National Defence | Défense nationale

DEFENCE RESEARCH AND DEVELOPMENT CANADA (DRDC)

RECHERCHE ET DEVELOPPEMENT POUR LA DÉFENSE CANADA (RDDC)

Interim development progress on a mathematical programming model for Northern Operational Support Hub location and facility selection

The mass evacuation scenario

D. G. Hunter
DRDC – Centre for Operational Research and Analysis

Defence Research and Development Canada
Reference Document
DRDC-RDDC-2023-D069
May 2023

Canada

IMPORTANT INFORMATIVE STATEMENTS

Abstract

The Canadian Joint Operations Command Operational Research and Analysis team has been tasked with supporting the Northern Operational Support Hub facility location and selection process. To do this, the team is developing a series of optimization models representing the logistical aspects of the implementation of domestic contingency plans (CONPLANs) in the Canadian North. This Reference Document contains the development to date on a mathematical specification of the first scenario, a CONPLAN LENTUS response to a major maritime disaster involving a cruise ship. The framework constructed permits the evaluation of the operational value of existing and hypothetical infrastructure, such that a globally optimal set of infrastructure investments may be selected within a fixed budget.

Significance for defence and security

The model framework created is flexible and adaptable to many types of operations. It has the potential to be useful not only for the original infrastructure investment decision problem, but also for operational logistic planning.

Résumé

L'équipe d'analyse et de recherche opérationnelle du Commandement des opérations interarmées du Canada a été chargée de soutenir le processus de sélection et d'implantation du Carrefour de soutien opérationnel septentrional. Pour ce faire, l'équipe développe une série de modèles d'optimisation représentant les aspects logistiques de la mise en œuvre des plans de circonstance (CONPLANs) nationaux dans le Nord canadien. Ce document de référence décrit le développement actuel d'une spécification mathématique du premier scénario, une réponse CONPLAN LENTUS à une catastrophe maritime majeure impliquant un navire de croisière. Le cadre élaboré permet d'évaluer la valeur opérationnelle des infrastructures existantes et hypothétiques, de sorte qu'un ensemble optimal d'investissements dans ces infrastructures peut être sélectionné dans les limites d'un budget fixe.

Importance pour la défense et la sécurité

Le cadre de modèle créé est flexible et adaptable à de nombreux types d'opérations. Il a le potentiel d'être utile non seulement pour le problème original de décision d'investissement dans l'infrastructure, mais aussi pour la planification logistique opérationnelle.

Table of contents

List of figures

List of tables

Acknowledgements

The author would like to thank Martin Laskowski, Tom Guardino, and Dr. Robert Fourer, of AMPL Optimization, Inc., for much technical support and useful modelling consultations.

1 Introduction

The Canadian Joint Operations Command (CJOC) Operational Research and Analysis (OR&A) team has been tasked with supporting the Northern Operational Support Hub facility location and selection process. To do this, the team is developing a series of optimization models representing the logistical aspects of implement domestic contingency plans (CONPLANs) in the Canadian North. This work is the successor candidate to the model described in [1] which was created for the Northern Infrastructure Study [2], and uses a version of the CONPLAN LENTUS scenario described there as a starting point.

This Reference Document captures a snapshot of the work to date on a mathematical programming model of a maritime mass evacuation operation. This model combines a vehicle-routing problem, multiple knapsack problems, and a scheduling problem. As such it is complicated to describe and debug. The version described here creates a basic framework for future addition of desired model features.

1.1 Model concept

This model describes a mass evacuation operation from a cruise ship in the Arctic. The number of persons to be evacuated (passengers and crew, hereafter referred to as *evacuees*), is assumed to be on the order of 100–1000. It is further assumed that it was necessary to evacuate the cruise ship by life boat far from any community, and that the evacuees are now located on a coastline somewhere in the Canadian Northern area of responsibility (AOR). This point is henceforth referred to as the *event site*. They must be cared for and transported to an ultimate destination in southern Canada via one or more forward operating locations (FOLs). FOLs are all based in northern communities based on the initiating directive for this work. The medical condition of the evacuees changes over time due to environmental conditions, local facilities and the level of care available.

Evacuees are transported by from the event site to an FOL by air or water vehicles, and then from there to points further south by air vehicles. Every place that a vehicle may travel to or from is called a *node*. Nodes are connected by directional links called *arcs*. Vehicles move between the nodes along defined arcs only. They expend fuel in order to do so. At nodes, there are any number of activities that may be required of vehicles, including but not limited to take-off and landing (docking for ships), loading and unloading, refuelling, and maintenance. Some or all of these activities may require the use of certain personnel or infrastructure at a node, as may node occupation itself. The ability to have personnel at a node may itself be limited by infrastructure at the node. Analysis of the effect of the presence or absence of infrastructure on the execution of the mission is the ultimate object of this model.

1.2 Structure of this Document

This Document has three sections, including this introduction. The model presented in Section 2 is fully functional and can be solved using AMPL in conjunction with a mathematical programming solver such as Gurobi® or IBM's CPLEX, and appropriate input data. Section 3 concludes the Document with a summary of planned changes for the coming round of software development.

2 Model definition

In this section, I discuss the definition of the model, beginning with a description of the notation used in the model code and the corresponding mathematical statements in this Document. I then continue by enumerating and describing the set objects, model parameters, model variables and the objective function for the model. The section is concludes with the model constraints.

2.1 Overview

2.1.1 Notation

I use the following naming conventions in the AMPL model files to improve the clarity of statements and equations:

UPPER_CASE: Sets.

lowerCamelCase: Parameters.

UpperCamelCase: Variables.

lowercase: Iterators.

Constraint names may use either lower or upper camel case. Underscores are used to enhance readability but have no significance otherwise.

In the mathematical description that follows, sets are denoted using boldface uppercase letters, such as $\mathbf{T}$. Iterators over sets are usually simply the same letter as the set name but in lowercase italics, e.g., t. Exceptions to this are given in Table 2.

Parameters are written as Greek letters with subscripts and superscripts as needed. Subscripts are always the set indices. Superscripts are descriptive codes and mnemonics. For example, durations are denoted in this document using τ. The model parameter `durationTravel` is written as τ^{t}_{ijv}. It is indexed over the set of arcs $\mathbf{A}$ using (i, j) and over the set of vehicles $\mathbf{V}$ using v. The superscript t distinguishes this duration from the other durations with differing superscripts.

Another example is α^{pp}_{v}, which denotes the vehicle cargo capacity conversion factor of passengers per pallet. It is indexed using v over the set of vehicles $\mathbf{V}$. The superscript *pp* is a mnemonic for passengers per pallets.

Variables are denoted using x, y, or z. Generally speaking, x variables are related to vehicles, y variables to evacuees, and z variables to nodes, but this is not a rule. They have sub- and superscripts and a prescript. The subscripts show the set indices for the variables, as for parameters. The prescript and superscript indicate the subject or purpose of the variable. For instance, ${}^{A}x^{s}_{tvij}$, ${}^{A}x^{o}_{tvij}$, and ${}^{A}x^{e}_{tvij}$ are used to control and track the presence of vehicle v on arc (i, j) during time step t. The prescript A is the distinguishing factor that shows that these variables are concerned with arc occupation. The superscripts s, o, and e respectively indicate that the vehicle started, is on, or ended the arc in this time step.

2.1.2 Set operators

Several specific set operators are used below, usually with respect to the Time set. This set is ordered but contains values that are not adjacent integers, such that variables defined over this set cannot be referenced using arithmetic indices, such as x_{t-1} to reference the value of x from the previous time step. Instead, we take advantage of the properties of ordered sets within AMPL to use the operators in Table 1 for such references:

Table 1: *Operators for relative set indexing.*

Operator	Function
first(**S**)	Selects the first member of ordered set S
last(**S**)	Selects the last member of an ordered set
prev(s)	Selects the member of the ordered set **S** preceding s
next(s)	Selects the member of the ordered set **S** following s

2.2 Sets

The sets defined in the model are listed in Table 2. Each is described in further detail below.

Table 2: *List of sets with corresponding model names.*

Set	Description	Iterator(s)	AMPL Model Name
T	Time [min]	t	`TIME`
N	Nodes	i, j	`NODES`
A	Arcs	i, j	`ARCS`
V	Vehicle enumerators	v	`VEHICLES`
E	Evacuee enumerators	e	`EVACUEES`
M	Medical (triage) states	m	`MEDICAL_STATE`
L	Evacuee locations	l	`LOCATIONS`

2.2.1 Time

The set **T** is an ordered set of consecutive values enumerating the specific points in time measured in minutes from the beginning of the modelled time period. It is loaded from the data file for each model instance. The intervals between times must conform to the defined timestep parameter Δt to avoid problems at runtime.

2.2.2 Nodes

Nodes, set **N**, are physical locations within the model, representing evacuation beach sites, FOLs, other bases and infrastructure sites. Infrastructure can only exist at nodes at this time, and most vehicle activities that are not travel must occur at a node.

2.2.3 Arcs

The arcs of sets **A** represent the connecting paths between the nodes of set **N**. They have associated distance parameters that are used for calculations of travel time and fuel consumption. Arcs are directional: the arc (i, j) originates at node i and terminates at node j. Normally there will be pairs of arcs defining travel in both directions, but this is not necessary.

Arcs may become vehicle-specific in future to permit the inclusion of transportation infrastructure such as highways and railways.

2.2.4 Vehicles

The set of vehicles $\mathbf{V}$ enumerates all vehicles within the model. Vehicle types exists but are not explicitly defined in the model at this time. Each vehicle's parameters must be populated directly in the input file.

2.2.5 Evacuees

The set $\mathbf{E}$ enumerates the evacuees in the model. Evacuees are modelled as individuals rather than as a pool of persons in a particular medical state. This enables the use of the location, medical transition, and medical state variables described below.

2.2.6 Medical states

Set $\mathbf{M}$ enumerates the possible medical states of evacuees in the model. Medical states are currently approximated using a five-state triage model that is mapped onto an integer scale. This is given below in Table 3. The lowest possible value is 1, the largest 5. These correspond to a a colour code and a description of the general health of the person in question. The effects of these states on are central to the evacuation decisions within the model. The states also directly affect the value of the objective function.

Table 3: *List of medical states, with corresponding colour codes and descriptions.*

Medical state	Colour name	Description
1	White	Perfectly healthy, no concerns; ambulatory
2	Green	Minimal injuries or conditions; ambulatory
3	Yellow	Serious injuries or conditions; non-ambulatory
4	Red	Critical injuries or conditions; non-ambulatory
5	Black	Dead

2.2.7 Locations

The set of locations $\mathbf{L}$ is unusual in that it is defined as the union of the sets of nodes $\mathbf{N}$ and vehicles $\mathbf{V}$. This set captures all of the possible places where an evacuee, personnel or other cargo can be located at any given time step, and is used to govern the update of evacuee medical transitions. Because if its definition, it is possible to index over the node or vehicle subsets as needed.

2.3 Parameters

The parameters currently defined in the model are listed in Table 4, along with their corresponding names in the AMPL model. Most are relatively self-explanatory, but the rate of medical state progression does require some additional description. The `medTransitionRate` Π_{ml} describes the rate at which a patient or evacuee becomes less (or more) healthy as a function of time and location. Positive values of Π_{ml} indicate that the evacuee's condition is deteriorating. Negative values of Π_{ml} mean that the evacuee is recovering.

Table 4: *List of parameters with corresponding model names.*

Parameter	Description	AMPL Model Name
ϵ	A very small number	`epsilon`
Δt	Duration of a time step [min]	`timestep`
ϕ_t^{VFR}	Visual flight rule flag for time t	`visualFlightRules`
δ_{ij}	Distance along arc (i, j) [km]	`distance`
ν_i^{Type}	Type of node i: EVENT, FOL, or REAR	`nodeType`
$\nu_i^{Runways}$	Number of runways at node i	`nodeRunways`
$\nu_i^{RunwayType}$	Airfield capability category at node i	`nodeRunwayType`
$\nu_i^{InitFuel}$	Amount of fuel available at node i at first time step [kg]	`nodeInitFuel`
ν_{iv}^{Use}	Flag for ability of vehicle v to operate at node i	`nodeVehCanUse`
β_v^t	Vehicle type—I/O marker only	`vType`
τ_v^{fa}	Time of first availability for vehicle v†	`timeFirstAvail`
β_v^0	Initial vehicle location	`initVehLocation`
τ_v^{LM}	Duration of light maintenance for vehicle v [min]†	`durationLightMaint`
τ_v^{MM}	Duration of medium maintenance for vehicle v [min]†	`durationMedMaint`
τ_v^{HM}	Duration of major (heavy) maintenance for vehicle v [min]†	`durationMajorMaint`
π_v^{MM}	Time between medium maintenance periods for vehicle v [min]	`intervalMedMaint`
ρ_{iv}^{rf}	Maximum refueling rate for vehicle v at node i [kg/min]	`rateRefuel`
ρ_v^{fc}	Fuel consumption of vehicle v [kg/min]	`rateFuelConsumption`
ρ_v	Speed of vehicle v [km/hour]	`speed`
κ_v^f	Fuel capacity of vehicle V [kg]	`capacityFuel`
κ_v^p	Basic capacity for seated passengers by vehicle	`capacityPassenger`
κ_v^c	Cargo capacity of vehicle v [pallets]	`capacityPallets`
α_v^{pp}	Passenger slots exchanged for one pallet of cargo by vehicle	`cfPassengersPerPallet`
α_{vm}^{ms}	Passenger slots used by evacuee medical state and vehicle	`cfPassCapacityUseByMedState`
τ_v^{pf}	Duration of pre-flight checks for vehicle v [min]	`durationPreflight`
τ_v^{dep}	Duration of departure process for vehicle v [min]	`durationDepart`
τ_v^{arr}	Duration of arrival process for vehicle v [min]	`durationArrive`
τ_v^{ld}	Duration to load vehicle v [min]	`durationLoad`
τ_v^{ul}	Duration to unload vehicle v [min]	`durationUnload`
τ_{ijv}^t	Time for vehicle v to travel along arc (i, j)	`durationTravel`
dead	Medical state corresponding to dead	`dead`
μ^{min}	Minimum possible medical state value	`minMedState`
μ^{max}	Maximum possible medical state value	`maxMedState`
λ_e^0	Initial location of evacuee e	`initEvacueeLocation`
π_e^0	Initial medical state progression of evacuee e	`initEvacueeProgression`
μ_{em}^0	Initial medical state flag of evacuee e in state m	`initEvacueeState`
Π_{ml}	Rate of medical state progression in medical state m at location l [min^{-1}]	`medTransitionRate`

† – Parameter not currently used

2.4 Variables

The variables currently defined in the model are listed in Table 5, along with their data types and corresponding names in the AMPL model. Real variables have implicit constraints as shown by their defined ranges in the table.

