q_{\pi_A,2} \vee \neg q_{\pi_B,5} \vee q_{\pi_B,1}) \\
&\quad \wedge (\neg q_{\pi_A,7} \vee q_{\pi_A,3} \vee \neg q_{\pi_B,6} \vee q_{\pi_B,2}) \wedge \ldots) \\
&= \langle \ldots, [\neg q_{\pi_A,6}, q_{\pi_A,2}, \neg q_{\pi_B,5}, q_{\pi_B,1}], \\
&\quad [\neg q_{\pi_A,7}, q_{\pi_A,3}, \neg q_{\pi_B,6}, q_{\pi_B,2}], \ldots \rangle
\end{aligned}$$

The whole encoding can be found as SAT instance on CD as of Section A.

The following theorem and corollary ensures the correct possibility to use the order encoding method of a PESP with respect to a given periodic event network $\mathcal{N}$. The fundamentals of this proof are already given, because we can extract exactly one value for a potential (Lemma 3.2), we can exclude all pairs within an encoded rectangle (Lemma 3.5) and we can cover each not feasible pair of a constraint with a set of rectangles (Lemma 3.4, Lemma 3.6). We have to generalize this for a set of potentials and a set of constraints.

Theorem 3.2 (Soundness and Completeness Order Encoding). *Let $\mathcal{N} = (\mathcal{V}, \mathcal{A}, t_T)$ be a periodic event network and*

$$F := encode\_ordered\_pesp(\mathcal{N}) \in \mathcal{L}(\Sigma_{SAT})$$

be the order encoded propositional formula of $\mathcal{N}$. Then

$$\exists I : I \models F \Leftrightarrow \exists \Pi_{\mathcal{V}} : \Pi_{\mathcal{V}} \textit{ is valid}$$

with I being an interpretation and $\Pi_{\mathcal{V}}$ a schedule of $\mathcal{N}$.

Proof.

$$
\begin{aligned}
& \exists I : I \models F \\
\overset{Def\,3.22}{\Leftrightarrow}\ & \exists I : I \models (\Omega^{\mathcal{V}}_{ordered} \land \Psi^{\mathcal{A}}_{ordered}) \\
\Leftrightarrow\ & \exists I : I \models \Omega^{\mathcal{V}}_{ordered} \land I \models \Psi^{\mathcal{A}}_{ordered} \\
\overset{(24)}{\Leftrightarrow}\ & \exists I : I \models \bigwedge_{n \in \mathcal{V}} encode\_ordered(\pi_n) \land I \models \Psi^{\mathcal{A}}_{ordered} \\
\overset{Lem\,3.2,\ (25)}{\Leftrightarrow}\ & \exists I : \forall n \in \mathcal{V} : \Pi_{\mathcal{V}}(n) := \xi_{\pi_n}(I) \land I \models \Psi^{\mathcal{A}}_{ordered} \\
\overset{Def\,3.21}{\Leftrightarrow}\ & \exists I : \forall n \in \mathcal{V} : \Pi_{\mathcal{V}}(n) := \xi_{\pi_n}(I) \\
& \qquad \land I \models \bigwedge_{a \in \mathcal{A}} encode\_ordered\_con(a) \\
\Leftrightarrow\ & \exists I : \forall n \in \mathcal{V} : \Pi_{\mathcal{V}}(n) := \xi_{\pi_n}(I) \\
& \qquad \land \forall a \in \mathcal{A} : I \models encode\_ordered\_con(a) \\
\overset{Lem\,3.5,\ Lem\,3.4,\ Lem\,3.6}{\Leftrightarrow}\ & \exists I : \forall n \in \mathcal{V} : \Pi_{\mathcal{V}}(n) := \xi_{\pi_n}(I) \land \forall a \in \mathcal{A} : a \text{ holds under } \Pi_{\mathcal{V}} \\
\overset{Def\,2.23}{\Leftrightarrow}\ & \exists I : \forall n \in \mathcal{V} : \Pi_{\mathcal{V}}(n) := \xi_{\pi_n}(I) \land \Pi_{\mathcal{V}} \text{ is valid} \\
\overset{Lem\,3.2,\ Def\,3.6}{\Leftrightarrow}\ & \exists \Pi_{\mathcal{V}} : \Pi_{\mathcal{V}} \text{ is valid}
\end{aligned}
$$

□

We can directly derive the following corollary.

