A comparative analysis of optimization algorithms for activity-based applications

Luca Bataillard [*] Janody Pougala [†] Tom Häring [†]
Michel Bierlaire [†]

June 10, 2022

Semester project in Computer Science report
Transport and Mobility Laboratory
School of Architecture, Civil and Environmental Engineering
Ecole Polytechnique Fédérale de Lausanne
`transp-or.epfl.ch`

[*]École Polytechnique Fédérale de Lausanne (EPFL), School of Computer and Communication Sciences, {luca.bataillard@epfl.ch}

[†]École Polytechnique Fédérale de Lausanne (EPFL), School of Architecture, Civil and Environmental Engineering (ENAC), Transport and Mobility Laboratory, Switzerland, {janody.pougala, tom.haering, michel.bierlaire}@epfl.ch

Abstract

In order to forecast travel demand, activity-based models generate schedules for every individual in a population sample. Most models generate these schedules using a sequential rule-based approach. Recent activity-based models are based on econometric utility-maximisation, expressed as a Mixed-Integer Linear Program. In order to improve computational efficiency and model expressiveness, we explore adaptations of the same scheduling problem in constraint programming. In a first effort, we investigate a direct translation of the existing MILP formulation. We then adapt the direct translation in two different manners to model the problem in ways that exploit the advantages of constraint programming. In these new programs, we use idiomatic constraint programming expressions: implications as constraints, global constraints such as ELEMENT, and interval variables. We find that constraint programming not only makes the model more intuitive, but also improves solver performance by at least one order or magnitude.

1 Introduction

Transportation agencies and researchers require travel demand forecasting tools such as Activity-Based Models (ABMs) to make informed decisions. Activity-Based Models assume that travel demand is derived from people's need to participate in activities. Such models generate daily schedules for those activities, subject to various time and space constraints, in order to forecast travel demand.

Two general approaches to activity-based modelling exist: rule-based and econometric. Rule-based models use decision heuristics to build feasible schedules. On the other hand, econometric models generate schedules as consequence of utility maximization.

In traditional econometric models, scheduling choices such as activity location or mode choice are decided sequentially. Decisions at one stage are dependent on decisions made at an earlier stage. This can result in the model failing to capture trade-offs between different objectives of the individual such as activity participation and duration, location, and transport mode mode choice. For example, individuals staying home on a given day will not plan errands located close to their workplace, but will postpone them to a later workday. A sequential model might fail to model this behaviour if activity participation is decided before location.

Pougala et al. (2021) introduces a new approach to econometric activity scheduling based on mixed-integer optimization. This allows all choice dimensions to be taken into account simultaneously as part of a general framework. The dimensions of activity participation, start time and duration, sequencing, location, and mode of transport are modelled as part of a single Mixed-Integer Linear Program (MILP).

Such an optimization framework presents some challenges, in particular when it comes to runtime. In travel demand forecasting, individual schedules are generated for every agent in the model. Given the complexity of the MILP, full-scale simulations of large populations take significantly longer compared to simpler rule-based models.

The aim of this project is to explore how Constraint Programming (CP) models could help with the challenges of the scheduling MILP defined in Pougala et al. (2021). We first cover some differences between CP and MILP and review existing applications of CP to scheduling problems. We then define a equivalent formulation of the optimization problem in CP, up to time discretization. Afterwards, we explore how expressiveness and performance can be improved by writing equivalent CP formulations that utilize advanced Constraint Programming features. Finally, we implement all CP models using the CP-SAT constraint solver, part of Google OR-Tools optimization framework (Perron and Furnon, n.d.), and compare the relative performance of the CP and MILP formulations.

This report is organized in five sections. We provide some context on constraint programming and review similar applications from the literature in section 2. In section 3, we define the modeling problem precisely and present three variants of the CP model: a direct translation, a version where activity duplication for mode and location choice is replaced with index variables, and a version that uses interval variables. We then present our method for measuring model performance and compare the results for each model in section 4. Finally, we present our conclusions in section 5.

2 Literature review

Constraint Programming is an optimization model that presents a similar outward structure to traditional Mixed Integer Linear Programming, with some key differences. The optimization problem is represented as a set of decision variables, an objective function, and a set of constraints that the variables must respect. CP is limited to discrete values but allows for more expressive native constraints, such as implications. MILP can use continuous variables but makes more assumptions about the mathematical structure of the problem, such as enforcing linear constraints.

Crucially, CP solves problems using logical inferences and constraint propagation to reduce the feasible domain. On the other hand, MILP uses relaxations and branch and bound to find optimal solutions to a problem.

Constraint programming is widely used in the field of Artificial Intelligence (AI) and often applied to scheduling problems. Pape (2005) explains how CP can be used to represent common scheduling problems. The author defines several properties of scheduling problems. He shows common representations of scheduling constraints and explains the mechanisms behind constraint propagation.

The link between CP and traditional linear optimization is explored in Hooker (2002). The paper lays out the history of the two fields and what each brings to the table. The main advantage of CP is global, efficient and expressive constraints such as implications, ALL-DIFFERENT, or ELEMENT. In contrast, MILPs have efficient objective maximization thanks to relaxation techniques. The paper tries to find how to combine the strengths of each framework

Constraint programming for activity-based modelling, however, is not common practice. Most of the literature regarding CP for scheduling comes from the AI community. Refanidis and Yorke-Smith (2010) applies CP to the problem of individual activity scheduling. The scheduling model in the paper is based on the maximization of a utility function and fulfills a similar function to the one in Pougala et al. (2021).