Table 5: *List of parameters with data types and corresponding model names. Real-valued variables also show the value range with the type.*

Variable	Description	Type	AMPL Model Name
x^n_{tvi}	Indicator of vehicle v presence at a node i at time t	binary	`VehAtNode`
${}^{A}x^s_{tvij}$	Indicator of vehicle v starting move along arc (i, j) at time t	binary	`VehStartArc`
${}^{A}x^o_{tvij}$	Indicator of vehicle v travelling on arc (i, j) at time t	binary	`VehOnArc`
${}^{A}x^e_{tvij}$	Indicator of vehicle v ending move along arc (i, j) at time t	binary	`VehEndArc`
${}^{d}x^s_{tv}$	Indicator of vehicle v starting departure at time t	binary	`StartDeparture`
${}^{d}x^i_{tv}$	Indicator of vehicle v in departure at time t	binary	`InDeparture`
${}^{d}x^e_{tv}$	Indicator of vehicle v ending departure at time t	binary	`EndDeparture`
${}^{a}x^s_{tv}$	Indicator of vehicle v starting arrival at time t	binary	`StartArrival`
${}^{a}x^i_{tv}$	Indicator of vehicle v in arrival at time t	binary	`InArrival`
${}^{a}x^e_{tv}$	Indicator of vehicle v ending arrival at time t	binary	`EndArrival`
${}^{u}x^s_{tv}$	Indicator of vehicle v starting unloading at time t	binary	`StartUnloading`
${}^{u}x^i_{tv}$	Indicator of vehicle v in unloading at time t	binary	`InUnloading`
${}^{u}x^e_{tv}$	Indicator of vehicle v ending unloading at time t	binary	`EndUnloading`
${}^{l}x^s_{tv}$	Indicator of vehicle v starting loading at time t	binary	`StartLoading`
${}^{l}x^i_{tv}$	Indicator of vehicle v in loading at time t	binary	`InLoading`
${}^{l}x^e_{tv}$	Indicator of vehicle v ending loading at time t	binary	`EndLoading`
${}^{p}x^s_{tv}$	Indicator of vehicle v starting preflight at time t	binary	`StartPreflight`
${}^{p}x^i_{tv}$	Indicator of vehicle v in preflight at time t	binary	`InPreflight`
${}^{p}x^e_{tv}$	Indicator of vehicle v ending preflight at time t	binary	`EndPreflight`
${}^{r}x^s_{tv}$	Indicator of vehicle v starting refuelling at time t	binary	`StartRefuelling`
${}^{r}x^i_{tv}$	Indicator of vehicle v in refuelling at time t	binary	`InRefuelling`
${}^{r}x^e_{tv}$	Indicator of vehicle v ending refuelling at time t	binary	`EndRefuelling`
y^f_{tv}	Quantity of fuel carried by vehicle v at the end of time step t	real $[0, \kappa^f_v]$	`VehFuel`
y^r_{tv}	Enabling flag—Vehicle v may refuel at time t	binary	`CanNowRefuel`
z^f_{ti}	Quantity of fuel at node i at the end of time step t	real $[0, \infty]$	`NodeFuel`

Table 5: *(continued)*

Variable	Description	Type	AMPL Model Name
z^{fv}_{itv}	Quantity of fuel transferred from node i to vehicle v during time step t	real $[0,\, \rho^{rf}_{iv}]$	`NodeFuelTransferToVeh`
${}^{e}x^{l}_{etl}$	Indicator of evacuee e presence at location l at time t	binary	`EvacueeLocation`
${}^{e}x^{m}_{etm}$	Indicator of evacuee e in medical state m at time t	binary	`EvacueeState`
${}^{e}x^{t}_{et}$	Evacuee e medical transition value at time t	real $[\mu^{min},\, \mu^{max}]$	`EvacueeMedTransition`
${}^{e}y^{ml}_{etml}$	Indicator of evacuee e presence at location i in medical state m at time t	binary	`EvacueeLocationState`
${}^{e}y^{ld}_{etvi}$	Enabling flag—evacuee e can load onto vehicle v at node i during time at time step t	binary	`EvacueeCanLoad`
${}^{e}y^{ul}_{etvi}$	Enabling flag—evacuee e can unload from vehicle v at node i during time at time step t	binary	`EvacueeCanUnload`

2.5 Objective function

Several objective functions were tested. One set was integer-valued the other real-valued. It was believed that a real-valued objective would help the the solver work more quickly, but this was not borne out in practice. The integer-valued objective function is shown in Equation (1). This function, LivingEvacueesAtFOL, sums the number of evacuees that are located at the FOL for all time steps. This rewards solutions that evacuee people more quickly.

$$\text{LivingEvacueesAtFOL: } \mathbf{maximize} \sum_{\substack{e\in\mathbf{E}\\ t\in\mathbf{T}\\ m\in\mathbf{M}}} {}^{e}y^{ml}_{etml}, \text{ such that } l = \text{FOL}, m \neq \text{dead} \tag{1}$$

For reference, the best real-valued objective function tested was BadHealthOutcomes, shown in Equation (2). In addition to the primary real-valued summation term over ${}^{e}x^{t}_{et}$, it contains two additional terms that are in fact integer-valued. These were added to increase the penalties for allowing evacuees to die and for excess travel time.

Using this objective function, Gurobi was able to solve the linear relaxation of the model quickly. However, regardless of these and other attempts to improve the real-valued objective function, integer solutions for even simple data sets involving one helicopter and two nodes took several days and frequently simply stalled before reaching the proof of optimality stopping condition. This appears to be dependent on the number of time steps in the data set.

$$\text{BadHealthOutcomes: } \mathbf{minimize} \sum_{\substack{e\in\mathbf{E}\\ t=\text{last}(\mathbf{T})}} {}^{e}x^{t}_{et} + 2\sum_{\substack{e'\in\mathbf{E}\\ t'\in\mathbf{T}\\ m\in\mathbf{M}}} {}^{e}x^{m}_{e't'm} + 0.5\sum_{\substack{t''\in\mathbf{T}\\ v\in\mathbf{V}\\ (i,j)\in\mathbf{A}}} {}^{A}x^{o}_{t''vij}, \text{ such that } m = \text{dead} \tag{2}$$

2.6 Constraints

The constraints are grouped into several large and overlapping categories. They have been sorted as much as possible by their primary focus.

2.6.1 Initialization

Constraints (3)–(10) set the initial conditions of variables based on the input data in the relevant parameters.

$$\text{Initialize vehicle locations:} \quad x^n_{tvi} = \begin{cases} 1 \text{ if } i = \beta^0_v, \\ 0 \text{ otherwise,} \end{cases} \quad \forall\, t = \text{first}(\mathbf{T}), v \in \mathbf{V}, i \in \mathbf{N} \tag{3}$$

$$\text{Initialize no vehicles on arcs:} \quad {}^A x^o_{tvij} = 0, \quad \forall\, t = \text{first}(\mathbf{T}), v \in \mathbf{V}, (i,j) \in \mathbf{A} \tag{4}$$

$$\text{Initialize vehicle arrival status:} \quad {}^a x^e_{tv} = 1, \quad \forall\, t = \text{first}(\mathbf{T}), v \in \mathbf{V} \tag{5}$$

$$\text{Initialize vehicle fuel:} \quad y^f_{tv} = \kappa^f_v, \quad \forall\, t = \text{first}(\mathbf{T}), v \in \mathbf{V} \tag{6}$$

$$\text{Initialize node fuel:} \quad z^f_{ti} = \nu^{InitFuel}_i, \quad \forall\, t = \text{first}(\mathbf{T}), i \in \mathbf{N} \tag{7}$$

$$\text{Initialize evacuee locations:} \quad {}^e x^l_{etl} = \begin{cases} 1 \text{ if } l = \lambda^0_e, \\ 0 \text{ otherwise} \end{cases} \quad \forall\, t = \text{first}(\mathbf{T}), e \in \mathbf{E}, l \in \mathbf{L} \tag{8}$$

$$\text{Initialize evacuee state:} \quad {}^e x^m_{etm} = \mu^0_{em}, \quad \forall\, t = \text{first}(\mathbf{T}), e \in \mathbf{E}, m \in \mathbf{M} \tag{9}$$

$$\text{Initialize evacuee medical progression:} \quad {}^e x^t_{et} = \pi^0_e, \quad \forall\, t = \text{first}(\mathbf{T}), e \in \mathbf{E} \tag{10}$$

2.6.2 Vehicle routing

These constraints handle the movement of vehicles between nodes and arcs. The small size of time steps with respect to the distances involved in the model means that most travel between nodes requires the vehicles to remain on the connecting arcs for many time steps.

2.6.2.1 General rules

These constraints restrict a vehicle from travelling during prohibited periods (Constraint (11)) and from occupying nodes prohibited to the vehicle (Constraint (12)). This makes sense given that all vehicles represented to this point are aircraft. When ships, land vehicles or both are added, these constraints will be modified to apply to specific vehicle types only.

Prohibited periods for aircraft may be due to availability of appropriate flight aids for the time and weather, or due to the weather itself. Nodes are prohibited to aircraft based on the airfield capabilities there, primarily the existence of an appropriate runway or landing surface. For example, CC130J Hercules aircraft are prohibited from all event nodes because there event nodes do not have a runway.

One assumption for air vehicles at the present time is that they will operate under visual flight rules (VFR) only. This is based on the lack of appropriate navigational aids at the event site(s) and also at many airfields in the North. These aids are need for take-offs and landings at night and in bad weather. If this is not a concern it can be negated by changing the input data to show that VFR are in effects for all time steps. If some but not all aircraft can use instrument flight rules (IFR), then the model will require additional parameters and constraints to capture this. Another consideration is that the VFR parameter ϕ^{VFR}_t is global and does not capture the potentially large variation in daytime hours between nodes and along arcs in the model.

$$\text{Bound—visual flight rules:} \quad {}^{A}x^{o}_{tvij} \leq \phi^{VFR}_{t}, \forall\, t \in \mathbf{T}, v \in \mathbf{V}, (i,j) \in \mathbf{A} \tag{11}$$

$$\text{Bound—vehicle can use node:} \quad x^{n}_{tvi} \leq \nu^{Use}_{iv}, \quad \forall\, t \in \mathbf{T}, v \in \mathbf{V}, i \in \mathbf{N} \tag{12}$$

2.6.2.2 Constraints pertaining to beginning and ending travel (on arc) status

Constraint (13) is the key constraint that controls the state of vehicles while travelling along an arc. Three variables are used for this: ${}^{A}x^{s}_{tvij}$, which is equal to 1 only for the first time step of a transit; ${}^{A}x^{e}_{tvij}$ which equal 1 only on the last time step of the transit; and ${}^{A}x^{o}_{tvij}$, which must equal 1 for every time step of the transit. Constraint (13) requires that either exactly two of these be equal to 1, or all of them must be equal to 0. There is an implicit assumption that every arc will require at least two time steps to transit built into this assumption, as ${}^{A}x^{s}_{tvij}$ and ${}^{A}x^{e}_{tvij}$ can never be equal to 1 during the same time step. An example is shown in Figure 1. In this figure, the coloured blocks show when a variable is equal to 1; Time when the variable is equal to 0 are shown as blank space. Here we see a vehicle which starts travelling along an arc at time step t and arrives at its destination at time step $t + 4\Delta t$. The vehicle is on-arc for all off the intervening time steps, start and end included. This diagram corresponds to ${}^{A}x^{s}_{tvij} = 1$, ${}^{A}x^{e}_{t+4\Delta t,vij} = 1$, and ${}^{A}x^{s}_{t'vij} = 1\, \forall\, t' \in \{t, \ldots, t + 4\Delta t\}$. By definition of Constraint (13), these variables must be equal to zero for all other time steps between t and $t + 4\Delta t$ inclusive.

$$\text{Vehicle on-arc logic: } {}^{A}x^{s}_{tvij} - {}^{A}x^{e}_{tvij} = {}^{A}x^{o}_{tvij} - \begin{cases} {}^{A}x^{o}_{\text{prev}(t),vij} \text{ if } t > \text{first}(\mathbf{T}), \\ 0 \text{ if } t = \text{first}(\mathbf{T}), \end{cases} \quad \forall\, t \in \mathbf{T}, v \in \mathbf{V}, (i,j) \in \mathbf{A} \tag{13}$$

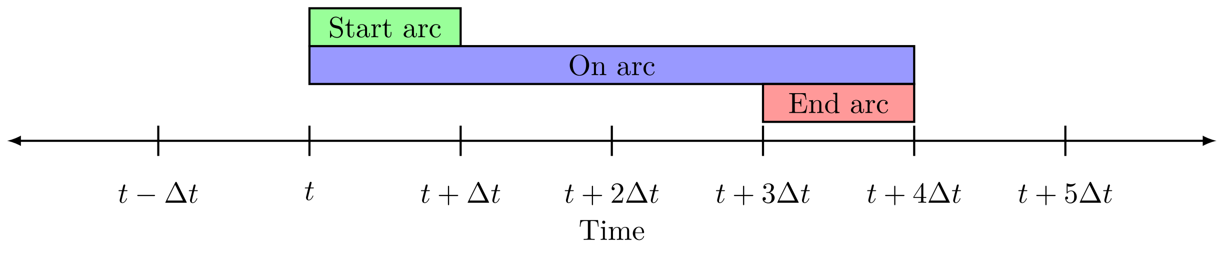

Figure 1: *Example of vehicle on-arc logic showing time steps when vehicle start arc, overlap. All variables are equal to zero for time steps without illustration.*

Constraint (14) ensures that vehicles remain on an arc for at least as long as the minimum travel time defined by parameter τ^{t}_{ijv}, relative to a given departure time t_d. Because of the incentive to evacuate as quickly as possible, and the fact that each time step on arc consumes fuel, there is no need to place an upper bound on on-arc durations.

$$\text{Vehicle on-arc duration: } {}^{A}x^{s}_{t_d vij} \leq {}^{A}x^{o}_{tvij}, \forall\, t \in t_d \ldots t_d + \tau^{t}_{ijv} : t < \text{last}(\mathbf{T}), v \in \mathbf{V}, (i,j) \in \mathbf{A} \tag{14}$$

This sort of paired on-off logic and duration constraints show up again and again in this model as the basis of time allocation. Using this structure allows the simple addition of space and capacity constraints on vehicle movements and operations.