Corollary 3.2 (Soundness and Completeness Order Encoding). *Let* $\mathcal{N} = (\mathcal{V}, \mathcal{A}, t_T)$ *be a periodic event network and*

$$F := encode\_ordered\_pesp(\mathcal{N}) \in \mathcal{L}(\Sigma_{SAT})$$

be the order encoded propositional formula of $\mathcal{N}$. *Then*

$$\nexists I : I \models F \Leftrightarrow \nexists \Pi_{\mathcal{V}} : \Pi_{\mathcal{V}} \textit{ is valid}$$

with I *being an interpretation and* $\Pi_{\mathcal{V}}$ *a schedule of* $\mathcal{N}$.

Proof. Analogous to proof of Corollary 3.1 □

3.6. Comparison between Direct Encoding and Order Encoding

In this section, the direct and order encoding approach are compared with respect to number of clauses and number of variables. The needed time to solve such an instance will be explored and evaluated in Section 4, as well as the verification of this' section's results.

Since $encode\_direct\_pesp$ and $encode\_ordered\_pesp$ are mappings to propositional formulas in conjunctive normal form, we can handle them as set of clauses, which are handled as sets of literals. This results in the following definitions. With $|x|$ being the cardinality of a set x or the number of clauses, if x is a propositional formula in CNF, it follows

Definition 3.23. *Let $F \in \mathcal{L}(\Sigma_{SAT})$ a propositional formula in conjunctive normal form. Then*

$$|F| \in \mathbb{N}$$

is the number of clauses *of F.*

Definition 3.24. *Let $F \in \mathcal{L}(\Sigma_{SAT})$ a propositional formula in conjunctive normal form. Then*

$$\begin{aligned} vars\colon \mathcal{L}(\Sigma_{SAT}) &\to 2^{\mathcal{R}} \\ F &\mapsto \{p \in \mathcal{R} \mid C \in F : p \in C \vee \neg p \in C\} \end{aligned}$$

maps to the set of variables, *that occur in F with $C \in F$ being a clause.*

Hence, $|vars(F)|$ is the number of variables, that occur in F. The remaining question is, how we can estimate or precisely evaluate these values with respect to a given encoding F.

3.6.1. Number of Variables

With F being a direct encoded propositional formula with respect to a given periodic event network $\mathcal{N} = (\mathcal{V}, \mathcal{A}, t_T)$, we know with Definition 3.2, that we have for each potential π_n $(n \in \mathcal{V})$ t_T variables, since $\pi_n \in [0, t_T - 1]$. Consequently, we can conclude, that for all periodic event networks $\mathcal{N}$

$$|vars(encode\_direct\_pesp(\mathcal{N}))| = |vars(F)| = t_T \cdot |\mathcal{V}| \tag{33}$$

Consider a propositional formula G, that is the order encoded representation of the periodic event network $\mathcal{N}$. We know with Definition 3.5, that for each potential, we have $t_T - 1$ variables. Hence,

$$|vars(encode\_ordered\_pesp(\mathcal{N}))| = |vars(G)| = (t_T - 1) \cdot |\mathcal{V}| \tag{34}$$

Thus, there are $|\mathcal{V}|$ less variables, if the network is order encoded than direct encoded. Typically, the period is $t_T = 60$ or $t_T = 120$. This results in a rather small difference with respect to variable count. We can conclude

$$|vars(encode\_ordered\_pesp(\mathcal{N}))| \approx |vars(encode\_direct\_pesp(\mathcal{N}))| \tag{35}$$

for a greater period t_T under the practical assumption, that $|\mathcal{V}|$ is rather large compared to t_T.

3.6.2. Estimation of Clause Count

Estimating the clause count needs to consider the clause counts of $\Omega_t^{\mathcal{V}}$ and $\Psi_t^{\mathcal{A}}$ $(t \in \{direct, ordered\})$, because each encoding results in the propositional formula

$$\Omega_t^{\mathcal{V}} \wedge \Psi_t^{\mathcal{A}} \quad (t \in \{direct, ordered\})$$

and, thus, it yields the homomorphism

$$|\Omega_t^{\mathcal{V}} \wedge \Psi_t^{\mathcal{A}}| = |\Omega_t^{\mathcal{V}}| + |\Psi_t^{\mathcal{A}}| \quad (t \in \{direct, ordered\}),$$

since it is in conjunctive normal form. Firstly, the variable encodings will be evaluated. Because $\Omega_t^{\mathcal{V}}$ $(t \in \{direct, ordered\})$ conjuncts $|\mathcal{V}|$ times $encode\_direct$ respectively $encode\_ordered$ with the same potential's domain, we know

$$\begin{aligned} |\Omega_{direct}^{\mathcal{V}}| &= |\mathcal{V}| \cdot |encode\_direct(\pi_n)| \quad (n \in \mathcal{V}) \\ |\Omega_{ordered}^{\mathcal{V}}| &= |\mathcal{V}| \cdot |encode\_ordered(\pi_n)| \quad (n \in \mathcal{V}). \end{aligned} \tag{36}$$