Other studies, not covering the exact same problem of activity-based scheduling, offer insights on how to apply constraint programming to scheduling problems. Naderia et al. (n.d.) explores different ways of expressing the job-shop scheduling problem in CP, and compares them to an equivalent MILP model. Ways of expressing employee, train, and quay-crane scheduling constraints are explored in Weil et al., 1995, Rodriguez, 2007, and Unsal and Oguz, 2013 respectively.

3 Translating the model

In this section, we present three new Constraint Programming models that solve the Activity-Based Scheduling problem defined in Pougala et al. (2021). The problem is defined identically for all three models, and all models have the same possible objective functions. Table 1 recaps the features and differences between the models.

Model	**Time**	**Activities**	**Implications**	**Sequencing**
MILP	Continuous	Duplicated by mode & location	Encoded as $\leq$	Sequencing indicators z_{ab}
Direct CP	Discrete			
Indexed CP		Mode & location separate dvars, travel times using ELEMENT	Encoded as $\Rightarrow$	
Interval CP				Interval variables & NOOVERLAP

Table 1: Comparison of features between ABM models

3.1 Problem definition

We present a constraint program model that is as close as possible to the mixed-integer linear programming model presented in Pougala et al. (2021).

Time, as in Pougala et al. (2021), starts at $t = 0$ and finishes at $t = T$, where $T \in \mathbb{N}$ represents the end of the day. Since constraint programming only allows decision variables to take integer values, time is discretised to a finite number of time steps. The set of all time steps is denoted by $\mathcal{T} = \{0, 1, ..., T\}$.

Space is expressed identically as a discrete and finite set of locations L, indexed by l. The location $l_0 = 0$ is referred to as home. For this model, we assume all schedules must start and finish at this location.

The set of modes considered by the agent is denoted by M, and is indexed by m. Travel time between two locations l_o and l_d is denoted by $\rho(l_o, l_d, m)$. Travel time is infinite if the destination l_d cannot be reached from the origin l_o using mode m.

The agent considers a discrete and finite set of activities A, indexed by a. We define A_H to be the set of all activities that occur at home. Each activity has:

- a set L_a of possible locations the activity can occur at.
- a desired start time $x_a^* \in \mathcal{T}$.
- a desired duration $\tau_a^* \in \mathcal{T}$.
- a flexibility level for early start f_a^{se}, late start f_a^{sl}, short duration f_a^{ds}, and long duration f_a^{dl}. These set, for each activity $a \in A$, how harshly deviations from the desired start time and duration are penalized.
- a minimum duration $\tau_a^{\min} \in \mathcal{T}$.
- a feasible time interval $[\gamma_a^-, \gamma_a^+]$ during which the activity can occur.

Each model should generate a utility-maximizing schedule respecting the same constraints as the MILP in Pougala et al. (2021). A schedule $\mathcal{S}$ is a sequence of n activities $A = (a_0, ..., a_{n-1})$. Each schedule must start with an activity a_0 called "dawn" and end with an activity a_{n-1} called "dusk". Both of these dummy activities must take place at home. Each activity a in the schedule is associated with an location l_a, a starting time x_a, a duration τ_a, and a transport mode m_a.

3.2 Objective function

The objective function is defined using the same parameters and utility specification as the MILP from Pougala et al. (2021):

$$U_{\mathcal{S}} = U + \sum_{a=0}^{n-1} \left(U_s^1 + V_s^2 + V_s^3 + U_s^4 + V_s^5 \right) \tag{1}$$

This means that in the experimental investigation, the penalties for each flexibility level have the same values. The error terms are sampled using the same distributions. The penalties and error values are sampled as hourly values then discretised into the T time steps used by the CP model.

The deterministic utilities V_s^2, V_s^3, and V_s^5 are modelled identically to Pougala et al. (2021). V_s^2 penalises deviation from the desired start time, V_s^3 penalises deviation from the desired duration, and V_s^5 penalises total travel time. Due to the specificities of our CP Solver, some additional helper variables are created to store the results of expressions such as $\max(0, x_{a_s} - x_{a_s}^+)$. These do not impact the actual model constraint set and are only used to compute the objective function.

The time-independent stochastic utilities U, U_s^1, and U_s^4 are, as in Pougala et al. (2021), represented by a random variable. The generic utility U is sampled from Gumbel$(0, 1)$. The other two utilities are represented as a sum of four error functions for each activity in the schedule:

- The error functions $f_e(v)$ take a decision variable v as input and output an error value.
- The functions f_e are defined, like in the MILP, using a vector e of size n sampled from a standard normal distribution.
- The size of the error vector n depends on the decision variable used: for decision variables $v = w_a$, z_a, x_a, τ_a, we have $n = 2, 2, 4, 6$ respectively.

We express the random activity utility as follows:

$$U_a^1 + U_a^2 = f_{e_w}(w_a) + f_{e_x}(x_a) + f_{e_\tau}(\tau_a) + \sum_{b \in A} f_{e_z}(z_{ab}) \qquad \forall a \in A \tag{2}$$

In order to test model performance under objective functions of varying complexity, three variants of the error functions were evaluated: a zero function, a stepwise function, and a piecewise linear function.

