A Column Generation Algorithm for Evacuation Planning with Cycle-Free Paths

Mohd. Hafiz Hasan (0000-0002-1930-822X) and Pascal Van Hentenryck (Pascal’s ORCID)

1University of Michigan, Ann Arbor MI 48109, USA
{hasanm, pvanhent}@umich.edu

Abstract. Evacuation planning algorithms are critical tools for assisting authorities in orchestrating large-scale evacuations to ensure optimal utilization of resources. We introduce a column generation algorithm that produces zone-based evacuation plans that consist of non-preemptive evacuation schedules with evacuation paths that are guaranteed to be cycle-free. The algorithm builds upon a similar, previously introduced algorithm that was discovered to contain a critical shortcoming; it may produce paths containing cycles which are highly undesirable from an evacuation perspective. Instead of finding the shortest path in a time-expanded graph in the algorithm’s pricing subproblem, we propose finding one subject to a feasibility constraint which only allows paths that will eventually be cycle-free to be considered. We also suggest a hybrid algorithm to efficiently solve this new subproblem. Through experiments comparing results of both original and new algorithms when applied to a real-world evacuation scenario, we demonstrate that the new algorithm is able to produce provably high quality evacuation plans that match the quality of those produced by the original, while guaranteeing evacuation paths that are cycle-free only at the expense of increased CPU time consumption during its column generation phase.

Keywords: Column generation, Evacuation planning, K shortest paths, Mixed-integer programming, Constraint programming

1 Introduction

Large-scale evacuations are often necessary and critical to the preservation of safety and lives of residents in regions threatened by man-made or natural disasters like floods, hurricanes, and wildfires. According to a report by the International Federation of Red Cross and Red Crescent Societies [20], the first decade of the 21st century witnessed 7184 disasters around the world which affected a total of 2.55 billion people, accounted for the deaths of more than 1 million people, and incurred $986 billion in economic losses. Effective disaster management requires, among others, evacuation plans that ensure resources like transportation network capacity and time are not completely overwhelmed by evacuee demand. Traffic congestion and associated delays which usually result
from self-evacuations, in which individuals are given the freedom to choose their own evacuation routes, destinations, and times, significantly increase the risk of casualties being stranded in disaster-affected areas. Therefore, it is crucial for emergency authorities to be equipped with disaster management tools capable of generating and prescribing plans that guarantee optimal utilization of evacuation resources to attain specific goals like minimizing total evacuation time or maximizing number of evacuees reaching safety. Evacuation planning algorithms fulfill this need by producing a set of operational instructions for authorities to manage and orchestrate large-scale evacuations through specification of directions for evacuees, such as routes from their homes to designated safe destinations and accompanying departure times.

Pillac et al. [18] introduced such an algorithm called the column generation approach for joint mobilization and evacuation planning. The method, which we will refer to as the column generation algorithm from this point forth, is capable of generating provably high quality, zone-based evacuation plans with non-preemptive departure schedules using column generation. Unfortunately, we have discovered that its pricing subproblem which is solved by finding the shortest path in a time-expanded graph may generate evacuation paths that contain cycles, which are highly undesirable from an evacuation perspective as they route evacuees through unnecessary detours instead of directly to their safe destinations. In this paper, we propose an alternate pricing subproblem which seeks the feasible shortest path in the time-expanded graph to address this issue. We also propose several algorithms to solve this subproblem, and compare the performance of the column generation algorithm incorporating the new pricing subproblem against the original by applying them on a real-world evacuation scenario.

The rest of this paper is organized as follows. Section 2 outlines some related works, while Sec. 3 introduces notation and concepts used throughout the paper. Section 4 summarizes the original column generation algorithm proposed in [18], while Sec. 5 elaborates its shortcomings and details our proposed solution for addressing them. Section 6 describes our experimental setup and discusses results of the suggested modifications. Finally, Sec. 7 provides some concluding remarks.

2 Related Work

The importance of prescriptive evacuations, in which evacuation is orchestrated by a central authority as opposed to self-evacuation, is well recognized in the field, making it the focal point of various studies. Cova and Johnson [4] introduced a mixed-integer programming (MIP), lane-based evacuation routing model that generates routing plans that trades total vehicle travel distance against traffic merging while preventing traffic crossing at intersections. Lu et al. [15] proposed two heuristic algorithms which model capacity as a time series and produce single and multiple capacity constrained routes respectively with total evacuation times that are within 10% of optimal solutions. They then introduced an improved heuristic based on similar ideas in [14] that reduces the run time
of previous algorithms but still produces high quality, suboptimal solutions. Liu et al. [13] presented a many-to-one cell transmission model (CTM [5]) of a prescriptive dynamic traffic assignment problem and a heuristic algorithm called HASTE to approximate an optimal solution to the problem. Miller-Hooks and Sorrel [16] suggested a metaheuristic based on the principles of noisy genetic algorithms to solve the Maximal Dynamic Expected Flows Problem which seeks paths and associated flows to maximize the number of evacuees reaching safety within a fixed time bound in a network where arc capacities and traversal times are represented with discrete random variables with time-varying distribution functions.