With $encode\_direct$ mapping to a nested conjunction of the potential's domain and single conjunction over a reduced potenial's domain for the $encode\_ordered$ function, we get

$$\begin{aligned} |encode\_direct(\pi_n)| &\overset{Def\_3.2}{=} |[0, t_T - 1]| \cdot |[0, t_T - 1]| + 1 \\ &= t_T \cdot t_T + 1 = t_T^2 + 1 \quad (n \in \mathcal{V}) \\ |encode\_ordered(\pi_n)| &\overset{Def\_3.5}{=} |[1, t_T - 2]| = t_T - 2 \quad (n \in \mathcal{V}). \end{aligned} \tag{37}$$

Combining (36) and (37) and applying the big O notation for t_T and $|\mathcal{V}|$, we receive

$$\begin{aligned} |\Omega_{direct}^{\mathcal{V}}| &\in \mathcal{O}(t_T^2 |\mathcal{V}|) \\ |\Omega_{ordered}^{\mathcal{V}}| &\in \mathcal{O}(t_T |\mathcal{V}|). \end{aligned} \tag{38}$$

Secondly, we must evaluate the encoding of the constraints $\Psi_t^{\mathcal{A}}$ $(t \in \{direct, ordered\})$. On the one hand, with $encode\_direct\_con$ is a conjunction over all infeasible pairs $(n, m) \in P_a$ of Definition 3.3, the direct encoding approach yields

$$|\Psi_{direct}^{\mathcal{A}}| = \sum_{a \in \mathcal{A}} |P_a|$$

$|P_a|$ is in $\mathcal{O}(t_T^2)$ for t_T, because for all $(i, j) \in P_a$, we know $(i, j) \in [0, t_T - 1] \times [0, t_T - 1]$. Thus, we can conclude as big O notation

$$|\Psi_{direct}^{\mathcal{A}}| \in \mathcal{O}(t_T^2 |\mathcal{A}|). \tag{39}$$

On the other hand, we know with Definition 3.13, that φ_{t_T} is evaluated either two or three times. Hence, we can conclude

$$|\Psi_{ordered}^{\mathcal{A}}| \in \mathcal{O}(t_T |\mathcal{A}|). \tag{40}$$

Finally, we can conclude with (38), (39) and (40)

$$\begin{aligned} |encode\_direct\_pesp(\mathcal{N})| &\in \mathcal{O}(t_T^2(|\mathcal{V}| + |\mathcal{A}|)) \\ |encode\_ordered\_pesp(\mathcal{N})| &\in \mathcal{O}(t_T(|\mathcal{V}| + |\mathcal{A}|)) \end{aligned} \tag{41}$$

We can conclude, that $|encode\_direct\_pesp(\mathcal{N})|$ should be around t_T times greater than $|encode\_ordered\_pesp(\mathcal{N})|$ for the same instance. This result will be experimentally verified in Section 4.

4. Computational Results

This section will cover the topics, how a PESP can be used to solve an industrial problem and to compare a state-of-the-art PESP solver with a SAT solver for a corresponding encoding of the periodic event network with respect to solution time. In this case the industrial problem will be modeling railway networks with respect to a feasible time table for all trains within the network.

4.1. Modeling Railway Networks as PESP

As stated above, this section covers the modeling of a restriction system, which leads to feasible time tables of trains. Since this is a very complex topic in itself, it can only be discussed shortly in this work. For further investigations, please read [Opi09].

Each train L has in each station s an arrival and departure event $L_{t,s}$ $(t \in \{dep, arr\})$. Each departure and arrival event happens periodically often with period t_T. Between these two events, there is a time consuming constraint $((L_{arr,s}, L_{dep,s}), [l,u]_{t_T})$, which represents the minimum and maximum holding time in station s with l and u, respectively. The time of travel from station s_1 to station s_2 for a train L is represented in the time consuming constraint $((L_{dep,s_1}, L_{arr,s_2}), [k,k]_{t_T})$. k is calculated by the track's length between s_1 and s_2 as well as the driving dynamics of the train L.