- The zero function $f_e(v) = 0$ amounts to setting U_a^1 and U_a^2 to 0 and using the general error term U as the only source of randomness in the objective function.
- The stepwise function uses the values of the error vector e to define the function steps. We define a set of n equally-spaced points $\{p_i\}_{i=1}^n$ in the domain $\mathcal{D}$ of the decision variable such that $p_n = \max\{\mathcal{D}\}$. We then define f_e as a stepwise function with points $P = \{(p_i, e_i)\}_{i=1}^n$:

$$f_e(v) = \begin{cases} e_1 & v \leq p_1 \\ e_i & p_i \leq v < p_{i+1} \quad \forall i \in \{1, ..., n\} \\ e_n & p_n \leq v \end{cases}$$

- The piecewise linear function uses the same set of points $\{p_i\}_{i=1}^n$ as the stepwise function and is defined as follows:

$$f_e(v) = \begin{cases} e_1 & v \leq p_1 \\ e_i + \frac{e_{i+1} - e_i}{p_{i+1} - p_i} \cdot (v - p_i) & p_i \leq v < p_{i+1} \quad \forall i \in \{1, ..., n\} \\ e_n & p_n \leq v \end{cases}$$

Figure 1 illustrates the difference between a stepwise and a piecewise linear function.

Special care must be taken when representing stepwise and piecewise linear functions in CP. Indeed, only integer decision variables are allowed in CP. Both variants of the error functions are represented as a sum of expressions $f_e(v) = \sum_i c_i \cdot E_i$, with some constant $c_i \in \mathbb{R}$ and an integer decision variable E_i. In the stepwise version, each variable E_i represents the value of one step in the stepwise function. For the piecewise version, each E_i represents the value of one slope in the piecewise linear function. Consider a step E_i for the stepwise function $f_e(v)$, with input decision variable $x \in \mathcal{D}$. We can represent each step in the stepwise function as two constraints, where $\mathcal{I}[p_i \leq x < p_{i+1}]$ is a boolean variable indicating if x is in the given step:

$$\mathcal{I}[p_i \leq x < p_{i+1}] \Rightarrow (E_i = 1) \tag{3}$$

$$\neg\mathcal{I}[p_i \leq x < p_{i+1}] \Rightarrow (E_i = 0) \tag{4}$$

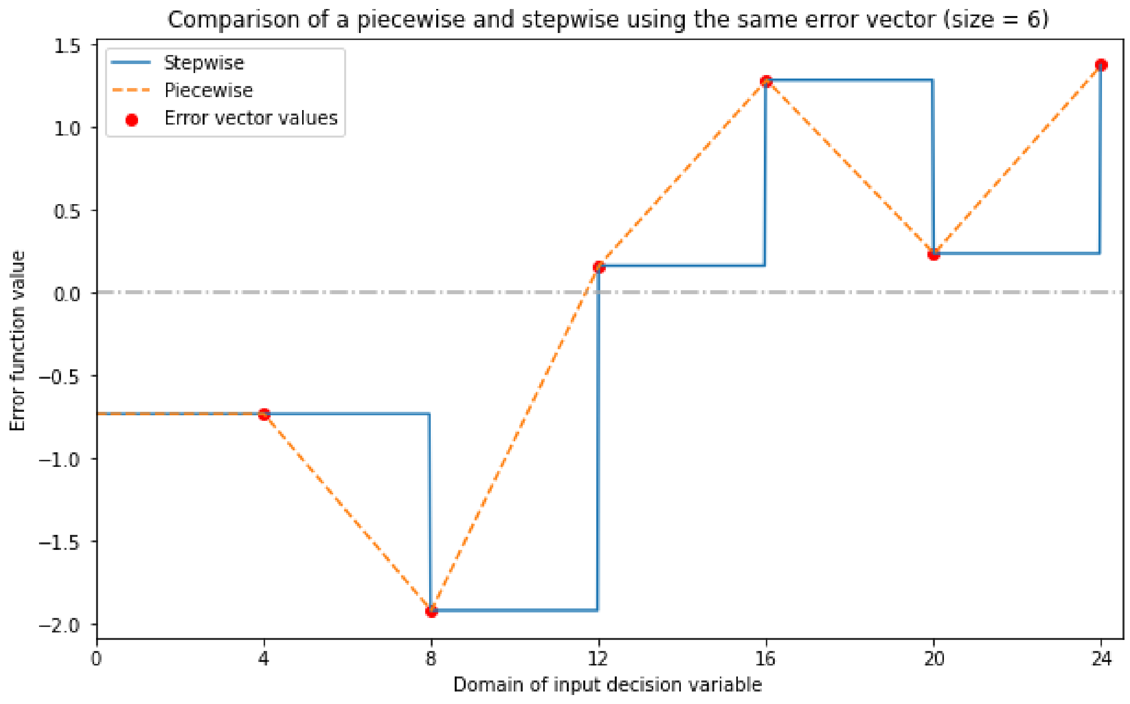

Figure 1: *Difference between a piecewise error function and a stepwise error function using the same input variable and error vector.*

The stepwise error function will then be written as:

$$f_e(v) = \sum_{i=1}^{n} e_i \cdot E_i \tag{5}$$

The corresponding slopes in the piecewise linear function are split into their constant part $E_{i,c}$ and their linear part $E_{i,s}$:

$$\mathcal{I}[p_i \leq x < p_{i+1}] \Rightarrow (E_{i,c} = 1) \tag{6}$$
$$\mathcal{I}[p_i \leq x < p_{i+1}] \Rightarrow (E_{i,s} = (x - p_i)) \tag{7}$$
$$\neg\mathcal{I}[p_i \leq x < p_{i+1}] \Rightarrow (E_{i,c} = 0) \tag{8}$$
$$\neg\mathcal{I}[p_i \leq x < p_{i+1}] \Rightarrow (E_{i,s} = 0) \tag{9}$$

The piecewise linear error function can be expressed as:

$$f_e(v) = \sum_{i=1}^{n} e_i \cdot E_{i,c} + \frac{e_{i+1} - e_i}{p_{i+1} - p_i} \cdot E_{i,s} \tag{10}$$

3.3 Direct translation

This model utilises the same activity duplication mechanism as the MILP from Pougala et al. (2021). This means that, for each activity, we enumerate the combinations of locations and transport modes associated with the activity. The duplicates for each activity form a group G_a. The model considers each duplicate a separate activity, with the exception that no two activities from the same group can be part of schedule at the same time.