Constraints (15)–(20) prevent the solver from doing things that are impossible or nonsensical in the real world, or that create model problems with vehicles. Constraint (15) ensures that all trips that are started are finished. Constraints (16) prevents trips from being started if they would be completed after the end of the modelled time period. Constraint (17) requires that a vehicle always occupy exactly one node or arc.[1] Constraints (18)–(20) link the vehicles beginnings and ends of arcs to the nodes from which they originated and prevent travel between nodes without using the connecting arc.[2]

Complete all arcs:
$$\sum_{t\in\mathbf{T}} {}^{A}x^{s}_{tvij} - \sum_{t\in\mathbf{T}} {}^{A}x^{e}_{tvij} = 0, \qquad \forall\, v \in \mathbf{V}, (i,j) \in \mathbf{A} \tag{15}$$

No trips outside time window:
$${}^{A}x^{s}_{t_d vij} = 0, \qquad \forall\, t_d \in \text{last}(\mathbf{T}) - \tau^{t}_{ijv} \dots \text{last}(\mathbf{T}), \; v \in \mathbf{V}, (i,j) \in \mathbf{A} \tag{16}$$

Vehicles occupy exactly one location:
$$\sum_{i\in\mathbf{N}} x^{n}_{tvi} + \sum_{(i,j)\in\mathbf{A}} {}^{A}x^{o}_{tvij} = 1, \qquad \forall\, t \in \mathbf{T}, v \in \mathbf{V} \tag{17}$$

Link arc end to node:
$${}^{A}x^{e}_{\text{prev}(t),vij} \le x^{n}_{tvj}, \qquad \forall\, t \in \mathbf{T} : t > \text{first}(\mathbf{T}), \; v \in \mathbf{V}, (i,j) \in \mathbf{A} \tag{18}$$

No teleportation between nodes 1:
$$x^{n}_{tvi} + \sum_{(i',j)\in\mathbf{A}:i'=i} {}^{A}x^{s}_{tvi'j} \ge x^{n}_{\text{prev}(t),vi}, \quad \forall\, t \in \mathbf{T} : t > \text{first}(\mathbf{T}), \; v \in \mathbf{V}, i \in \mathbf{N} \tag{19}$$

No teleportation between nodes 2:
$$x^{n}_{tvi} + \sum_{j\in\mathbf{N}:j\ne i} x^{n}_{\text{prev}(t),vj} \le 1, \qquad \forall\, t \in \mathbf{T} : t > \text{first}(\mathbf{T}), \; v \in \mathbf{V}, i \in \mathbf{N} \tag{20}$$

2.6.3 Arrival and departure

Arrivals and departures are processes with durations that always occur at a node. These processes represent landing and take off for aircraft, and docking and sailing for ships. The states themselves are not specific to a particular node and so must be linked to node occupation in the current time step and arc occupation in the previous time step. This is done via Constraints (21)–(22) for arrivals and by Constraints (23)–(24) for departures. Constraints (25)–(28) set the start, end and in-process statuses for arrivals and departure. These replicate the logic used for the same purpose for on-arc status in Constraints (13) and (14). The same logic is used throughout the model for every process which requires a period of time to complete.

Link arrival to node:
$${}^{a}x^{i}_{tv} \le \sum_{i\in\mathbf{N}} x^{n}_{tvi}, \qquad \forall\, t \in \mathbf{T} : t < \text{last}(\mathbf{T}), v \in \mathbf{V} \tag{21}$$

Link arrival to arc:
$${}^{a}x^{s}_{tv} \le \sum_{(i,j)\in\mathbf{A}} {}^{A}x^{e}_{\text{prev}(t),vij}, \qquad \forall\, t \in \mathbf{T} : t > \text{first}(\mathbf{T}), v \in \mathbf{V} \tag{22}$$

Link departure to node:
$${}^{d}x^{i}_{tv} \le \sum_{i\in\mathbf{N}} x^{n}_{tvi}, \qquad \forall\, t \in \mathbf{T} : t < \text{last}(\mathbf{T}), v \in \mathbf{V} \tag{23}$$

Link departure to arc:
$${}^{d}x^{e}_{\text{prev}(t),v} \le \sum_{(i,j)\in\mathbf{A}} {}^{A}x^{s}_{tvij}, \qquad \forall\, t \in \mathbf{T} : t > \text{first}(\mathbf{T}), v \in \mathbf{V} \tag{24}$$

Arrival logic:
$${}^{a}x^{s}_{tv} - {}^{a}x^{e}_{tv} = \begin{cases} {}^{a}x^{i}_{\text{prev}(t),v} \text{ if } t > \text{first}(\mathbf{T}), \\ 0 \text{ if } t = \text{first}(\mathbf{T}), \end{cases} \qquad \forall\, t \in \mathbf{T}, v \in \mathbf{V} \tag{25}$$

Arrival duration:
$${}^{a}x^{s}_{t_d v} \le {}^{a}x^{i}_{tv} \qquad \forall\; t \in t_d \dots t_d + \tau^{arr}_{v} : t < \text{last}(\mathbf{T}),$$

[1] There is no limit to total number of vehicles on an arc or at a node at this time, though this could certainly be added. A limit on total node occupancy (maximum on ground [MOG]) by aircraft is planned for the next round of model development.
[2] The need for Constraint (16) is no longer clear. Removal will be tested in the next model iteration.

$$t_d > \text{first}(\mathbf{T}), v \in \mathbf{V} \tag{26}$$

Departure logic: $${}^{d}x^{s}_{tv} - {}^{d}x^{e}_{tv} = \begin{cases} {}^{d}x^{i}_{\text{prev}(t),v} \text{ if } t > \text{first}(\mathbf{T}), \\ 0 \text{ if } t = \text{first}(\mathbf{T}), \end{cases} \quad \forall\, t \in \mathbf{T}, v \in \mathbf{V} \tag{27}$$

Departure duration: $${}^{d}x^{s}_{t_d v} \leq {}^{d}x^{i}_{tv} \quad \forall\, t \in t_d \dots t_d + \tau^{dep}_{v} : t < \text{last}(\mathbf{T}), \; t_d > \text{first}(\mathbf{T}), v \in \mathbf{V} \tag{28}$$

Constraints (29)–(33) close loopholes that allowed arrivals and departure periods to fall partly outside the model's time boundaries, or other strange behaviour.

Bound arrival time: $${}^{a}x^{s}_{tv} = 0, \forall\, t \in \text{last}(\mathbf{T}) \dots \text{last}(\mathbf{T}) - \tau^{arr}_{v} + 1 : t < \text{last}(\mathbf{T}), \; v \in \mathbf{V} \tag{29}$$

Bound departure time a: $${}^{d}x^{e}_{tv} = 0, \forall\, t \in \text{first}(\mathbf{T}) \dots \text{first}(\mathbf{T}) + \tau^{dep}_{v} - 1 : t > \text{first}(\mathbf{T}), \; v \in \mathbf{V} \tag{30}$$

Bound departure time b: $${}^{d}x^{s}_{tv} = 0, \forall\, t \in \text{last}(\mathbf{T}) \dots \text{last}(\mathbf{T}) - \tau^{dep}_{v} + 1 : t < \text{last}(\mathbf{T}), \; v \in \mathbf{V} \tag{31}$$

Control arrival starts: $${}^{a}x^{s}_{tv} + {}^{a}x^{s}_{\text{prev}(t),v} \leq 1, \forall t \in \mathbf{T} : t > \text{first}(\mathbf{T}), v \in \mathbf{V} \tag{32}$$

Control departure starts: $${}^{d}x^{s}_{tv} + {}^{d}x^{s}_{\text{prev}(t),v} \leq 1, \forall t \in \mathbf{T} : t > \text{first}(\mathbf{T}), v \in \mathbf{V} \tag{33}$$

2.6.4 Ground operations

The ground operations constraints were designed with aircraft in mind. Some modifications may be needed as other vehicle types are added. That being said, most of the activity types are similar, even if the terminology is incorrect for land- and seagoing vehicles.

At this time, all ground operations are mutually exclusive of the others, meaning that only one may occur in any given time step. This is enforced by Constraint (34). As activities such as maintenance are added to the model, they must be incorporated into this constraint. It is conceivable that preflight checks could fall outside of this stricture. If this is desired, the ${}^{p}x^{i}_{tv}$ term should be removed from Constraint (34). An additional constraint would then be needed to ensure that preflight checks only overlap appropriate activities, such as loading.

One activity at a time: $${}^{d}x^{i}_{tv} + {}^{a}x^{i}_{tv} + {}^{r}x^{i}_{tv} + {}^{u}x^{i}_{tv} + {}^{l}x^{i}_{tv} + {}^{p}x^{i}_{tv} \leq 1, \forall\, t \in \mathbf{T}, v \in \mathbf{V} \tag{34}$$

Constraints (35)–(38) ensure that ground operations occur only at nodes.

Link refuelling to node: $${}^{r}x^{i}_{tv} \leq \sum_{i \in \mathbf{N}} x^{n}_{tvi}, \forall t \in \mathbf{T}, v \in \mathbf{V} \tag{35}$$

Link unloading to node: $${}^{u}x^{i}_{tv} \leq \sum_{i \in \mathbf{N}} x^{n}_{tvi}, \forall t \in \mathbf{T}, v \in \mathbf{V} \tag{36}$$

Link loading to node: $${}^{l}x^{i}_{tv} \leq \sum_{i \in \mathbf{N}} x^{n}_{tvi}, \forall t \in \mathbf{T}, v \in \mathbf{V} \tag{37}$$

Link preflight to node: $${}^{p}x^{i}_{tv} \leq \sum_{i \in \mathbf{N}} x^{n}_{tvi}, \forall t \in \mathbf{T}, v \in \mathbf{V} \tag{38}$$

Constraints (39)–(44) create the start/end and duration logic for ground operations except refuelling, which is handled separately below. Durations are currently fixed by vehicle type, with no consideration for the volume of cargo or the facilities at the node. Constraint (45) requires vehicles to unload their cargo before engaging in any other operation.

$$\text{Unloading logic: } {}^{u}x^{s}_{tv} - {}^{u}x^{e}_{tv} = \begin{cases} {}^{u}x^{i}_{\text{prev}(t),v} \text{ if } t > \text{first}(\mathbf{T}), \\ 0 \text{ if } t = \text{first}(\mathbf{T}), \end{cases} \quad \forall\, t \in \mathbf{T}, v \in \mathbf{V} \tag{39}$$

$$\text{Loading logic: } {}^{l}x^{s}_{tv} - {}^{l}x^{e}_{tv} = \begin{cases} {}^{l}x^{i}_{\text{prev}(t),v} \text{ if } t > \text{first}(\mathbf{T}), \\ 0 \text{ if } t = \text{first}(\mathbf{T}), \end{cases} \quad \forall\, t \in \mathbf{T}, v \in \mathbf{V} \tag{40}$$

$$\text{Preflight logic: } {}^{p}x^{s}_{tv} - {}^{p}x^{e}_{tv} = \begin{cases} {}^{p}x^{i}_{\text{prev}(t),v} \text{ if } t > \text{first}(\mathbf{T}), \\ 0 \text{ if } t = \text{first}(\mathbf{T}), \end{cases} \quad \forall\, t \in \mathbf{T}, v \in \mathbf{V} \tag{41}$$

$$\text{Unloading duration: } {}^{u}x^{s}_{t_d v} \le {}^{u}x^{i}_{tv} \quad \forall\, t \in t_d \ldots t_d + \tau^{ul}_{v} : t < \text{last}(\mathbf{T}),\ t_d > \text{first}(\mathbf{T}), v \in \mathbf{V} \tag{42}$$

$$\text{Loading duration: } {}^{l}x^{s}_{t_d v} \le {}^{l}x^{i}_{tv} \quad \forall\, t \in t_d \ldots t_d + \tau^{ld}_{v} : t < \text{last}(\mathbf{T}),\ t_d > \text{first}(\mathbf{T}), v \in \mathbf{V} \tag{43}$$

$$\text{Preflight duration: } {}^{p}x^{s}_{t_d v} \le {}^{p}x^{i}_{tv} \quad \forall\, t \in t_d \ldots t_d + \tau^{pf}_{v} : t < \text{last}(\mathbf{T}),\ t_d > \text{first}(\mathbf{T}), v \in \mathbf{V} \tag{44}$$

$$\text{Link unloading to arrival: } {}^{u}x^{s}_{tv} = {}^{a}x^{e}_{\text{prev}(t),v}, \ \forall\, t \in \mathbf{T} : t > \text{first}(\mathbf{T}), v \in \mathbf{V} \tag{45}$$

2.6.5 Fuel

Fuel is a critical resource without which no vehicle or generator can operate. Furthermore, it is often in short supply in the Canadian North in the quantities needed by the Canadian Armed Forces (CAF). It is expected that all fuel required for helicopter operations and power generation will need to be transported forward to the FOL. Vehicular fuel capacity also constrains movement. Fuel is therefore tracked carefully within the model. Fuel transport has not yet been implemented. Instead fuel is placed at nodes as part of the model initialization and used to refuel vehicles as necessary.

Refuelling, the transfer of fuel from nodes to vehicles, is governed by Constraints (46)–(49). The maximum fuel transfer rate from node i to vehicle v is limited to:

- The maximum transfer rate for the node-vehicle pair (Constraints (46) and (47));
- Time steps when vehicle v is at node i (Constraint (46)); and
- Time steps when vehicle v is in the InRefueling state (Constraint (47)).