In order to omit, that two trains hit on a single-way track, we need to constraint the respective arrival events. The lower and upper bound is calculated by taking into account the times of travel for each train, as well as some buffer times. This results in a time consuming constraint $((L_{arr,s_1}, J_{arr,s_2}), [l,u]_{t_T})$ for trains L and J.

On the other hand, symmetry constraints $((L_{arr,s}, J_{arr,s}), [l,u]_{t_T})$ are defined for two trains L and J in the same station s, which have the same route and same driving dynamics in order to get a so called symmetric time table.

A time table is called conflict free, if and only if the schedule of the corresponding periodic event network is valid.

4.2. Railway Network Instances

The networks for the computations are modeled and converted to a corresponding periodic event network $\mathcal{N}$ at TU Dresden, Faculty of Transportation and Traffic Science, Chair of Traffic Flow Science. It ranges from rather trivial instances with around 1,000 constraints to hard instances with around 10,000 constraints.

The PESPs swg_i $(i \in \{1,\ldots,4\})$ and seg_j $(j \in \{1,2\})$ are subnetworks of south west and south east Germany, respectively. $fernsym$ represents the whole intercity rail traffic network of Germany. All networks are represented in Table 1 with their respective number of nodes and total number of constraints.

The last four columns represent the number of variables and the number of clauses with respect to their corresponding encodings $F := encode\_direct\_pesp(\mathcal{N})$ and $G := encode\_ordered\_pesp(\mathcal{N})$. The search space for the SAT instances, with respect to variable count, is similar for each instance with respect to direct and order encoding,

Table 1: PESP instances and corresponding encodings

	PESP $\mathcal{N} = (\mathcal{V}, \mathcal{A}, t_T)$		direct encoding F		order encoding G	
instance	$\lvert\mathcal{V}\rvert$	$\lvert\mathcal{A}\rvert$	$\lvert vars(F)\rvert$	$\lvert F\rvert$	$\lvert vars(G)\rvert$	$\lvert G\rvert$
swg_2	60	1,145	7,200	2,037,732	7,140	83,740
$fernsym$	128	3,117	15,360	6,657,955	15,232	353,276
swg_4	170	7,107	20,400	6,193,570	20,230	399,191
swg_3	180	2,998	21,600	4,874,144	21,420	214,011
swg_1	221	7,443	26,520	7,601,906	26,299	462,217
seg_2	611	9,863	73,320	25,101,341	72,709	1,115,210
seg_1	1,483	10,351	177,960	34,323,942	176,477	1,348,045

Table 2: PESP instances and corresponding times to solve

instance	$pespsolve$ /s	$direct + riss$ /s	$ordered + riss$ /s	speedup
swg_3	66	50	2	33
swg_2	512	37	2	256
swg_4	912	752	8	114
$fernsym$	2,035	294	7	290
swg_1	>86,400	18	7	>12,342
seg_1	>86,400	16	10	>8,640
seg_2	>86,400	>86,400	11	>7,854

which empirically verifies the assumption of (35). Hence, the interesting question is, how the obvious difference in clause count and the structure of the encoding itself influence the time to solve for the SAT solver.

All instances with their respective encodings can be found on the CD as described in Section A.

4.3. Evaluation and Comparison

As proposed in Section 2.2.3, we use a state-of-the-art PESP solver $pespsolve$ as domain specific solver, developed by, TU Dresden, Faculty of Transportation and Traffic Science, Chair of Traffic Flow Science, especially Prof. Nachtigall [Nac96].

The other two methods are equivalent, despite the encoding method: encoding a given PESP with $encode\_direct\_pesp$ respectively $encode\_ordered\_pesp$. The resulting propositional formula in CNF will be solved by a state-of-the-art SAT solver called riss, developed by Norbert Manthey, TU Dresden, Faculty of Computer Science, Chair of Knowledge Representation and Reasoning [Man10], which are called $direct + riss$ and $ordered + riss$, respectively. The total time to solve will be the time to encode the formula plus the time, which riss, the SAT solver, needs to solve the instance. The precise parameters for the solvers can be found on the CD as of Section A.