We define the following decision variables:

- $\omega_a \in \{0, 1\}$: binary variable that indicates if activity $a \in A$ occurs in the schedule.
- $z_{ab} \in \{0, 1\}$: binary variable that indicates if activity $a \in A$ occurs right after activity $b \in A$ in the schedule.
- $x_a \in \mathcal{T}$: start time of activity $a \in A$.
- $\tau_a \in \mathcal{T}$: duration of activity $a \in A$.
- $\alpha_a^{\text{car}} \in \{0, 1\}$: binary variable to indicate if the agent uses the car to move from activity $a \in A$ to the subsequent activity.

We can write the constraints of the direct translation model as follows:

$$\sum_{a \in A} \left(\tau_a + \sum_{b \in A} z_{ab} \rho(l_a, l_b, m_a) \right) = T \tag{11}$$

$$\sum_{a \in G_{\text{dawn}}} \omega_a = 1 \tag{12}$$

$$\sum_{a \in G_{\text{dusk}}} \omega_a = 1 \tag{13}$$

$$\omega_a \Rightarrow (\tau_a^{\text{min}} \leq \tau_a) \qquad \forall a \in A \tag{14}$$

$$\omega_a \Rightarrow (\tau_a \leq T) \qquad \forall a \in A \tag{15}$$

$$\neg\omega_a \Rightarrow (\tau_a = 0) \qquad \forall a \in A \tag{16}$$

$$\neg z_{ab} \vee \neg z_{ba} = \textsc{True} \qquad \forall a, b \in A \tag{17}$$

$$z_{ad} = 0 \qquad \forall a \in A, d \in G_{\text{dawn}} \tag{18}$$

$$z_{da} = 0 \qquad \forall a \in A, d \in G_{\text{dusk}} \tag{19}$$

$$\omega_a = \sum_{b \in A} z_{ba} \qquad \forall a \in A \setminus G_{\text{dawn}} \tag{20}$$

$$\omega_a = \sum_{b \in A} z_{ab} \qquad \forall a \in A \setminus G_{\text{dusk}} \tag{21}$$

$$z_{ab} \Rightarrow (x_a + \tau_a + \rho(l_a, l_b, m_a) = x_b) \qquad \forall a, b \in A, a \neq b \tag{22}$$

$$\sum_{b \in G_a} \omega_b \leq 1 \qquad \forall a \in A \tag{23}$$

$$x_a \geq \gamma_a^- \qquad \forall a \in A \tag{24}$$

$$x_a + d_a \leq \gamma_a^+ \qquad \forall a \in A \tag{25}$$

$$\alpha_a^{\text{car}} = 1 \qquad \forall a \in \{a \in A_H \mid m_a = \text{car}\} \tag{26}$$

$$\alpha_a^{\text{car}} = 0 \qquad \forall a \in \{a \in A_H \mid m_a \neq \text{car}\} \tag{27}$$

$$\neg\alpha_a^{\text{car}} \Rightarrow (\omega_a = 0) \qquad \forall a \in \{a \in A \mid m_a = \text{car}\} \tag{28}$$

$$z_{ab} \Rightarrow (\alpha_a^{\text{car}} = \alpha_b^{\text{car}}) \qquad \forall a \in A, b \in A \setminus A_H \tag{29}$$

Equation (11) enforces that the time taken by all activities sums to a whole day. Equations (12) and (13) ensure that at least one variant of dawn, respectively dusk, occurs. Equations (14) and (15) make sure that, if an activity occurs, its duration is between its minimum and maximum duration. Conversely, equation (16) enforces the duration to be zero if an activity doesn't occur. Equations (20) and (21) ensure that, if an activity occurs, then it has a single predecessor and successor. If it doesn't occur, the activity has no predecessor or successor. This excludes the first and last activity in the schedule. Equation (22) enforces time consistency between subsequent activities, taking travel time into account. Equation (23) only allows a single activity to be selected from each duplicate group. Equations (24) and (25) ensure that an activity starts and finishes within its feasible time range. The last four equations enforce mode consistency within a tour away from home. Equation (26) makes the car available for home activities whose mode is driving, and equation (27) prohibits car use for home activities that use a different travel mode. Equation (28) prohibits selecting an activity that uses driving if the car is not available for that activity. Finally, equation (29) makes the car available to an activity away from home only if the car was available for its predecessor.

3.4 Indexed version

This model improves the expressiveness of the direct translation by replacing activity duplication with the array indexing features of constraint programming. Activities are no longer duplicated by mode and location. Instead, mode and location choice for each activity are modeled as separate decision variables. We find the corresponding travel time using the ELEMENT constraint from CP. Given a target variable t, an index variable i, and an array of variables V, ELEMENT enforces the target to take the value of V indexed by i:

$$\textsc{Element}(i, V, t) \iff t = V[i] \tag{30}$$

Travel time for each mode $m \in M$ from all locations $l_a \in L$ to every location $l_b \in L$ is stored in an array D. In order to simplify model definition, this travel time array D of size $|M||L|^2 + 1$ is expressed as a flattened version of the three-dimensional array of travel times indexed by (m, l_a, l_b). We access the travel time from l_o to l_d using mode m as follows, assuming indices are zero-indexed:

$$\rho(l_a, l_b, m) = D[|L|^2 m + |L| l_a + l_b] \tag{31}$$

The very last element of D is set to zero and can never be accessed when indexing using l_a, l_b, and m. We use this element to set the travel time between two activities to zero when these activities do not follow each other.