Additionally, the total quantity of fuel transferred to all vehicles at node i in time step t cannot be more than the quantity of fuel available at that node in the preceding time step (Constraint (48)). The resulting fuel transferred from node i to all vehicles v at that node is governed by Constraint (49). When fuel transport is added to the model, this constraint will needed to be modified.

$$\text{Bound node fuel transfer to vehicle a:} \quad z^{fv}_{itv} \le \rho^{rf}_{iv} \Delta t \, x^{n}_{tvi}, \qquad \forall\, t \in \mathbf{T} : t > \text{first}(\mathbf{T}), v \in \mathbf{V}, i \in \mathbf{N} \tag{46}$$

$$\text{Bound node fuel transfer to vehicle b:} \quad z^{fv}_{itv} \le \rho^{rf}_{iv} \Delta t \; {}^{r}x^{i}_{tv}, \qquad \forall\, t \in \mathbf{T} : t > \text{first}(\mathbf{T}), v \in \mathbf{V}, i \in \mathbf{N} \tag{47}$$

$$\text{Bound node fuel transfer to vehicle c:} \sum_{v \in \mathbf{V}} z^{fv}_{itv} \le z^{f}_{i,\text{prev}(t)}, \qquad \forall\, t \in \mathbf{T} : t > \text{first}(\mathbf{T}), i \in \mathbf{N} \tag{48}$$

$$\text{Increment node fuel:} \quad z^{f}_{it} = z^{f}_{i,\text{prev}(t)} - \sum_{v \in \mathbf{V}} z^{fv}_{itv}, \; \forall\, t \in \mathbf{T} : t > \text{first}(\mathbf{T}), i \in \mathbf{N} \tag{49}$$

Constraint (50) calculates changes to vehicles' carried fuel quantities. The calculation works because at most only one arc or node may be occupied by a vehicle. Thus only one term or the other may be greater than zero during any given timestep.

$$\text{Increment vehicle fuel: } y^{f}_{tv} = y^{f}_{\text{prev}(t),v} + \sum_{i \in \mathbf{N}} z^{\text{fv}}_{itv} - \rho^{fc}_{v} \Delta t \sum_{(i,j) \in \mathbf{A}} {}^{A}x^{o}_{tvij}, \; \forall\, t \in \mathbf{T} : t > \text{first}(\mathbf{T}), v \in \mathbf{V} \tag{50}$$

2.6.6 Evacuees

The evacuees from the stricken ship are the primary cargo of interest in this model. The objective function is based on the quality of their rescue and health outcomes. The following constraints control where evacuees may be, how they may move, and how their health is affected by their current condition and their location.

2.6.6.1 Basic evacuee constraints

Evacuees must always be at some location l in set $\mathbf{L}$ and must be at only one location. Similarly, they must always be in one of the five medical states m in the set $\mathbf{M}$.

$$\text{Occupy exactly one location:} \quad \sum_{l \in \mathbf{L}} {}^{e}x^{l}_{etl} = 1, \quad \forall\, t \in \mathbf{T}, e \in \mathbf{E} \tag{51}$$

$$\text{Occupy exactly one state:} \quad \sum_{m \in \mathbf{M}} {}^{e}x^{m}_{etm} = 1, , \forall\, t \in \mathbf{T}, e \in \mathbf{E} \tag{52}$$

2.6.6.2 Complex evacuee constraints

Constraints (53)–(55) calculate the binary product of `EvacueeLocation` ${}^{e}x^{l}_{etl}$ during the current time step and `EvacueeState` ${}^{e}x^{m}_{e,\text{prev}(t),m}$ from the previous time step. This multiplication creates a sparse four-dimensional binary variable called `EvacueeLocationState` ${}^{e}y^{ml}_{etml}$. Because of Constraint (51) on ${}^{e}x^{l}_{etl}$ and Constraint (52) on ${}^{e}x^{m}_{etm}$, for each evacuee–time step pair (e, t), ${}^{e}y^{ml}_{etml}$ can only be equal to 1 if both ${}^{e}x^{m}_{e,\text{prev}(t),m}$ and ${}^{e}x^{l}_{etl}$ equal 1, and there will always be exactly one combination of m and l for which this is true.

`EvacueeLocationState` ${}^{e}y^{ml}_{etml}$ is used to update ${}^{e}x^{m}_{etm}$ and the evacuee medical transition variable ${}^{e}x^{t}_{et}$, to calculate the evacuee vehicle capacity usage, and to as the variable of interest in the integer-valued objective function in Equation (1). We need to use the previous step's `EvacueeState` ${}^{e}x^{m}_{e,\text{prev}(t),m}$, otherwise we set up circular logic with the state updates to `EvacueeMedProgression` and `EvacueeState`below in Constraints (59)–(67).

The parallel Constraints (56)–(58) initialize ${}^{e}y^{ml}_{etml}$ at the first time step. These are similar to the corresponding Constraints (53)–(55), except that they do not use the preceding time step's values of ${}^{e}x^{m}_{etm}$, since they do not exist. This does not cause any circular logic loops because no updates are done to the values of ${}^{e}x^{l}_{etl}$ and ${}^{e}x^{m}_{etm}$ for $t = \text{first}(T)$.

Compute product of EvacueeLocation ${}^{e}x^{l}_{etl}$ and EvacueeState ${}^{e}x^{m}_{e,\text{prev}(t),m}$

$$\begin{aligned}
&\text{Compute product a: } {}^{e}y^{ml}_{etml} \le {}^{e}x^{l}_{etl}, && \forall\, e \in \mathbf{E}, t \in \mathbf{T} : t > \text{first}(\mathbf{T}), m \in \mathbf{M}, l \in \mathbf{L} \quad (53)\\
&\text{Compute product b: } {}^{e}y^{ml}_{etml} \le {}^{e}x^{m}_{e,\text{prev}(t),m}, && \forall\, e \in \mathbf{E}, t \in \mathbf{T} : t > \text{first}(\mathbf{T}), m \in \mathbf{M}, l \in \mathbf{L} \quad (54)\\
&\text{Compute product c: } {}^{e}y^{ml}_{etml} \ge {}^{e}x^{l}_{etl} + {}^{e}x^{m}_{e,\text{prev}(t),m} - 1, && \forall\, e \in \mathbf{E}, t \in \mathbf{T} : t > \text{first}(\mathbf{T}), m \in \mathbf{M}, l \in \mathbf{L} \quad (55)\\
&\text{Initialize } {}^{e}y^{ml}_{etml} \text{ a: } {}^{e}y^{ml}_{etml} \le {}^{e}x^{l}_{etl}, && \forall\, e \in \mathbf{E}, t \in \mathbf{T} : t = \text{first}(\mathbf{T}), m \in \mathbf{M}, l \in \mathbf{L} \quad (56)\\
&\text{Initialize } {}^{e}y^{ml}_{etml} \text{ b: } {}^{e}y^{ml}_{etml} \le {}^{e}x^{m}_{etm}, && \forall\, e \in \mathbf{E}, t \in \mathbf{T} : t = \text{first}(\mathbf{T}), m \in \mathbf{M}, l \in \mathbf{L} \quad (57)\\
&\text{Initialize } {}^{e}y^{ml}_{etml} \text{ c: } {}^{e}y^{ml}_{etml} \ge {}^{e}x^{l}_{etl} + {}^{e}x^{m}_{etm} - 1, && \forall\, e \in \mathbf{E}, t \in \mathbf{T} : t = \text{first}(\mathbf{T}), m \in \mathbf{M}, l \in \mathbf{L} \quad (58)
\end{aligned}$$

Constraint (59) computes the change of each evacuee's medical transition variable ${}^{e}x^{t}_{et}$ as a function of their medical state, location, and the transition rate parameter Π_{ml}. There is no requirement for Π_{ml} to contain all positive values; negative values represent conditions that lead to improving health for an evacuee.

$$\text{Update evacuee medical transition: } {}^{e}x^{t}_{et} = {}^{e}x^{t}_{e,\text{prev}(t)} + \sum_{\substack{m\in\mathbf{M}\\ l\in\mathbf{L}}} {}^{e}y^{ml}_{etml}\Pi_{ml},\ \forall\, e \in \mathbf{E}, t \in \mathbf{T} : t > \text{first}(\mathbf{T}) \quad (59)$$

2.6.6.3 Evacuee medical state updates

Constraints (60)–(67) define a set of left and right (upper) bounds on ${}^{e}x^{m}_{etm}$ based on linear functions of EvacueeMedTransition ${}^{e}x^{t}_{et}$. When combined with the requirement that exactly one state be occupied (Constraint (52)), these enforce the selection of correct EvacueeState values for both ascending and descending EvacueeMedTransition boundary crossings. Only a right bound is needed for medical state 1 and only a left bound for medical state 5. An important characteristic of these bounds to note is that they always specify a value of ${}^{e}x^{m}_{etm}$ that is within the $[0, 1]$ permissible set for a binary variable. Constraining inequalities that do not do this will result in an infeasible model.

A graphical depiction of the bounds created by these constraints is found in Figure 2. The filled regions define when ${}^{e}x^{m}_{etm}$ may be greater than 0. For example, ${}^{e}x^{m}_{et,m=3}$ may be greater than 0 when $3.0 \le {}^{e}x^{t}_{et} \le 4.0 - \epsilon$. For all other values of ${}^{e}x^{t}_{et}$, ${}^{e}x^{m}_{et,m=3}$ must be less than 1 and is therefore equal to 0. When these bounds are combined with the requirement of Constraint (52) that $\sum_{m\in\mathbf{M}} {}^{e}x^{m}_{etm} = 1$, it is guaranteed that the correct value of ${}^{e}x^{m}_{etm}$ will be selected.

Bound—evacuee state 5, left: $${}^{e}x_{etm}^{m} \leq \frac{{}^{e}x_{et}^{t}}{5}, \quad \forall\, e \in \mathbf{E}, t \in \mathbf{T} : t > \text{first}(\mathbf{T}), m \in \mathbf{M} : m = 5 \qquad (60)$$

Bound—evacuee state 4, left: $${}^{e}x_{etm}^{m} \leq \frac{{}^{e}x_{et}^{t}}{4}, \quad \forall\, e \in \mathbf{E}, t \in \mathbf{T} : t > \text{first}(\mathbf{T}), m \in \mathbf{M} : m = 4 \qquad (61)$$

Bound—evacuee state 4, right: $${}^{e}x_{etm}^{m} \leq 6 - \epsilon - {}^{e}x_{et}^{t}, \quad \forall\, e \in \mathbf{E}, t \in \mathbf{T} : t > \text{first}(\mathbf{T}), m \in \mathbf{M} : m = 4 \qquad (62)$$

Bound—evacuee state 3, left: $${}^{e}x_{etm}^{m} \leq \frac{{}^{e}x_{et}^{t}}{3}, \quad \forall\, e \in \mathbf{E}, t \in \mathbf{T} : t > \text{first}(\mathbf{T}), m \in \mathbf{M} : m = 3 \qquad (63)$$

Bound—evacuee state 3, right: $${}^{e}x_{etm}^{m} \leq 3 - \epsilon - \frac{{}^{e}x_{et}^{t}}{2}, \quad \forall\, e \in \mathbf{E}, t \in \mathbf{T} : t > \text{first}(\mathbf{T}), m \in \mathbf{M} : m = 3 \qquad (64)$$

Bound—evacuee state 2, left: $${}^{e}x_{etm}^{m} \leq \frac{{}^{e}x_{et}^{t}}{2}, \quad \forall\, e \in \mathbf{E}, t \in \mathbf{T} : t > \text{first}(\mathbf{T}), m \in \mathbf{M} : m = 2 \qquad (65)$$

Bound—evacuee state 2, right: $${}^{e}x_{etm}^{m} \leq 2 - \epsilon - \frac{{}^{e}x_{et}^{t}}{3}, \quad \forall\, e \in \mathbf{E}, t \in \mathbf{T} : t > \text{first}(\mathbf{T}), m \in \mathbf{M} : m = 2 \qquad (66)$$

Bound—evacuee state 1, right: $${}^{e}x_{etm}^{m} \leq 1.5 - \epsilon - \frac{{}^{e}x_{et}^{t}}{4}, \quad \forall\, e \in \mathbf{E}, t \in \mathbf{T} : t > \text{first}(\mathbf{T}), m \in \mathbf{M} : m = 1 \qquad (67)$$

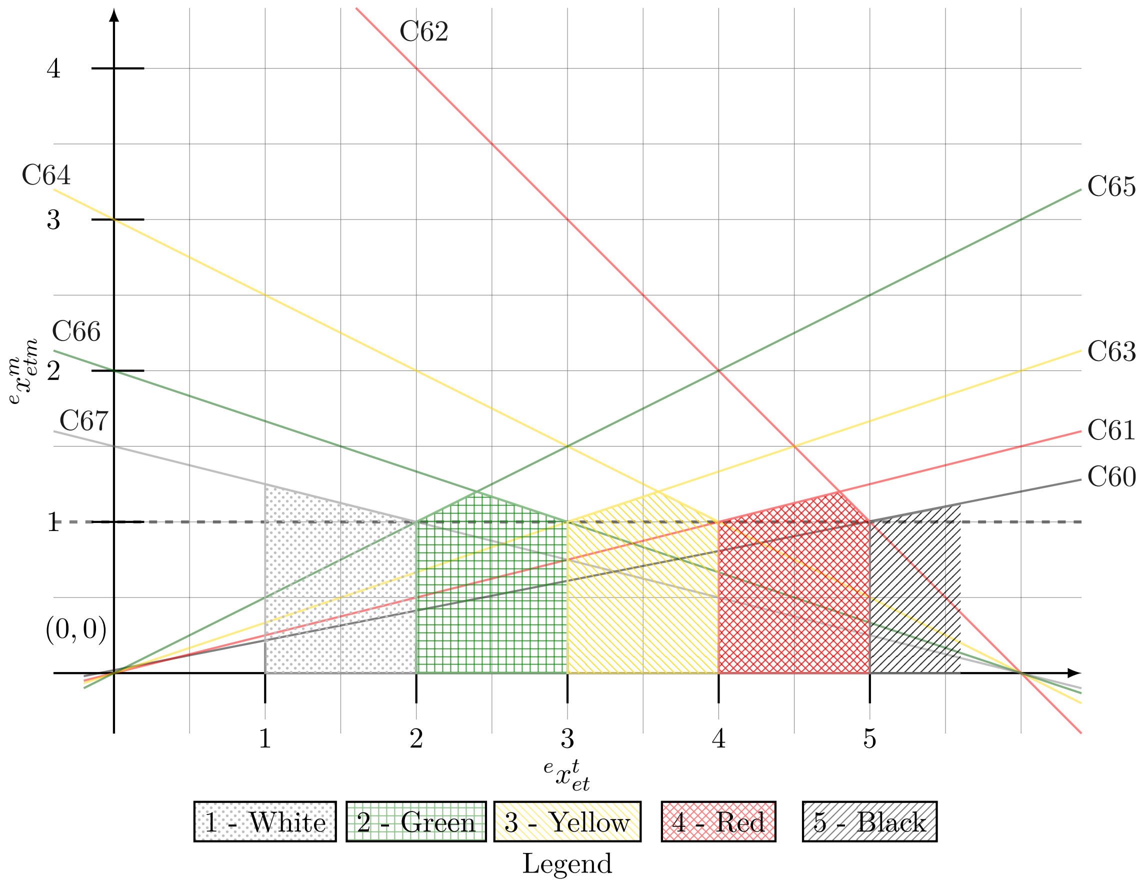

Figure 2: *Constraints relating of medical transition value and medical state. The lines are the upper-bound Constraints (60)–(67). Each is labelled by constraint number and colour-coded by applicable medical state. The filled areas are the ranges of medical transition value ${}^{e}x_{et}^{t}$ for which the corresponding medical state ${}^{e}x_{etm}^{m}$ may be equal to 1.*

2.6.6.4 Evacuee transport

The constraints in this section govern the movement of evacuees between locations. In general, evacuees should continue at their present location unless:

a) They are on a vehicle, that vehicle is located at a node and the vehicle is currently in an unloading process; or

b) They are at a node, and a vehicle is located at that node and that vehicle is currently in a loading process.