All solvers are given a timeout of 24 h respectively 86,400 s. These are represented as

“>86,400” in Table 2, otherwise the needed time to solve (in seconds) is displayed in the respective cell.

It is obvious that the SAT solving approach $ordered + riss$ is a lot faster than the traditional PESP solver $pespsolve$. The speedup, last column in the respective table, ranges from 33 to over 12,000 , which has a huge impact, because from now on, it is possible to solve a lot larger PESP instances than before. This could have several reasons: Firstly, modern SAT solvers use a lot of modern implementation methods, like cache optimizations and variable pre fetching, in order to improve the performance [Man10]. Secondly, the algorithm of SAT solvers do not trivial backtracking, but back jumping over several decision levels and learn lemmas in order to not fall into the same tree, which leads to a conflict. This yields to a tremendous cut in the search tree.

Even within the two different encodings, there is a severe advantage of the order encoding approach. The differences in time between $direct + riss$ and $ordered + riss$ may have two reasons: Firstly, we know with (41), that $encode\_direct\_pesp$ maps to a propositional formula in CNF, which has t_T times the number of clauses, than $encode\_ordered\_pesp$ for the same periodic event network. Thus, the to be solved instance is a lot greater, which leads in general to longer times to solve. Secondly, $encode\_ordered$ uses the fact, that the domain is ordered. Consequently, it does not lose this structure. This seems to help the SAT solver a lot to find a solution in shorter time.

The claim, that $orderes + riss$ is time wise exponentially shorter with linear increase in PESP constraints, will not verified in this work. However, it can be very possibly assumed.

5. Conclusion

In this work it has been shown, that we can successfully reduce a PESP instance to a SAT instance and solve it with a state-of-the-art SAT solver in a very short time frame compared to a state-of-the-art PESP solver.

After introducing some needed preliminaries, the reductions have been defined. The correctness and soundness could be proofed for both approaches successfully with respect to the corresponding PESP. This important proofs ensures the correct ability to use this method to solve PESP instances.

After comparing the two methods, direct and order encoding, they have been applied to several industrial instances. It has been shown that especially the order encoding approach outperforms by far the traditional PESP solver. Thus, this new method can be applied to a whole new set of larger instances, which could not have been solved before.

There are several interesting remaining fields: Improving the encoding itself, applying the presented methods to even larger instances, applying optimization techniques to gain better schedules and improving the PESP solver with the gained knowledge. In the following, these claims will be described.

Several preprocessing methods could be applied to the periodic event network, in order to encode more information into the SAT instance, namely, propagating gained information of constraints to neighboured constraints and cutting search space of the potentials, which are not part of the solution space. This may help the SAT solver to find faster a solution. In addition, the instance can be shortened by removing redundant constraints.

Another encoding, namely "compact order encoding" could be tried. This encoding has even less variables than the presented approaches, but has more clauses than the order encoding method. It is a hybrid between the order encoding and the so called logarithmic encoding. Consequently, the search space could be cut a lot, with the disadvantage of having more clauses and around a doubled clause size compared to an order encoded instance.

Furthermore, it should be tried to solve even larger and harder instances. The largest possible instance for the railway modeling topic would be a union of all subnetworks to a railway network of whole Germany. If it could be achieved to solve such an instance in an adequate time frame, a huge goal would be achieved with respect to solving a single instance for feasibility.

Another interesting part, the optimization of timetables, which has been already discussed a lot for example in [Opi09], could be supported by a fast feasibility checking method, like presented in this work. In the process, several constraints of the periodic network with respect to its bounds and an extracted objective functional will be changed. This could be achieved by an adapted branch and bound algorithm.

On the one hand, it seems obvious that the current algorithms in a generic solver, like a SAT solver, seem to be superior to a lot native domain solvers, like a domain specific PESP solver. On the other hand, the domain specific solvers have often superior knowledge about the domain, which cannot, or have not yet been found, be encoded into a SAT instance. Hence, it could be even faster, if the techniques of a SAT solver could be used in a native domain solver. For example, back jumping in the decision tree and learning lemmas to not fall into the same conflicts and unit propagation [Man10]. Applying these method into a state-of-the-art PESP solver could maybe even outperform the presented method in this work.

As seen, there remain a lot of scientific questions and fields, which should be worth investigating in, since it is an area with high industrial need.

A. CD

The CD contains three directories. `example_instances/` contains the PESP instance as well as its corresponding encodings of Examples 2.7,3.5 and 3.11.