Thanks to mode choice being a decision variable for every activity, mode consistency on tours away from home can be expressed in a very concise way. Indeed, the choice of mode for activity a refers to the trip from a to its subsequent activity. This means we can ensure mode consistency by enforcing that, for two subsequent activities a and b, if b is not at home, then the mode choice of activity a must be the same as activity b. To remain consistent with the MILP and the direct CP translation, we limit this mode consistency to driving only.

The indexed model defines the following decision variables:

- $w_a \in \{0, 1\}$: binary variable that indicates if activity $a \in A$ occurs in the schedule.
- $z_{ab} \in \{0, 1\}$: binary variable that indicates if activity $a \in A$ occurs right after activity $b \in A$ in the schedule.
- $x_a \in \mathcal{T}$: start time of activity $a \in A$.
- $\tau_a \in \mathcal{T}$: duration of activity $a \in A$.
- $m_a \in M$: mode choice for activity $a \in A$.
- $l_a \in L_a$: location choice among possible locations for activity $a \in A$.
- $t_{ab} \in \mathcal{T}$: travel time from activity $a \in A$ to activity $b \in B$.

- $i_{ab} \in \{0, |M||L|^2\}$: the index in the travel time array D for pair of activities $a, b \in A$
- $d_a \in \{0, 1\}$: binary variable indicating if the agent chooses to use the car for $a \in A$.

We can write the constraints of the indexed model like so:

$$\sum_{a \in A} \left(\tau_a + \sum_{b \in A} t_{ab} \right) = T \tag{32}$$

$$\omega_{\text{dawn}} = 1 \tag{33}$$

$$\omega_{\text{dusk}} = 1 \tag{34}$$

$$\omega_a \Rightarrow (\tau_a^{\text{min}} \leq \tau_a) \qquad \forall a \in A \tag{35}$$

$$\omega_a \Rightarrow (\tau_a \leq T) \qquad \forall a \in A \tag{36}$$

$$\neg\omega_a \Rightarrow (\tau_a = 0) \qquad \forall a \in A \tag{37}$$

$$\neg z_{ab} \vee \neg z_{ba} = \textsc{True} \qquad \forall a, b \in A \tag{38}$$

$$z_{a,\text{dawn}} = 0 \qquad \forall a \in A \tag{39}$$

$$z_{\text{dusk},a} = 0 \qquad \forall a \in A \tag{40}$$

$$\omega_a = \sum_{b \in A} z_{ba} \qquad \forall a \in A, a \neq \text{dawn} \tag{41}$$

$$\omega_a = \sum_{b \in A} z_{ab} \qquad \forall a \in A, a \neq \text{dusk} \tag{42}$$

$$z_{ab} \Rightarrow (x_a + \tau_a + t_{ab} = x_b) \qquad \forall a, b \in A, a \neq b \tag{43}$$

$$x_a \geq \gamma_a^- \qquad \forall a \in A \tag{44}$$

$$x_a + d_a \leq \gamma_a^+ \qquad \forall a \in A \tag{45}$$

$$\neg z_{ab} \Rightarrow (i_{ab} = |M||L|^2) \qquad \forall a, b \in A \tag{46}$$

$$z_{ab} \Rightarrow (i_{ab} = |L|^2 m_a + |L| l_a + l_b) \qquad \forall a, b \in A \tag{47}$$

$$\textsc{Element}(i_{ab}, D, t_{ab}) \qquad \forall a, b \in A \tag{48}$$

$$d_a \Rightarrow (m_a = m^{\text{driving}}) \qquad \forall a \in A \tag{49}$$

$$\neg d_a \Rightarrow (m_a \neq m^{\text{driving}}) \qquad \forall a \in A \tag{50}$$

$$(z_{ab} \wedge d_a) \Rightarrow (d_b = 1) \qquad \forall a \in A, b \in A \setminus A_H \tag{51}$$

$$(z_{ab} \wedge \neg d_a) \Rightarrow (d_b = 0) \qquad \forall a \in A, b \in A \setminus A_H \tag{52}$$

Most equations remain identical to the direct translation and perform the same function. Equations (33), (34), (39), (40), (41), and (42) are simplified thanks to the lack of duplicate activities: only a single copy of dusk and dawn exist. Equations (32) and (43) use the travel time decision variable t_{ab} instead of the constant values. Equations (46), (47), and (48) enforce travel time computation as described above. Mode consistency is enforced using equations (49) to (52).

3.5 Interval version

Constraint programming allows the definition of interval variables. These variables $I(x, \tau, y)$ represent a time interval, starting at x and ending at y, with a duration τ. They are treated as both a variable than can be used as input to other constraints, and as a constraint in themselves, enforcing:

$$I(x, \tau, y) \iff x + \tau = y \tag{53}$$

Interval variables can be used in global constraints such as NOOVERLAP$(I_1, ..., I_k)$, which enforces that no interval in $I_1, ..., I_k$ can overlap in time.