To that end, most of these constraints prevent undesired actions. For example Constraint (68) prevents evacuees from moving directly between nodes. Evacuees may only move from a node to a vehicle or vice versa.

$$\text{Bound—no evacuee teleportation: } {}^{e}x^{l}_{eti} \leq 1 - \sum_{j \in \mathbf{N}: j \neq i} {}^{e}x^{l}_{e,\text{prev}(t),j}, \ \forall\, e \in \mathbf{E}, t \in \mathbf{T} : t > \text{first}(\mathbf{T}) \tag{68}$$

Constraints (69)–(72) determine whether it is permissible for evacuee e to be loaded onto vehicle v from node i at time step t. These four constraints describe the loading permission variable ${}^{e}y^{ld}_{etvi}$ as the triple binary product of ${}^{l}x^{i}_{tv}$, ${}^{e}x^{l}_{e,\text{prev}(t),i}$, and x^{n}_{tvi}. Constraint (69) says that loading is possible if vehicle v is in the loading process at time t, Constraint (70) states at loading is possible if evacuee e was located at node i at time $\text{prev}(t)$, and Constraint (71) requires the presence of vehicle v at node i in time step t. Note that these three constraints permit the loading permission variable to be true but do not force it to be so. Because this is an enabling flag variable, we wish to force it to be true if it can be. Constraint (72) does this. Forcing the value of the enabling flag in this way reduces the size of the decision problem for the solver—the solver no longer has to sort through the possible states during its branch-and-bound search.

$$\text{Evacuee loading possible a: } {}^{e}y^{ld}_{etvi} \leq {}^{l}x^{i}_{tv}, \quad \forall\, v \in \mathbf{V}, e \in \mathbf{E}, t \in \mathbf{T} : t > \text{first}(\mathbf{T}), i \in \mathbf{N} \tag{69}$$

$$\text{Evacuee loading possible b: } {}^{e}y^{ld}_{etvi} \leq {}^{e}x^{l}_{e,\text{prev}(t),i}, \ \forall\, v \in \mathbf{V}, e \in \mathbf{E}, t \in \mathbf{T} : t > \text{first}(\mathbf{T}), i \in \mathbf{N} \tag{70}$$

$$\text{Evacuee loading possible c: } {}^{e}y^{ld}_{etvi} \leq x^{n}_{tvi}, \quad \forall\, v \in \mathbf{V}, e \in \mathbf{E}, t \in \mathbf{T} : t > \text{first}(\mathbf{T}), i \in \mathbf{N} \tag{71}$$

$$\text{Evacuee loading possible d: } {}^{e}y^{ld}_{etvi} + 2 \geq {}^{l}x^{i}_{tv} + {}^{e}x^{l}_{e,\text{prev}(t),i} + x^{n}_{tvi}, \ \forall\, v \in \mathbf{V}, e \in \mathbf{E}, \quad t \in \mathbf{T} : t > \text{first}(\mathbf{T}), i \in \mathbf{N} \tag{72}$$

Constraint (73) describes the two conditions under which an evacuee can be located on a vehicle; firstly, if the evacuee was already on that vehicle in the previous time step, or secondly, if the evacuee is able to load onto that vehicle in the current time step. All considerations about collocation of evacuee, node and vehicle have already been taken into account in Constraints (69)–(72).

$$\text{Link evacuee location to loading: } {}^{e}x^{l}_{etv} \leq {}^{e}x^{l}_{e,\text{prev}(t),v} + \sum_{i \in \mathbf{N}} {}^{e}y^{ld}_{etvi}, \ \forall\, e \in \mathbf{E}, t \in \mathbf{T} : t > \text{first}(\mathbf{T}), v \in \mathbf{V} \tag{73}$$

Constraints (74)–(76) are unloading permission constraints analogous to loading permission Constraints (69)–(71). The primary difference is that the evacuee must now have been located on vehicle v in the preceding time step (Constraint (75)). The reader may notice the absence of a fourth constraint analogous to Constraint (72). For reasons that have yet to be determined, addition of such a constraint creates an infeasible model. Leaving it out makes the model feasible at the cost of computational complexity.

$$\text{Evacuee unloading possible a: } {}^{e}y^{ul}_{etvi} \leq {}^{u}x^{i}_{tv}, \qquad \forall\, v \in \mathbf{V}, e \in \mathbf{E}, t \in \mathbf{T} : t > \text{first}(\mathbf{T}), i \in \mathbf{N} \tag{74}$$

$$\text{Evacuee unloading possible b: } {}^{e}y^{ul}_{etvi} \leq {}^{e}x^{l}_{e,\text{prev}(t),v}, \; \forall\, v \in \mathbf{V}, e \in \mathbf{E}, t \in \mathbf{T} : t > \text{first}(\mathbf{T}), i \in \mathbf{N} \tag{75}$$

$$\text{Evacuee unloading possible c: } {}^{e}y^{ul}_{etvi} \leq x^{n}_{tvi}, \qquad \forall\, v \in \mathbf{V}, e \in \mathbf{E}, t \in \mathbf{T} : t > \text{first}(\mathbf{T}), i \in \mathbf{N} \tag{76}$$

Constraint (77) states that an evacuee may be located at node i if either a) the evacuee was already on that vehicle in the previous time step; or b) the evacuee is permitted to unload from a vehicle to that node in the current time step.

$$\text{Link evacuee location to unloading: } {}^{e}x^{l}_{eti} \leq {}^{e}x^{l}_{e,\text{prev}(t),i} + \sum_{v \in \mathbf{V}} {}^{e}y^{ld}_{etvi}, \; \forall\, e \in \mathbf{E}, t \in \mathbf{T} : t > \text{first}(\mathbf{T}), i \in \mathbf{N} \tag{77}$$

2.6.6.5 Vehicle capacity constraints on evacuees

Each vehicle has a maximum capacity for each cargo type. For evacuees and personnel, this is determined by parameter κ^{p}_{v} (`capacityPassenger`). This capacity assumes the passengers are in a seated position. Some evacuees will be in poor health and so will need to be transported on stretchers. patients on stretchers consume more space than seated passengers. How much more varies by vehicle and is contained in parameter α^{ms}_{vm} (`cfPassCapacityUseByMedState`).

$$\text{Bound - vehicle evacuee capacity: } \sum_{\substack{e \in \mathbf{E} \\ m \in \mathbf{M}}} \left({}^{e}y^{ml}_{etmv} \alpha^{ms}_{vm} \right) \leq \kappa^{p}_{v}, \; \forall\, t \in \mathbf{T}, v \in \mathbf{V} \tag{78}$$

Note that this constraint is enforced at all time steps. Because evacuees' health can continue to decline during the travel time of a vehicle, it is possible for an evacuee to change medical state and hence their capacity requirement en route. If this change would cause the vehicle capacity to be exceeded, the load is not valid and the solver must choose a different combination of evacuees for transport.

3 Conclusion

The model described in the Document is self-consistent and fully functional, but lacks many features that are needed for application to the intended problem. Nonetheless, the framework that has been created will support the addition of these features with a reasonable amount of effort. The current plan is to add features in the following order:

1. The second (rear) echelon of transportation between the FOL and locations in the south of Canada, such as Canadian Forces Base (CFB) Trenton, CFB Winnipeg, and major urban centres;
2. Add Canadian Coast Guard (CCG) ships to the model. This should be primarily a data change, but it needs to be tested to ensure that the correct behaviour/doctrine is implemented;
3. Add capacities to nodes for fuel, personnel, evacuees, and supplies. This includes MOG constraints for aircraft; and
4. Add consequences for the delivery of supplies and personnel to nodes.

Features beyond this point will be added after the next model version lockdown.

References

[1] Hunter, D., Chan, J., and Rempel, M. (2019), Description of the two-echelon capacitated vehicle routing model used for the Northern Infrastructure Study, Defence Research and Development Canada, Reference Document, DRDC-RDDC-2019-D027.

[2] Hunter, D., Chan, J., and Rempel, M. (2021), Assessing the impact of infrastructure on Arctic operations, Defence Research and Development Canada, Scientific Report, DRDC-RDDC-2021-R024.