The directory `results_instances/` contains the three directories `pesps/`, `order_encodings/` and `direct_encodings/` with its respective meanings of Section 4. The third directory `solvers/` contains the parameters and versions of the used solvers of Section 4.

Since the direct encoding instances of seg_1 and seg_2 were too large, it has been packed to the respective `*.gz` file.

The whole directory structure is listed below:

```
example_instances/
    example_direct.cnf
    example_ordered.cnf
    example.pesp

results_instances/
    direct_encodings/
        fernsym_direct.cnf
        seg_1_binding_weight_direct.cnf.gz
        seg_2_wholeeast_no_g_direct.cnf.gz
        swg_1_mannheim_basel_direct.cnf
        swg_2_kk_linkerrhein_worms_mannheim_direct.cnf
        swg_3_pv_gv_tradeoff_direct.cnf
        swg_4_3gv_direct.cnf
    order_encodings/
        fernsym_ordered.cnf
        seg_1_binding_weight_ordered.cnf
        seg_2_wholeeast_no_g_ordered.cnf
        swg_1_mannheim_basel_ordered.cnf
        swg_2_kk_linkerrhein_worms_mannheim_ordered.cnf
        swg_3_pv_gv_tradeoff_ordered.cnf
        swg_4_3gv_ordered.cnf
    pesps/
        fernsym.pesp
        seg_1_binding_weight.pesp
        seg_2_wholeeast_no_g.pesp
        swg_1_mannheim_basel.pesp
        swg_2_kk_linkerrhein_worms_mannheim.pesp
        swg_3_pv_gv_tradeoff.pesp
        swg_4_3gv.pesp
solvers/
    pespsolve
    riss
```

B. List of Figures

References

[CBRZ01] Edmund Clarke, Armin Biere, Richard Raimi, and Yunshan Zhu. Bounded model checking using satisfiability solving. *Form. Methods Syst. Des.*, 19:7–34, July 2001.

[CDE08] Cristian Cadar, Daniel Dunbar, and Dawson R. Engler. Klee: Unassisted and automatic generation of high-coverage tests for complex systems programs. In Richard Draves and Robbert van Renesse, editors, *OSDI*, pages 209–224. USENIX Association, 2008.

[Coo71] Stephen A. Cook. The complexity of theorem-proving procedures. In *Proceedings of the third annual ACM symposium on Theory of computing*, STOC '71, pages 151–158, New York, NY, USA, 1971. ACM.

[Höl09] Steffen Hölldobler. *Logik und Logikprogrammierung 1: Grundlagen*. Synchron Wissenschaftsverlag der Autoren, 2009.

[Lau07] D. Lau. *Algebra und Diskrete Mathematik 1: Grundbegriffe der Mathematik, Algebraische Strukturen 1, Lineare Algebra und Analytische Geometrie, Numerische Algebra*. Springer-Lehrbuch. Springer, 2007.

[LM07] Christian Liebchen and Rolf H. Möhring. The modeling power of the periodic event scheduling problem: railway timetables-and beyond. In *Proceedings of the 4th international Dagstuhl, ATMOS conference on Algorithmic approaches for transportation modeling, optimization, and systems*, ATMOS'04, pages 3–40, Berlin, Heidelberg, 2007. Springer-Verlag.

[Man10] Norbert Manthey. Improving sat solvers using state-of-the-art techniques, 2010.

[MZ06] Ilya Mironov and Lintao Zhang. Applications of sat solvers to cryptanalysis of hash functions. In *In Theory and Applications of Satisfiability Testing 2006*, pages 102–115, 2006.

[Nac96] Karl Nachtigall. Periodic network optimization with different arc frequencies. *Discrete Applied Mathematics*, 69(1-2):1–17, 1996.

[Odi94] Michiel A. Odijk. *Construction of periodic timetables, Part 1: A cutting plane algorithm*. 1994.

[Opi09] Jens Opitz. *Automatische Erzeugung und Optimierung von Taktfahrplänen in Schienenverkehrsnetzen*. PhD thesis, Technische Universität Dresden, Fakultät Verkehrswissenschaften, Professur für Verkehrsströmungslehre, 2009.

[TTB11] Tomoya Tanjo, Naoyuki Tamura, and Mutsunori Banbara. A compact and efficient sat-encoding of finite domain csp. In *SAT*, pages 375–376, 2011.