Furthermore, we can define optional interval variables $I(x, \tau, y, \omega)$ that take an additional boolean variable. These interval variables only enforce the time consistency constraint if the boolean variable is true:

$$I(x, \tau, y, \omega) \iff (\omega \Rightarrow x + \tau = y) \tag{54}$$

These optional interval variables are taken into account in global constraints such as NOOVERLAP. They are counted only if the indicator ω is true.

The interval-based model is based on the indexed version, includes the same decision variables. We add the following decision variables:

- $\Theta_a \in \mathcal{T}$: a variable representing the sum of duration and travel time for activity $a \in A$.
- $y_a \in \mathcal{T}$: a variable representing the end time of an activity $a \in A$.
- $I_a(x_a, \Theta_a, y_a, \omega_a)$: an optional interval variable representing the time span of an activity $a \in A$.

The interval model keeps all the constraints from the indexed model, but adds the following additional constraints:

$$\Theta_a = \tau_a + \sum_{b \in A} t_{ab} \qquad \forall a \in A \tag{55}$$

$$y_a = x_a + \Theta_a \qquad \forall a \in A \tag{56}$$

$$\text{NOOVERLAP}\{I_a : a \in A\} \tag{57}$$

Constraints (55) and (56) define the inputs to the interval variable, and (57) ensures that activity time spans do not overlap. Note that equation (57) is redundant due to equation (43). This is because we need to retain some sequencing information in order to force a subsequent activity to start right after the end of the current activity's travel time. If we relax this assumption and remove equation (43), the interval model produces otherwise valid schedules. We kept both constraints for two reasons: the first is to remain consistent with the output of other models, and the second is because we noticed a strong performance drop if we removed equation (43). This slowdown is most likely due to the model having to consider more possible start times for each activity after the strict time-consistency constraint relaxation.

4 Empirical investigation

The aim of this section is to evaluate the models introduced in section 3. We wish to compare performance across the three newly defined models and the original MILP models. We first devise a procedure that can measure model performance using randomly generated activity data. We use this procedure to compare solver runtimes, as the number of activities increases, across all four models. We also investigate the performance impact of the three different objective functions on the CP models.

4.1 Experimental setup

All models in the experiment have the same interface. They take as input a list of activities as defined in section 3.1, and a list of travel times between start and end locations based on travel mode. The models return the solved schedule and the time taken to find the solved schedule.

The evaluation procedure is split into two parts: data generation and model evaluation. The data generation phase prepares the input data that is given to the models during evaluation. It does so by either loading an existing list of activities and travel times, or by generating a random set of activities and travel times. In the model evaluation phase, we run every model on the selected data one hundred times, sampling new error parameters every time. No maximum limit was placed on the runtime for all models. We record the solve time, objective value, and generated schedule for every iteration of each model. We plot the mean runtimes by model and error function, and compute a corresponding bootstrapped 95% confidence interval.

We generate four datasets of random activities, ranging from RANDOM-2 with two distinct activities, to RANDOM-5 with five distinct activities. A random dataset of size $N \geq 2$ contains dusk, dawn, and $N-2$ other randomly sampled activities. The properties of dusk and dawn such as location are not randomized and are set to pre-defined values. For each remaining activity, we uniformly sample a label for the activity from the set of available labels. We randomly choose $n \sim U[1, 3]$ locations without replacement, and randomly choose each flexibility level f_a^i. The n locations will be the set of locations where the activity can take place. We then sample the following time parameters:

- $\gamma_a^- \sim U(0, T-x)$: the start of the feasible time range, given a minimum feasible duration x.
- $\gamma_a^+ \sim U(\gamma_a^+ + x, T)$: the end of the feasible time range.
- $x_a^* \sim U(\gamma_a^-, \gamma_a^+)$: the desired start time.
- $\tau_a^* \sim U(0, \gamma_a^+ - x_a^*)$: the desired duration

We then uniformly sample a travel time between 5 and 45 minutes for each pair of locations and for each travel mode. Finally, we create a duplicate of each activity for every activity location and travel mode. This forms groups of duplicates G_a, as defined in Pougala et al. (2021).

4.2 Results

All Constraint Programming models generate schedules similar to the ones generated by the MILP. This is illustrated in Figure 2, which plots the schedule from one instantiation of the direct translation CP model on the CLAIRE dataset from Pougala et al. (2021). We can see that the agent starts at home, takes part in several activities at multiple locations, then returns home at the end of the day. We also see that travel time between activities is accounted for.

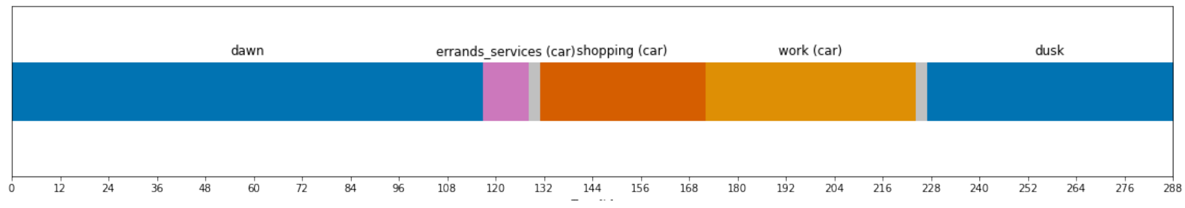

Figure 2: *A daily schedule generated by the direct translation CP using the* CLAIRE *dataset, with 5-minute time steps*

The average runtime of each model increases with the number of activities in the schedule, but the CP models perform significantly better with more complex schedules (Figure 3). The computational time of the MILP model increases rapidly from 0.38s with two activities to 67.6s with five activities, with runtime increasing by approximately one order of magnitude for every additional activity. Performance of the direct CP translation is initially in the same order of magnitude as the MILP, but runtime levels off as the complexity increases. With five distinct activities in the dataset, the direct translation is 20 times better than the MILP. Both the indexed and interval models perform significantly better than both the MILP and the direct translation. Runtime for the indexed model only increases by a factor of 9 from two to five activities, only one order of magnitude. At five activities, the indexed model performs 750 times faster than the MILP and 36 times faster than the direct CP translation. In all performance measurements, the interval model is 1.5 to 2 times slower than the indexed model.