Annex A AMPL code

The code for the model described in the Document is reproduced below with minor modifications. All changes made have been to make the text fit onto the page and improve readability. Change log markers were casualties of this process. The model is functionally unchanged.

```
/* Developmental model for the Northern Infrastructure and Northern Operational Support Hub Studies
v 2.4.0

Author(s): D.G. Hunter, M.Sc.
Centre for Operational Research and Analysis
Defence Research and Development Canada

Copyright (c) 2021-2023 His Majesty the King in right of Canada, as represented by the Minister of National Defence. All Rights Reserved.

This model is an evolution of nis_ip_v053 (version 0.5.3), but has been redesigned to allow a different and much smaller time step. The previous time step
was 1 day. This was originally done to minimize the number of variables generated in the hope that GMPL could be used to solve the problem, but this led to
numerous other issues.
- Shorter time steps will allow trips to exceed 1 day
- This will allow the tracking of the physical space use (ramp spaces, etc)
- It is conceivable that other transport types than aircraft can be modelled (ships are akin to very slow aircraft)

There is an implicit assumption that the timestep is short. Given aircraft speeds (~300 km/h for helicopters, much more for FW aicraft), a time step of 5
minutes limits distance resolution to about 25 km. This should be taken into account when generating input scenarios to reduce duplication as much as
possible.

Naming conventions
-----------------
UPPER_CASE      - sets
UpperCamelCase  - Variables
lowerCamelCase  - parameters
lowercase       - iterators

Underscores are used to enhance readability

OPEN QUESTIONS:

* How many vehicles should be able to refuel at once?
* Do I need a variable for the amount of fuel loaded on to a vehicle at a place and time? I might make it easier to write some constraints

Update log
Marker (if any) | yyyy-mm-dd | Author; Description
-------------------------------------------------------------------------------------------------------------------------------------------------------------
| 2021-06-22 | D.G. Hunter; Initial creation started
| 2021-07-20 | D.G. Hunter; Started over
| 2021-08-05 | D.G. Hunter; Minimal vehicle routing functionality
| 2021-08-13 | D.G. Hunter; Current state
|            |               - Vehicle routing functional;
|            |               - Arrival and departure processes >95% functional. There is still an issue with InDeparture and InArrival being
|            |                 1 when the vehicle is at rest at a node. This is due to a lack a constraint to force this issue but it has no
|            |                 effect on any consequential part of the model outcomes;
|            |               - Fuel consumption implemented and functioning as intended.
| 2022-06-09 | D.G. Hunter; Re-implementing original logic for process triples. Testing the addition one process at a time
  /A            | 2022-07-14 | D.G. Hunter; Beginning to implement inventory and cargo other than fuel, starting with evacuees
  /B            | 2022-09-16 | D.G. Hunter; /A mod got hung up on implementation of  evacuee medical state transitions. This change implements:
|     --     |               - A new definition of EvacueeState variable as a binary over{EVACUEES, TIME, MEDICAL_STATE}
| 2022-09-29 |               - Basic evacuee loading, unloading and capacity constraints
  /C            | 2022-10-04 | D.G. Hunter  Experiments in vehicle loading and evacuee transport: Something weird is going on with the vehicle - evacuee
|     --     | (DRDC) and   interactions. Changing the objective function changes vehicle movement: If the number of trips is maximized,
| 2022-11-?21| T. Guardino  vehicles move as expected. If the loading of evacuees onto vehicles is maximized, the loading works if vehicle
|            | (AMPL);      location is ignored, but attempting to require the vehicle to be at the evacuees' node in any way blocks loading.
|            |              This version (proto_2_3.mod) is used to play and experiment with the constraints without worrying about
|            |              maintaining previous functionality.
|            |              - there is also a problem with the fuel tracking at the nodes - the node fuel inventory is declining even when
|            |                vehicles don't move.
  /D            | 2022-12-08 | D.G. Hunter; - Removing unused parameters.
|     --     |              - implemented input parameter constraint on node occupation by vehicle
| 2022-12-09 |
  /E            | 2022-12-10 | D.G. Hunter; Changes too numerous to count. Most not marked. This is a to-do list instead:
|            |              - Verify need for "EndOfWorld" constraint
|     --     |              - Verify that subtour elimination is not needed
|            |              - Test linkage of preflight to node and to departure
|            |              - Test removal of i in NODES from Preflight Duration constraint
| 2023-01-31 |              - Test that med transition update works with change to incorporate time step into the calculation
| 2023-02-06 | D.G. Hunter; Version update to 2.4; locked down and merged to main.
-----------------------------------------------------------------------------------------------------------------------------------------------------------*/

#############################################################################################################################################################
# Set declarations                                                                                                                                          #
#############################################################################################################################################################

# Set parameters

set TIME ordered;                                # Enumeration of time periods. Time period durations are defined by the timestep param.
set NODES;                                       # Set of all nodes
set ARCS within (NODES cross NODES);             # Arcs between nodes
set VEHICLES;                                    # Set of all vehicles
set EVACUEES ordered;      # Set of all evacuees
set MEDICAL_STATE ordered;                         # Set of possible medical states of evacuees
set LOCATIONS = NODES union VEHICLES;              # Set of all nodes and vehicles, describing all possible locations of an evacuee or other cargo

##########################################################################################################################################
# Parameters                                                                                                                             #
##########################################################################################################################################

param epsilon := 1e-9;                             # A very small number
param timestep;                                    # Duration of a time step [min] - default expectation is 5 minutes.
param visualFlightRules    {TIME} binary;          # = 1 if visual flight rules are in effect at time t - effectively daytime hours.
# index over ARCS or NODES in the future; account for time at origin and destination
param distance             {ARCS};                 # Distance along arc (i,j) [km]

# Node parameters
param nodeType             {NODES} symbolic;       # EVENT, FOL, REAR
param nodeRunways          {NODES}  >= 0, <= 3;    # Number of runways at a node
param nodeRunwayType       {NODES} symbolic;       # NONE, VFR, IFR
param nodeInitFuel         {NODES} >=0;            # Amount of fuel available at node at first time step [kg]
param nodeVehCanUse        {NODES, VEHICLES} binary; # = 1 if vehicle v can operate on the runway(s) at node i. This is a proxy for runway way length and
# mass/weight limit that has been precalculated.

# Refueling
# The current assumption is that only one of the following parameters can be used. That is, there is no allowance at this time for a fixed base refuelling
# time plus an additional amount of time based on refueling rate and quantity of fuel transferred.

# Refueling option A
#   The rate at which fuel can be transferred to a vehicle. [km/min] Defined by node and vehicle. This assumes that
#   a) the primary time component of refueling is the fuel transfer rate, and
#   b) that refueling happens at a continuous rate. This is not true, as there is a fixed  time cost for the movement to and from the fueling point and for
#      the set up.
param rateRefuel           {NODES, VEHICLES} >=0;

#Refueling option B (NOT USED)
# A fixed time to refuel by vehicle type and node. [min]
param durationRefuel       {NODES, VEHICLES} >=0;

# Vehicle parameters
param vType                {VEHICLES} symbolic;  # CHINOOK, CORMORANT, GLOBEMASTER, HERCULES
param timeFirstAvail       {VEHICLES} >= 0;      # Time at which vehicle is available for use [min]
param speed                {VEHICLES} >=0;       # Speed of vehicle v [km/hour]
param capacityFuel         {VEHICLES} >=0;       # Fuel capacity of vehicle v [kg]
param capacityPassenger    {VEHICLES} integer >=0; # Basic capacity for seated passengers by vehicle
param capacityPallets      {VEHICLES} integer >=0; # Cargo capacity in pallets by vehicle type
param rateFuelConsumption  {VEHICLES} >=0;       # Fuel consumption of vehicle v [kg/min]
param durationDepart       {VEHICLES} >=0;       # Time taken by the departure process [min]
param durationArrive       {VEHICLES} :=10;      # Time taken by the arrival process [min]
param durationLoad         {VEHICLES} >=0;       # A flat rate for now [min]
param durationUnload       {VEHICLES} >=0;       # A flat rate for now [min]
param durationLightMaint   {VEHICLES} >=0;       # A flat rate average time for inter-trip servicing [min]
param durationMedMaint     {VEHICLES} >=0;       # Max service day length by vehicle type
param durationMajorMaint   {VEHICLES} >=0;       # Max service day length by vehicle type
param durationPreflight    {VEHICLES} >=0;       # Time required for pre-flight checks
param initVehLocation      {VEHICLES} symbolic;  # Must be a valid node name
param intervalMedMaint     {VEHICLES} >=0;       # Time between medium maintenance [min]
param cfPassengersPerPallet{VEHICLES} >=0;       # Number of standard passenger slots exchanged for one pallet of cargo
param cfPassCapacityUseByMedState{VEHICLES, MEDICAL_STATE} >=0; # Number of standard passenger spaces used by evacuee medical state. The default value is 1;
# This value will usually be higher for more injured/sick medical states.

param durationTravel      {v in VEHICLES, (i,j) in ARCS} = timestep * (ceil(distance[i,j]/speed[v] * 60/timestep)-1); # Do the time-and-space calculations
  # once
/* Evacuee Concept:
Each evacuee is treated as an individual element of the set of evacuees. This is a departure from the previous iteration of this model, where there was
simply a set of inventory locations for evacuees in various medical states and a transition matrix that updated the inventory each night. Now each evacuee
is described by three variables:
* A current location binary variable EvacueeLocation defined overthe 3-tuple {EVACUEES, TIME, LOCATIONS}. '1' indicates that the evacuee is at the
indicated location;
* A current state transition variable EvacueeMedTransition that may take on real values >= 1 and <= 5, which is the range of values spanned by
MEDICAL_STATE. This variable is incremented on each time step with the value of the transition step for their current location and medical state; and
* A current medical state variable EvacueeState, which must take on an integer value from the set MEDICAL_STATE. This is calculated at each time step by
setting the integer variable equal to the real-valued EvacueeMedTransition variable for the previous time step.

These are governed/informed by the following parameters:
* initEvacueeLocation - Initial distribution of evacuees by location.
* initEvacueeTransition - This gives the current transition progress by evacuee at the first time step.
* medTransitionRate - the medical state transition progress per time step by location and medical state. Note that location is defined across nodes and
  vehicles per the LOCATIONS set.
* numEvacuees - No. of persons to be evacuated - this quantity is conserved.
* dead - a convenient label for the corresponding medical state
*/

# Evacuee parameters
param dead;                                       # Medical state of dead people - this value must exist in MEDICAL_STATE
check dead in MEDICAL_STATE;
param minMedState := min {m in MEDICAL_STATE} m;
param maxMedState := dead;
param initEvacueeLocation {EVACUEES, LOCATIONS} binary, default 0;
param initEvacueeProgression {EVACUEES} >= 1, <=5;
param medTransitionRate {MEDICAL_STATE, LOCATIONS} >=-1, <=1;
param initEvacueeState {EVACUEES, MEDICAL_STATE} binary, default 0;
```

```
############################################################################################################################################################
# Variables                                                                                                                                                #
############################################################################################################################################################

/*  Example of timing
---------------------
PF  - Pre-Flight checks         UL  - Unloading
D   - Departure process         RF  - Refueling
iT  - in Transit = on an arcs   SVC - Service - light maintenance
A   - Arrival process           L   - Loading
Node
====
A    |--PF--|
A           |-D-|
-               |---iT---|
B                        |-A-|       +----+--------------------------------------+-----> Note - order of these does not matter
B                            |-UL-|  |    |                                      |
B                                 |-RF-|  |                                      |
B                                      |-SVC-|                                   |
B                                  +---------|---L---|                           |
B   Simultaneous, Loading optional <-+-------|-PF-|                              |
B                                                    |-D-|                       |
-                                                        |---iT---|           +--+--+
C                                                                 |-A-|       |     |
C                                                                     |-UL-|  |     |
C                                                                          |-SVC-|  |
C                                                                                |-RF-|
C                                                                                     |--PF--| <--- Note: Loading can be omitted
C                                                                                            |-D-|
*/

# Vehicle variables
/* var MediumMaintIn         {TIME, VEHICLES} binary;
var MediumMaintHappening  {TIME, VEHICLES} binary;
var MediumMaintOut        {TIME, VEHICLES} binary;
var TransitTimeSinceLastMedMaint {TIME, VEHICLES} >=0;

var MajorMaintIn          {TIME, VEHICLES} binary;
var MajorMaintHappening   {TIME, VEHICLES} binary;
var MajorMaintOut        {TIME, VEHICLES} binary;
var TransitTimeSinceLastMajorMaint {TIME, VEHICLES} >=0; */

# Transit variables
# /B var StartTransit          {TIME, VEHICLES} binary; # = 1 if vehicle v departs at time t, 0 otherwise
# /B var EndTransit            {TIME, VEHICLES} binary; # = 1 if vehicle v arrives at time t, 0 otherwise
# /B var InTransit             {TIME, VEHICLES} binary; # = 1 if vehicle v is travelling at time t, 0 otherwise

# We need some way to figure out a vehicle's current location
var VehAtNode               {TIME, VEHICLES, NODES} binary; # = 1 if vehicle v is at node i during timestep t
var VehStartArc             {TIME, VEHICLES, ARCS} binary; # = 1 if vehicle v leaves node i to go to node j during timestep t
var VehOnArc                {TIME, VEHICLES, ARCS} binary; # = 1 if vehicle v is travelling along arc (i,j) during timestep t
var VehEndArc               {TIME, VEHICLES, ARCS} binary; # = 1 if vehicle v arrives at node j from node i during timestep t

# Departure variables
var StartDeparture          {TIME, VEHICLES} binary; # = 1 if vehicle v departs at time t, 0 otherwise
var EndDeparture            {TIME, VEHICLES} binary; # = 1 if vehicle v arrives at time t, 0 otherwise
var InDeparture             {TIME, VEHICLES} binary; # = 1 if vehicle v is travelling at time t, 0 otherwise

# Arrival variables
var StartArrival            {TIME, VEHICLES} binary; # = 1 if vehicle v begins arriving at time t, 0 otherwise
var EndArrival              {TIME, VEHICLES} binary; # = 1 if vehicle v is arriving at time t, 0 otherwise
var InArrival               {TIME, VEHICLES} binary; # = 1 if vehicle v is ends arriving at time t, 0 otherwise

# Unloading variables
var StartUnloading          {TIME, VEHICLES} binary; # = 1 if vehicle v begins unloading at time t, 0 otherwise
var EndUnloading            {TIME, VEHICLES} binary; # = 1 if vehicle v is unloading at time t, 0 otherwise
var InUnloading             {TIME, VEHICLES} binary; # = 1 if vehicle v is ends unloading at time t, 0 otherwise

# Refueling Variables
var StartRefueling          {TIME, VEHICLES} binary; # = 1 if vehicle v begins refueling at time t, 0 otherwise
var EndRefueling            {TIME, VEHICLES} binary; # = 1 if vehicle v is refueling at time t, 0 otherwise
var InRefueling             {TIME, VEHICLES} binary; # = 1 if vehicle v is ends refueling at time t, otherwise

# Light Maintenance Variables
/* var StartLtMaint          {TIME, VEHICLES} binary; # = 1 if vehicle v begins light maintenance at time t, 0 otherwise
var EndLtMaint            {TIME, VEHICLES} binary; # = 1 if vehicle v is light maintenance at time t, 0 otherwise
var InLtMaint             {TIME, VEHICLES} binary; # = 1 if vehicle v is ends light maintenance at time t, 0 otherwise */

# Loading Variables
var StartLoading            {TIME, VEHICLES} binary; # = 1 if vehicle v begins loading at time t, 0 otherwise
var EndLoading              {TIME, VEHICLES} binary; # = 1 if vehicle v is loading at time t, 0 otherwise
var InLoading               {TIME, VEHICLES} binary; # = 1 if vehicle v is ends loading at time t, 0 otherwise

var StartPreflight          {TIME, VEHICLES} binary; # = 1 if vehicle v begins preflight process at time t, 0 otherwise
var EndPreflight            {TIME, VEHICLES} binary; # = 1 if vehicle v is preflight process at time t, 0 otherwise
var InPreflight             {TIME, VEHICLES} binary; # = 1 if vehicle v is ends preflight process at time t, 0 otherwise

var VehFuel                 {TIME, v in VEHICLES}, >=0, <= capacityFuel[v]; # Quantity of fuel on vehicle v at beginning of timestep t [kg]

# Node variables - inventory, etc.

var NodeFuel                {NODES, TIME} >=0, <= 1000000.; # [kg] Amount of fuel at node i at time step t. Only one fuel type for now
var NodeFuelTransferToVeh   {i in NODES, TIME, v in VEHICLES} >=0, <= rateRefuel[i,v]; # [kg] Mass of fuel transferred from node i to vehicle v during
# timestep
# Evacuee variables
var EvacueeLocation         {EVACUEES, TIME, LOCATIONS} binary;
var EvacueeState            {EVACUEES, TIME, MEDICAL_STATE} binary;
#/B var EvacueeMedTransition {EVACUEES, TIME} >= minMedState, <= maxMedState;
var EvacueeMedTransition    {EVACUEES, TIME} >= minMedState, <= maxMedState+1;

# Intermediate variables
var EvacueeLocationState    {EVACUEES, TIME, MEDICAL_STATE, LOCATIONS} binary;  # Used to store the (linearized) product of EvacueeLocation and EvacueeState

var EvacueeCanLoad          {EVACUEES, TIME, VEHICLES, NODES} binary;
var EvacueeCanUnload        {EVACUEES, TIME, VEHICLES, NODES} binary;

##########################################################################################################################################################