The effect of different error functions on each CP model can be seen in Figure 4. We notice that the performance between all three models is almost identical when using the general error term and the stepwise error function. However, when using the piecewise error function, we observe a large difference between the direct translation (2.32s) and the indexed version (0.04s).

4.3 Interpretation

This experimental framework shows that the CP models can generate adequate solutions to our scheduling problem, while significantly outperforming the runtime of the MILP. In particular, we find that runtime does not increase in the same manner when the number of activities in the schedule increases. This could be due to Constraint Programming handling many small constraints better than Mixed-Integer Linear Programming, as those are common in scheduling problems. However, the performance improvement could also be due to a better optimization of the CP-SAT solver compared to CPLEX, the solver used to solve the MILP model. It could also be explained in part by the discretization of time for CP, which might reduce the size of the

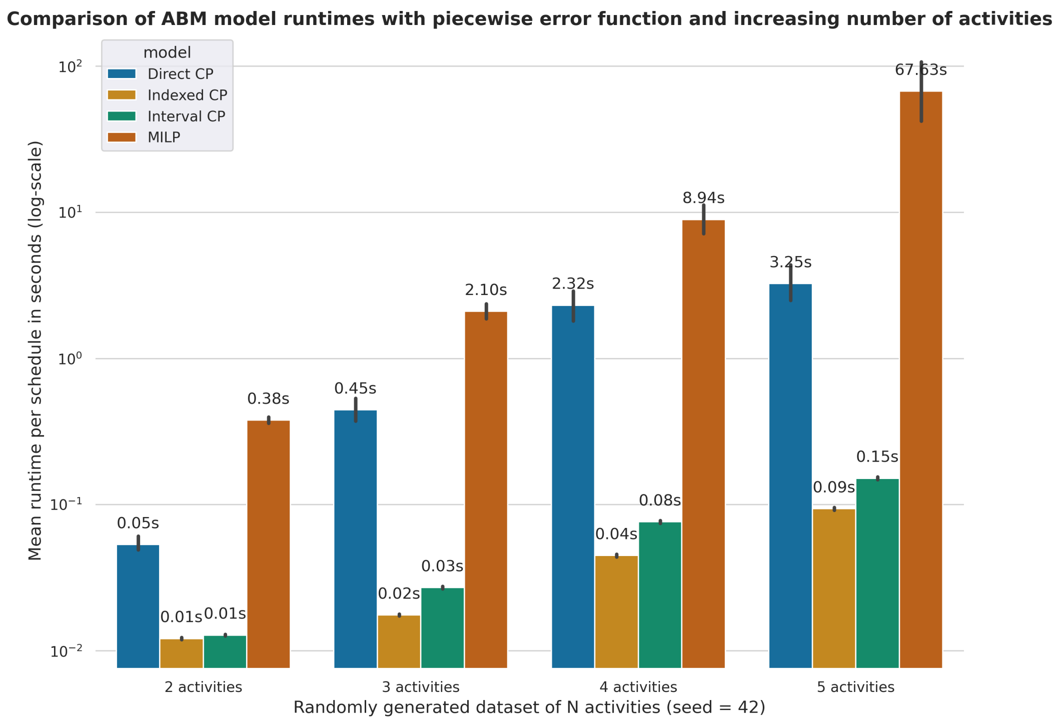

Figure 3: *Comparison of the average runtime per schedule, over 100 iterations of all four ABM models, with an increasing number of activities. All models use the piecewise error function.*