# Objective Function                                                                                                                                     #
##########################################################################################################################################################

# Objective: Keep people alive (and healthy)

maximize LivingEvacueesAtFOL: sum{t in TIME, e in EVACUEES, m in MEDICAL_STATE, l in LOCATIONS : m <> dead and l = "POND_INLET"}
EvacueeLocationState[e,t,m,l];

##########################################################################################################################################################
# Constraints                                                                                                                                            #
##########################################################################################################################################################

#---------------------------------------------------------------------------------------------------------------------------------------------------------
# Initial conditions
#---------------------------------------------------------------------------------------------------------------------------------------------------------
s.t. init_VehicleLocation     {t in TIME, v in VEHICLES, i in NODES : t = first(TIME)} :
VehAtNode[t,v,i] = if (i == initVehLocation[v]) then 1 else 0;
s.t. init_VehicleArcs         {t in TIME, v in VEHICLES, (i,j) in ARCS : t = first(TIME)} :
VehOnArc[t,v,i,j] = 0;
s.t. init_Arrivals            {t in TIME, v in VEHICLES : t = first(TIME)} :
EndArrival[t,v] = 1;

subject to init_NodeFuel      {i in NODES, t in TIME : t = first(TIME)} : NodeFuel[i,t] = nodeInitFuel[i];
subject to init_VehicleFuel   {t in TIME, v in VEHICLES : t = first(TIME)} : VehFuel[t,v] = capacityFuel[v];

s.t. init_EvacueeLocation     {e in EVACUEES, t in TIME, l in LOCATIONS : t = first(TIME)} : EvacueeLocation[e,t,l] =initEvacueeLocation[e,l];
s.t. init_EvacueeMedProgression {e in EVACUEES, t in TIME : t = first(TIME)} : EvacueeMedTransition[e,t] = initEvacueeProgression[e];
s.t. init_EvacueeMedicalState {e in EVACUEES, t in TIME, m in MEDICAL_STATE : t = first(TIME)} : EvacueeState[e,t,m] = initEvacueeState[e,m];
#---------------------------------------------------------------------------------------------------------------------------------------------------------
# Vehicle routing
#---------------------------------------------------------------------------------------------------------------------------------------------------------

# Apply visual flight rules time periods - this may need to change by node/arc and aircraft type
s.t. bound_visualFlightRules  {t in TIME, v in VEHICLES, (i,j) in ARCS} : VehOnArc[t,v,i,j] <= visualFlightRules[t];

# Prevent vehicles from occupying nodes prohibited to them
s.t. bound_VehCanUseNode {t in TIME, v in VEHICLES, i in NODES} : VehAtNode[t,v,i] <=nodeVehCanUse[i,v];

# Constraints pertaining to beginning and ending travel (on arc) status
s.t. define_VehOnArcLogic    {t in TIME, v in VEHICLES, (i,j) in ARCS} :
VehStartArc[t,v,i,j] - VehEndArc[t,v,i,j] = VehOnArc[t,v,i,j] - (if (t>first(TIME)) then VehOnArc[prev(t),v,i,j] else 0);

# Definine minimum time on an arc as a function of arc length and vehicle speed
s.t. define_OnArcDuration     {t_d in TIME, v in VEHICLES, (i,j) in ARCS, t in t_d .. t_d + durationTravel[v,i,j] by timestep : t < last(TIME)} :
VehStartArc[t_d,v,i,j] <= VehOnArc[t,v,i,j];

# All vehicles must complete all the arcs that they start
s.t. MaintainPerfectRecord    {v in VEHICLES, (i,j) in ARCS} :
sum {t in TIME} VehStartArc[t,v,i,j] = sum {t2 in TIME} VehEndArc[t2,v,i,j];

/*

It is not clear that this constraint is needed. Let’s see what happens without it

# Prevent weirdness when an travel on arc would continue past the end of the modelled time period
s.t. DoNotFallOffEndOfWorld  {v in VEHICLES, (i,j) in ARCS, t_d in last(TIME) - durationTravel[v,i,j] .. last(TIME) by timestep} :
VehStartArc[t_d,v,i,j] = 0; # NOTE: This depends on durationTravel being an exact multiple of timestep. If this fails, then replace with
# t_d in last(TIME) .. last(TIME) - durationTravel[v,i,j] by -timestep
*/

s.t. link_EndArcToAtNode      {t in TIME, v in VEHICLES, (i,j) in ARCS : t > first(TIME)} :
VehEndArc[prev(t),v,i,j] <= VehAtNode[t,v,j];

# Vehicles must be located at/on exactly 1 node or arc at all times
s.t. NoSchroedingersAircraft  {t in TIME, v in VEHICLES} : sum {i in NODES} VehAtNode[t,v,i] + sum{(i2,j) in ARCS} VehOnArc[t,v,i2,j] = 1;

# No vehicle teleportation
s.t. link_StayPutUnlessMoved  {t in TIME, v in VEHICLES, i in NODES : t > first(TIME)} :
VehAtNode[t,v,i] >= VehAtNode[prev(t),v,i] - sum {(i2,j) in ARCS : i2 == i} VehStartArc[t,v,i2,j];

# No vehicle teleportation 2
s.t. bound_NoTeleport {t in TIME, v in VEHICLES, i in NODES : t > first(TIME)} :
VehAtNode[t,v,i] <= 1 - sum {j in NODES : j != i} VehAtNode[prev(t),v,j];

#---------------------------------------------------------------------------------------------------------------------------------------------------------
# Arrival and Departure
#---------------------------------------------------------------------------------------------------------------------------------------------------------
```

```
s.t. link_ArrivalToNode       {t in TIME, v in VEHICLES : t < last(TIME)} :
InArrival[t,v] <= sum {i in NODES} VehAtNode[t,v,i];

s.t. link_DepartureToNode     {t in TIME, v in VEHICLES : t < last(TIME)} :
InDeparture[t,v] <= sum {i in NODES} VehAtNode[t,v,i];

s.t. define_ArrivalLogic {t in TIME, v in VEHICLES} :
StartArrival[t,v] - EndArrival[t,v] = InArrival[t,v] -if(t > first(TIME)) then InArrival[prev(t), v] else 0;

s.t. define_ArrivalDuration   {t_a in TIME, v in VEHICLES, t in t_a .. t_a + durationArrive[v] by timestep : t < last(TIME) and t_a > first(TIME)} :
StartArrival[t_a, v] <= InArrival[t,v];

s.t. link_ArcToArrival        {t in TIME, v in VEHICLES : t > first(TIME)} :
sum {(i,j) in ARCS} VehEndArc[prev(t),v,i,j] <= StartArrival[t,v];

s.t. define_DepartureLogic {t in TIME, v in VEHICLES} :
StartDeparture[t,v] - EndDeparture[t,v] = InDeparture[t,v] - if(t > first(TIME)) then InDeparture[prev(t), v] else 0;

s.t. define_DepartureDuration {t_d in TIME, v in VEHICLES, t in t_d .. t_d+durationDepart[v] by timestep : t < last(TIME)} :
StartDeparture[t_d, v] <= InDeparture[t,v];

s.t. link_DepartureToArc      {t in TIME, v in VEHICLES : t > first(TIME)} :
sum {(i,j) in ARCS} VehStartArc[t,v,i,j] = EndDeparture[prev(t),v];

s.t. bound_ArrivalByTime      {v in VEHICLES, t in last(TIME) .. last(TIME) - durationArrive[v] + 1 by -timestep} :
StartArrival[t,v] = 0;

s.t. bound_DepartureByTime_a   {v in VEHICLES, t in first(TIME) .. first(TIME) + durationDepart[v] - 1 by timestep} :
EndDeparture[t,v] = 0;
s.t. bound_DepartureByTime_b   {v in VEHICLES, t in last(TIME) .. last(TIME) - durationDepart[v] + 1 by -timestep} :
StartDeparture[t,v] = 0;

s.t. bound_NoAdjacentStartArrival {t in TIME, v in VEHICLES : t > first(TIME)} :
StartArrival[prev(t), v] + StartArrival[t,v] <= 1;

s.t. bound_NoAdjacentStartDeparture {t in TIME, v in VEHICLES : t > first(TIME)} :
StartDeparture[prev(t), v] + StartDeparture[t,v] <= 1;

#------------------------------------------------------------------------------------------------------------------------------------------------------------
#   Ground Operations
#------------------------------------------------------------------------------------------------------------------------------------------------------------

s.t. bound_OnlyOneActivityAtOnce {t in TIME, v in VEHICLES} :
InDeparture[t,v] + InArrival[t,v] + InRefueling[t,v] + InUnloading[t,v] + InLoading[t,v] + InPreflight[t,v] <= 1;

s.t. link_RefuelingToNode {t in TIME, v in VEHICLES} : InRefueling[t,v] <= sum {i in NODES} VehAtNode[t,v,i];

s.t. link_UnloadingToNode {t in TIME, v in VEHICLES} : InUnloading[t,v] <= sum {i in NODES} VehAtNode[t,v,i];

s.t. link_LoadingToNode {t in TIME, v in VEHICLES} : InLoading[t,v] <= sum {i in NODES} VehAtNode[t,v,i];

s.t. link_PreflightToNode {t in TIME, v in VEHICLES} : InPreflight[t,v] <= sum {i in NODES} VehAtNode[t,v,i];

# Force Preflight to immediately precede departure
s.t. link_PreflightToDeparture {t in TIME, v in VEHICLES : t > first(TIME)} : EndPreflight[prev(t),v] <= StartDeparture[t,v];

# Logic for start, end and in progress variables for ground activities
/* s.t. logic_LtMaint            {t in TIME, v in VEHICLES}:
StartLtMaint[t,v] - EndLtMaint[t,v] = InLtMaint[t,v] - (if t>first(TIME) then InLtMaint[prev(t),v] else 0); */

s.t. logic_Unloading          {t in TIME, v in VEHICLES}:
StartUnloading[t,v] - EndUnloading[t,v] = InUnloading[t,v] - (if t>first(TIME) then InUnloading[prev(t),v] else 0);

s.t. logic_Loading            {t in TIME, v in VEHICLES}:
StartLoading[t,v] - EndLoading[t,v] = InLoading[t,v] - (if t>first(TIME) then InLoading[prev(t),v] else 0);

# /C Force unloading cycle to occur by chaining StartUnloading to EndArrival
s.t. link_UnloadingAfterArrival {t in TIME, v in VEHICLES : t > first(TIME)}:
StartUnloading[t,v] = EndArrival[prev(t),v];

s.t. logic_Preflight          {t in TIME, v in VEHICLES}:
StartPreflight[t,v] - EndPreflight[t,v] = InPreflight[t,v] - (if t>first(TIME) then InPreflight[prev(t),v] else 0);

# Activity durations
/* s.t. define_LtMaintDuration   {v in VEHICLES, t_m in TIME, t in t_m .. t_m+durationLightMaint[v] by timestep : t < last(TIME)} :
StartLtMaint[t_m, v] <= InLtMaint[t,v];
s.t. define_RefuelingDuration {i in NODES, v in VEHICLES, t_f in TIME, t in t_f .. t_f+durationRefuel[i,v] by timestep : t < last(TIME)} :
StartRefueling[t_f, v] <= InRefueling[t,v];*/

s.t. define_UnloadingDuration {v in VEHICLES, t_u in TIME, t in t_u .. t_u+durationUnload[v] by timestep : t < last(TIME)} :
StartUnloading[t_u, v] <= InUnloading[t,v];

s.t. define_LoadingDuration   {v in VEHICLES, t_l in TIME, t in t_l .. t_l+durationLoad[v] by timestep : t < last(TIME)} :
StartLoading[t_l, v] <= InLoading[t,v];

s.t. define_PreflightDuration {v in VEHICLES, t_p in TIME, t in t_p .. t_p+durationPreflight[v] by timestep : t < last(TIME)} :
StartPreflight[t_p, v] <= InPreflight[t,v];
```

```
#-----------------------------------------------------------------------------------------------------------------------------------------------------------------------
# Fuel
#
#   FFFFFFFFFF  UU      UU  EEEEEEEEEE  LL
#   FF          UU      UU  EE          LL
#   FF          UU      UU  EE          LL
#   FFFFFFF     UU      UU  EEEEEEE     LL
#   FF          UU      UU  EE          LL
#   FF           UU    UU   EE          LL
#   FF            UUUUUU    EEEEEEEEEE  LLLLLLLLLL
#
#-----------------------------------------------------------------------------------------------------------------------------------------------------------------------

## Refueling ##
# Maximum transfer rate from node i to vehicle v is limited by:
# - rateRefuel for the node-vehicle pair (Constraints a and b);
# - time steps when vehicle v is at node i (Constraint a);
# - time steps when vehicle v is in the InRefueling state (Constraint b);
# - the total quantity of fuel available at node i in the preceding time step (Constraint c)
s.t. bound_NodeFuelTransferToVehicle_a {t in TIME, v in VEHICLES, i in NODES : t > first(TIME)} :
NodeFuelTransferToVeh[i,t,v] <= rateRefuel[i,v] * timestep * VehAtNode[t,v,i];

s.t. bound_NodeFuelTransferToVehicle_b {t in TIME, v in VEHICLES, i in NODES : t > first(TIME)} :
NodeFuelTransferToVeh[i,t,v] <= rateRefuel[i,v] * timestep * InRefueling[t,v];

s.t. bound_NodeFuelTransferToVehicle_c {t in TIME, i in NODES : t > first(TIME)} :
sum {v in VEHICLES} NodeFuelTransferToVeh[i,t,v] <= NodeFuel[i,prev(t)];

## Increment fuel quantities of nodes and vehicles at nodes ##

s.t. increment_NodeFuel {t in TIME, i in NODES : t > first(TIME)} :
NodeFuel[i,t] = NodeFuel[i, prev(t)] - sum {v in VEHICLES} NodeFuelTransferToVeh[i,t,v];

# Increment vehicle fuel
# + Fuel added by refuelling ops at nodes
# - Consume fuel for each time step that a vehicle is in transit along an arc
# These two items should never occur in the same time step
s.t. increment_VehicleFuel {t in TIME, v in VEHICLES : t > first(TIME)} :
VehFuel[t,v] - VehFuel[prev(t),v] = (sum {i in NODES} NodeFuelTransferToVeh[i,t,v]) -
(sum {(i2,j) in ARCS} VehOnArc[t,v,i2,j]) * rateFuelConsumption[v] * timestep;

#---------------------------------------------------------------------------------------------------------------------------------------------------------------------
# Evacuees
#
#   EEEEEEEEEE  VV      VV      AA        CCCCCC    UU      UU  EEEEEEEEEE  EEEEEEEEEE   SSSSSSSS
#   EE          VV      VV    AA  AA     CC    CC   UU      UU  EE          EE          SS      SS
#   EE           VV    VV     AA  AA    CC          UU      UU  EE          EE          SS
#   EEEEEEE      VV    VV    AAAAAAAA   CC          UU      UU  EEEEEEE     EEEEEEE       SSSSSS
#   EE            VV  VV     AA    AA   CC          UU      UU  EE          EE                  SS
#   EE             VVVV     AA      AA   CC    CC    UU    UU   EE          EE          SS      SS
#   EEEEEEEEEE      VV      AA      AA    CCCCCC      UUUUUU    EEEEEEEEEE  EEEEEEEEEE   SSSSSSSS
#
#---------------------------------------------------------------------------------------------------------------------------------------------------------------------
# Basics
# All evacuees occupy exactly one location
s.t. limit_EvacueesAtOneLocation {e in EVACUEES, t in TIME} : sum {l in LOCATIONS} EvacueeLocation[e,t,l] = 1;

# All evacuees occupy exactly one state
s.t. ExactlyOneMedState {e in EVACUEES, t in TIME : t > first(TIME)} : sum {m in MEDICAL_STATE} EvacueeState[e,t,m] = 1;

# Not basics
# Calculate the product of the current time step EvacueeLocation and previous time step EvacueeState. This creates a binary array that we will use to
# update the EvacueeState for the current time step and to calculate the evacuee vehicle capacity usage, among other things.
# We need to use the previous step’s EvacueeState, otherwise we set up circular logic with the state updates below
s.t. EvacueeLocationStateProduct_a {e in EVACUEES, t in TIME, m in MEDICAL_STATE, l in LOCATIONS : t > first(TIME)} :
EvacueeLocationState[e,t,m,l] <= EvacueeLocation[e,t,l];
s.t. EvacueeLocationStateProduct_b {e in EVACUEES, t in TIME, m in MEDICAL_STATE, l in LOCATIONS : t > first(TIME)} :
EvacueeLocationState[e,t,m,l] <= EvacueeState[e,prev(t),m];
s.t. EvacueeLocationStateProduct_c {e in EVACUEES, t in TIME, m in MEDICAL_STATE, l in LOCATIONS : t > first(TIME)} :
EvacueeLocationState[e,t,m,l] >= EvacueeLocation[e,t,l] + EvacueeState[e,prev(t),m] - 1;

# Set initial values of EvacueeLocationState
# - We don’t care about the timestep overlap here because the inital EvacueeState and EvacueeLocation are defined in the data parameters
s.t. init_EvacueeLocationStateProduct_a {e in EVACUEES, t in TIME, m in MEDICAL_STATE, l in LOCATIONS : t = first(TIME)} :
EvacueeLocationState[e,t,m,l] <= EvacueeLocation[e,t,l];
s.t. init_EvacueeLocationStateProduct_b {e in EVACUEES, t in TIME, m in MEDICAL_STATE, l in LOCATIONS : t = first(TIME)} :
EvacueeLocationState[e,t,m,l] <= EvacueeState[e,t,m];
s.t. init_EvacueeLocationStateProduct_c {e in EVACUEES, t in TIME, m in MEDICAL_STATE, l in LOCATIONS : t = first(TIME)} :
EvacueeLocationState[e,t,m,l] >= EvacueeLocation[e,t,l] + EvacueeState[e,t,m] - 1;

# Use the EvacueeLocationState to update EvacueeMedTransition
s.t. EvacueeMedTransitionUpdate {e in EVACUEES, t in TIME : t > first(TIME)} :
EvacueeMedTransition[e,t] = EvacueeMedTransition[e, prev(t)] +
timestep * sum {l in LOCATIONS, m in MEDICAL_STATE} EvacueeLocationState[e,t,m,l] * medTransitionRate[m,l];
# /E    sum {l in LOCATIONS, m in MEDICAL_STATE} EvacueeLocationState[e,t,m,l] * medTransitionRate[m,l];

# Evacuee medical state update
# - These define a set of left and right (upper) bounds based on linear functions of EvacueeMedTransition
# - When combined with the requirement that exactly one state be occupied (above), this should enforce the correct EvacueeState values for both ascending
#   and descending EvacueeMedTransition boundary crossings.
# - Only a rb needed for med state 1 and only a lb for med state 5
s.t. EvacueeState_5_lb {e in EVACUEES, t in TIME, m in MEDICAL_STATE : m = 5 and t > first(TIME)} :
```

```
EvacueeState[e,t,m] <= EvacueeMedTransition[e,t] / 5;
#s.t. EvacueeState_5_rb {e in EVACUEES, t in TIME, m in MEDICAL_STATE : m = 5 and t > first(TIME)} :
#    EvacueeState[e,t,m] >= EvacueeMedTransition[e,t] - 5 + epsilon;

s.t. EvacueeState_4_lb {e in EVACUEES, t in TIME, m in MEDICAL_STATE : m = 4 and t > first(TIME)} :
EvacueeState[e,t,m] <= EvacueeMedTransition[e,t] / 4;
s.t. EvacueeState_4_rb {e in EVACUEES, t in TIME, m in MEDICAL_STATE : m = 4 and t > first(TIME)} :
EvacueeState[e,t,m] <= 6 - epsilon - EvacueeMedTransition[e,t];

s.t. EvacueeState_3_lb {e in EVACUEES, t in TIME, m in MEDICAL_STATE : m = 3 and t > first(TIME)} :
EvacueeState[e,t,m] <= EvacueeMedTransition[e,t] / 3;
s.t. EvacueeState_3_rb {e in EVACUEES, t in TIME, m in MEDICAL_STATE : m = 3 and t > first(TIME)} :
EvacueeState[e,t,m] <= 3 - epsilon - EvacueeMedTransition[e,t] / 2;

s.t. EvacueeState_2_lb {e in EVACUEES, t in TIME, m in MEDICAL_STATE : m = 2 and t > first(TIME)} :
EvacueeState[e,t,m] <= EvacueeMedTransition[e,t] / 2;
s.t. EvacueeState_2_rb {e in EVACUEES, t in TIME, m in MEDICAL_STATE : m = 2 and t > first(TIME)} :
EvacueeState[e,t,m] <= 2 - epsilon - EvacueeMedTransition[e,t] / 3;

s.t. EvacueeState_1_rb {e in EVACUEES, t in TIME, m in MEDICAL_STATE : m = 1 and t > first(TIME)} :
EvacueeState[e,t,m] <= 1.5 - epsilon - EvacueeMedTransition[e,t] / 4;

# Evacuee transport
# Evacuees should continue at their present location unless
# a) they are on an aircraft and that aircraft is located at a node and the aircraft is currently in an unloading process
# b) they are at a node and an aircraft is located at a node and that aircraft is currently in a loading process

# No evacuee may move directly from one node to another. All evacuees must move from a node to a vehicle or vice versa
s.t. bound_NoEvacueeTeleportation {e in EVACUEES, t in TIME, i in NODES : t > first(TIME)} :
EvacueeLocation[e,t,i] <= 1 - sum{j in NODES : j != i} EvacueeLocation[e,prev(t),j];

s.t. VehicleEvacueeLoadingPossible_a {v in VEHICLES, e in EVACUEES, t in TIME, i in NODES : t > first(TIME)} :
InLoading[t,v] >= EvacueeCanLoad[e,t,v,i];
s.t. VehicleEvacueeLoadingPossible_b {v in VEHICLES, e in EVACUEES, t in TIME, i in NODES : t > first(TIME)} :
EvacueeLocation[e,prev(t),i] >= EvacueeCanLoad[e,t,v,i];
s.t. VehicleEvacueeLoadingPossible_c {v in VEHICLES, e in EVACUEES, t in TIME, i in NODES : t > first(TIME)} :
VehAtNode[t,v,i] >= EvacueeCanLoad[e,t,v,i];
s.t. VehicleEvacueeLoadingPossible_d {v in VEHICLES, e in EVACUEES, t in TIME, i in NODES : t > first(TIME)} :
InLoading[t,v] + EvacueeLocation[e,prev(t),i]  + VehAtNode[t,v,i] <= EvacueeCanLoad[e,t,v,i] + 2;

s.t. link_EvacueeLocation_to_EvacueeCanLoad {e in EVACUEES, t in TIME, v in VEHICLES : t > first(TIME)} :
EvacueeLocation[e,t,v] <= EvacueeLocation[e,prev(t),v] + sum {i in NODES} EvacueeCanLoad[e,t,v,i];

s.t. VehicleEvacueeUnloadingPossible_a {v in VEHICLES, e in EVACUEES, t in TIME, i in NODES : t > first(TIME)} :
EvacueeCanUnload[e,t,v,i] <= InUnloading[t,v];
s.t. VehicleEvacueeUnloadingPossible_b {v in VEHICLES, e in EVACUEES, t in TIME, i in NODES : t > first(TIME)} :
EvacueeCanUnload[e,t,v,i] <= EvacueeLocation[e,prev(t),v];
s.t. VehicleEvacueeUnloadingPossible_c {v in VEHICLES, e in EVACUEES, t in TIME, i in NODES : t > first(TIME)} :
EvacueeCanUnload[e,t,v,i] <= VehAtNode[t,v,i];

s.t. link_EvacueeLocation_to_EvacueeCanUnload {e in EVACUEES, t in TIME, i in NODES : t > first(TIME)} :
EvacueeLocation[e,t,i] <= EvacueeLocation[e,prev(t),i] + sum {v in VEHICLES} EvacueeCanUnload[e,t,v,i];


# Vehicle evacuee capacity constraint
s.t. VehicleEvacueeCapacity {v in VEHICLES, t in TIME, i in LOCATIONS inter VEHICLES : v = i} :
sum{e in EVACUEES, m in MEDICAL_STATE} (EvacueeLocationState[e,t,m,i] * cfPassCapacityUseByMedState[v,m]) <= capacityPassenger[v];
```