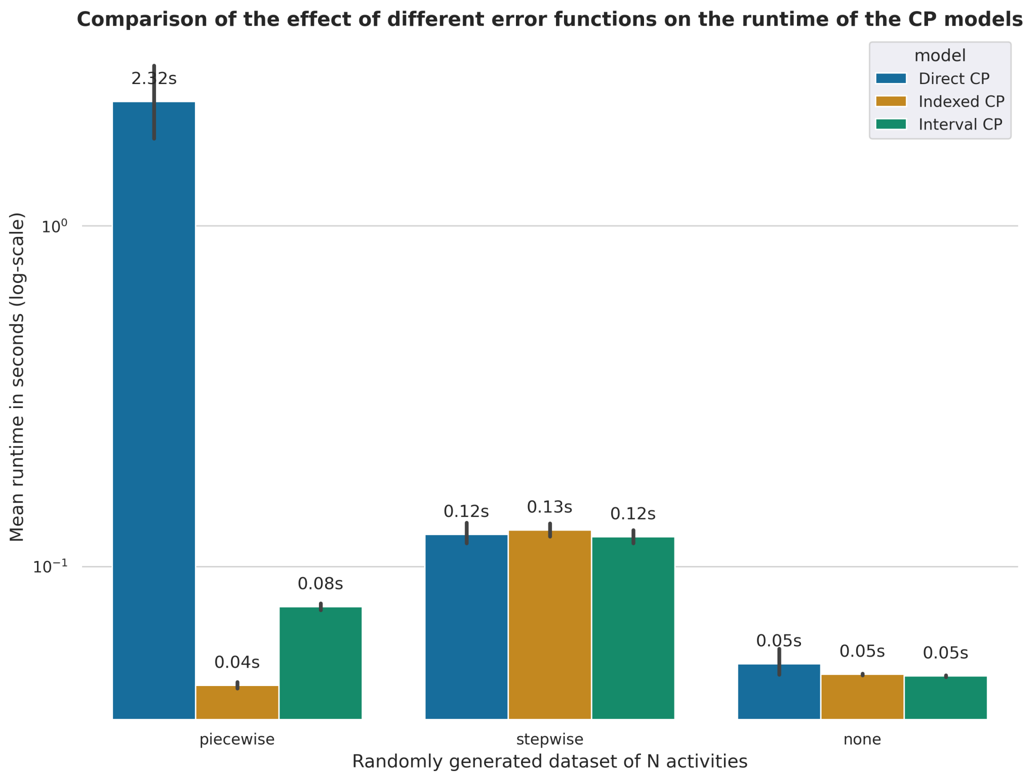

Figure 4: *Comparison of the effect of different error functions on the average per-schedule runtime of the CP models*

decision space for the model. More investigation is needed to determine exactly which factors explain this improvement in performance.

We find that the indexed model performs the best, closely followed by the interval model, with the direct translation a distant third. This could be explained by the more expressive structure offered by the indexed model. Disregarding the activity duplication mechanism in favour of array indexing might make the link between the activity and its different modes and locations clearer to the solver. The solver could use this to make more informed optimization choices. This seems to be corroborated by the performance difference between the direct and indexed models on the piecewise error function. The indexed model performs much better, which could be due to the piecewise error function being simpler to reason about for a model with more structure. The fact that this is not the case for the other two error functions is unclear, but might be due to the decreased interpretability of the error functions.

Compared to the indexed model, we can clearly see that the addition of interval variables does not improve performance. This is due to the two models being almost identical in structure, except for the NoOverlap constraint in the interval model. This constraint only increases model complexity without helping the solver find a solution faster. We can determine that interval variables are not particularly relevant for this scheduling problem: interval variables as defined by CP-SAT might be more useful for job-shop scheduling problems.

5 Conclusion and future work

This research project shows how to define an econometric activity-based scheduling model using Constraint Programming. Basing our models on the existing Mixed-Integer Linear Program from Pougala et al. (2021), we explore how to translate all the aspects of such a model: parameter space, objective functions and constraints. We define a direct translation that generates similar schedules, with minimal changes to the original framework and better computational performance. We then improve on the direct translation and introduce two new models: an indexed model and a interval model. These depart from the MILP formulation by introducing array indexing for location and mode choice for the former, and interval variables for the latter. We measure performance on randomly generated datasets of increasing complexity, and find that not only does CP in general perform better than MILP, the more expressive CP models such as the indexed model outperform the direct CP by one order of magnitude, and the MILP by two orders of magnitude.

We demonstrated that Constraint Programming is particularly well suited to the problem of activity-based scheduling. We also found that performance and model expressiveness can be significantly improved by using features specific to CP, such as logical implication constraints and array indexing using the Element global constraint. However, not all CP features bring these benefits: we found that interval variables and the NoOverlap constraint do not improve performance, nor do they make the model more understandable.

We can identify two key limitations of our study. Firstly, it is currently unclear which factors explain the performance improvements offered by CP: further examination is need to understand exactly which factors influence model runtimes. Secondly, this study does not compare the quality of the solutions generated by the different models. We think that devising a procedure to compare generated schedule quality, for example by using the objective value and controlling for the error parameters, is an interesting direction for further studies.

References

Hooker, J. N. (2002). Logic, Optimization, and Constraint Programming, *INFORMS Journal on Computing* **14**(4): 295–321.
URL: *http://pubsonline.informs.org/doi/abs/10.1287/ijoc.14.4.295.2828*

Naderia, B., Ruizb, R. and Roshanaeic, V. (n.d.). Mixed-integer programming versus constraint programming for shop scheduling problems: New results and outlook.

Pape, C. L. (2005). Constraint-Based Scheduling : A Tutorial.

Perron, L. and Furnon, V. (n.d.). Or-tools.
URL: *https://developers.google.com/optimization/*

Pougala, J., Hillel, T. and Bierlaire, M. (2021). Capturing trade-offs between daily scheduling choices, *Technical Report TRANSP-OR 210101*, Transport and Mobility Laboratory, Ecole Polytechnique Fédérale de Lausanne.

Refanidis, I. and Yorke-Smith, N. (2010). A constraint-based approach to scheduling an individual's activities, *ACM Transactions on Intelligent Systems and Technology* **1**(2): 1–32.
URL: *https://dl.acm.org/doi/10.1145/1869397.1869401*

Rodriguez, J. (2007). A constraint programming model for real-time train scheduling at junctions, *Transportation Research Part B: Methodological* **41**(2): 231–245.
URL: *https://linkinghub.elsevier.com/retrieve/pii/S0191261506000233*

Unsal, O. and Oguz, C. (2013). Constraint programming approach to quay crane scheduling problem, *Transportation Research Part E: Logistics and Transportation Review* **59**: 108–122.
URL: *https://linkinghub.elsevier.com/retrieve/pii/S1366554513001543*

Weil, G., Heus, K., Francois, P. and Poujade, M. (1995). Constraint programming for nurse scheduling, *IEEE Engineering in Medicine and Biology Magazine* **14**(4): 417–422.
URL: *http://ieeexplore.ieee.org/document/395324/*