List of symbols/abbreviations/acronyms/initialisms

AOR	area of responsibility
CAF	Canadian Armed Forces
CCG	Canadian Coast Guard
CFB	Canadian Forces Base
CJOC	Canadian Joint Operations Command
CONPLAN	contingency plan
FOL	forward operating location
IFR	instrument flight rules
MOG	maximum on ground
OR&A	Operational Research and Analysis
VFR	visual flight rules

DOCUMENT CONTROL DATA

*Security markings for the title, abstract and keywords must be entered when the document is sensitive.

1. ORIGINATOR (The name and address of the organization preparing the document. A DRDC Centre sponsoring a contractor's report, or a tasking agency, is entered in Section 8.) DRDC – Centre for Operational Research and Analysis NDHQ Carling, 60 Moodie Drive, Building 7S.2, Ottawa ON K1A 0K2, Canada	2a. SECURITY MARKING (Overall security marking of the document, including supplemental markings if applicable.) CAN UNCLASSIFIED 2b. CONTROLLED GOODS NON-CONTROLLED GOODS DMC A

3. TITLE (The document title and subtitle as indicated on the title page.)

Interim development progress on a mathematical programming model for Northern Operational Support Hub location and facility selection: The mass evacuation scenario

4. AUTHORS (Last name, followed by initials – ranks, titles, etc. not to be used. Use semi-colon as delimiter.)

Hunter, D. G.

5. DATE OF PUBLICATION (Month and year of publication of document.) May 2023	6a. NO. OF PAGES (Total pages, including Annexes, excluding DCD, covering and verso pages.) 34	6b. NO. OF REFS (Total cited in document.) 2

7. DOCUMENT CATEGORY (e.g., Scientific Report, Contract Report, Scientific Letter)

Reference Document

8. SPONSORING CENTRE (The name and address of the department project or laboratory sponsoring the research and development.)

DRDC – Centre for Operational Research and Analysis
NDHQ Carling,
60 Moodie Drive, Building 7S.2,
Ottawa ON
K1A 0K2, Canada

9a. PROJECT OR GRANT NO. (If appropriate, the applicable research and development project or grant number under which the document was written. Please specify whether project or grant.) CVPE_003	9b. CONTRACT NO. (If appropriate, the applicable contract number under which the document was written.)
10a. DRDC PUBLICATION NUMBER DRDC-RDDC-2023-D069	10b. OTHER DOCUMENT NO(s). (Any other numbers which may be assigned to this document either by the originator or by the sponsor.)

11a. FUTURE DISTRIBUTION WITHIN CANADA (Approval for further dissemination of the document. Security classification must also be considered.)

Public release

11b. FUTURE DISTRIBUTION OUTSIDE CANADA (Approval for further dissemination of the document. Security classification must also be considered.)

Public release

12. KEYWORDS, DESCRIPTORS or IDENTIFIERS (Use semi-colon as a delimiter.)

Optimization and Mathematical Programming; Transportation and Logistics; Arctic Region

13a. ABSTRACT (When available in the document, the English version of the abstract must be included here.)

The Canadian Joint Operations Command Operational Research and Analysis team has been tasked with supporting the Northern Operational Support Hub facility location and selection process. To do this, the team is developing a series of optimization models representing the logistical aspects of the implementation of domestic contingency plans (CONPLANs) in the Canadian North. This Reference Document contains the development to date on a mathematical specification of the first scenario, a CONPLAN LENTUS response to a major maritime disaster involving a cruise ship. The framework constructed permits the evaluation of the operational value of existing and hypothetical infrastructure, such that a globally optimal set of infrastructure investments may be selected within a fixed budget.

13b. RÉSUMÉ (When available in the document, the French version of the abstract must be included here.)

L'équipe d'analyse et de recherche opérationnelle du Commandement des opérations interarmées du Canada a été chargée de soutenir le processus de sélection et d'implantation du Carrefour de soutien opérationnel septentrional. Pour ce faire, l'équipe développe une série de modèles d'optimisation représentant les aspects logistiques de la mise en œuvre des plans de circonstance (CONPLANs) nationaux dans le Nord canadien. Ce document de référence décrit le développement actuel d'une spécification mathématique du premier scénario, une réponse CONPLAN LENTUS à une catastrophe maritime majeure impliquant un navire de croisière. Le cadre élaboré permet d'évaluer la valeur opérationnelle des infrastructures existantes et hypothétiques, de sorte qu'un ensemble optimal d'investissements dans ces infrastructures peut être sélectionné dans les limites d'un budget fixe.