Bretschneider and Kimms [3] introduced a pattern-based dynamic network flow model that restructures traffic routes during evacuation scenarios and a corresponding two-stage heuristic approach to solve the problem. The first stage solves traffic routing and vehicle flow in a street network that disregards intersection details, while the second calculates traffic routing and flow in a more detailed network which models every entrance, exit, and all possible turnings within intersections. Bish and Sherali [2] structures evacuation demand using an aggregate-level staging and routing strategy. They employed a generalized CTM framework to model traffic flow, utilized a lexicographic objective function that preemptively minimizes clearance time, total duration of all evacuation paths, and average evacuation time in that priority order, and designed two heuristics to produce high quality solutions to the problem. Bish et al. [1] takes different evacuee types based on their safe shelter requirements and shelter capacity restrictions into consideration. They suggest a MIP planning model combined with CTM that accounts for traffic flow dynamics and congestion to optimally stage and route evacuees so that network clearance time is minimized.

Even though emergency authorities typically prefer zone-based evacuations which necessitates all residents from the same residential zone be assigned a single evacuation route to avoid confusion and increase compliance, only a few works have proposed plans that satisfy these requirements. Huibregtse et al. [10] developed a two-phase optimization method that accommodates assumptions of either partial or full instruction compliance, in which the first phase constructs a search space containing feasible evacuation routes and departure times while the second applies an ant colony optimization algorithm to assign routes and departure times to each evacuation zone. The same authors in [9] studied the effects of road blocking as an intervention measure to increase instruction compliance and obtained 10% and 13% improvements in evacuation efficiency on the two cases they tested. Even et al. [7] took a different approach by introducing the Convergent Evacuation Planning Problem (CEPP) which aims to remove forks from all evacuation routes combined to eliminate delays caused by diverging and crossing routes. They proposed a two-stage approach to solve the problem which separates route design and evacuation scheduling. Their work served as the foundation of the algorithm by Romanski and Van Hentenryck [21] which solves CEPP using a Benders decomposition based approach. Pillac et al. [19] suggested a conflict-based path-generation heuristic which decomposes an
evacuation planning problem into a subproblem which generates new evacuation paths for each evacuated area and a master problem which selects paths and schedules flow of evacuees over them.

3 Notation and Preliminaries

Figure 1 shows an example of the evacuation scenario that is addressed by the column generation algorithm. Square 0 represents an evacuation node (e.g. a residential zone), triangles A and B represent safe nodes (final evacuation destinations), circles 1–3 represent transit nodes (road intersections), and arcs represent roads connecting the nodes. Times on each arc indicate when each road will become unavailable (e.g. due to being flooded) and the time on the evacuation node indicates the final deadline by which it must be evacuated. In this example, the evacuation deadline for node 0 is 13:00 since its last outgoing arc will be blocked at that time.

The evacuation scenario can be abstracted by a static evacuation graph \( G = (N = E \cup T \cup S, A) \) where \( E \), \( T \), and \( S \) are the set of evacuation, transit, and safe nodes respectively, and \( A \) is the set of all arcs. Each evacuation node \( k \in E \) has associated with it a demand \( d_k \) representing the number of vehicles to be evacuated and an evacuation deadline \( f_k \), while each arc \( e \in A \) has associated with it a travel time \( s_e \), a capacity \( u_e \) in terms of vehicles per unit time, and a block time \( f_e \) at which the road becomes unavailable. Figure 2 shows the static evacuation graph for the scenario of Fig. 1. The evacuation node is labeled with its demand and evacuation deadline while the arcs are labeled with their travel time, capacity, and block time. Also note that the evacuation node has no incoming arcs and the safe nodes have no outgoing arcs.

In order to reason about traffic flows over time, the static graph is converted into a time-expanded graph \( G^t = (N^t = E^t \cup T^t \cup S^t, A^t) \). The conversion is performed by first discretizing the time horizon \( \mathcal{H} \) into time steps of identical length \( t \), creating a copy of all nodes at each time step, and replacing each arc \( e = (i, j) \) with corresponding arcs \( e_t = (i_t, j_{t+s_e}) \) for each time step that \( e \) is available. Figure 3 shows the time-expanded graph constructed from the static graph of Fig. 2, in which each arc is labeled with its capacity. Infinite capacity
arcs are introduced connecting the evacuation and safe nodes at each time step to allow modeling of evacuees waiting at those nodes. Nodes that cannot be reached from either the evacuation or safe nodes within the time horizon are removed from the graph (they are greyed out in Fig. 3).

The algorithm also adopts the concept of response curves [17]. In this paper, a response curve $f$ is essentially a function that models the flow of vehicles over time. The number of evacuees departing from evacuation node $k$ at time $t$ beginning from evacuation start time $t_0$, $D_k(t, t_0)$, is defined using a selected response curve $f$ as follows:

$$D_k(t, t_0) = \begin{cases} 
0 & \text{if } t < t_0 \\
 f(t - t_0) & \text{if } t \geq t_0 
\end{cases} \quad (1)$$

$D_k(t, t_0)$ can be used to precisely specify a non-preemptive evacuation schedule for evacuation node $k$. Figure 4 shows $D_k(t, t_0)$ utilizing four different types of response curves. The S-shape, Rayleigh and inverse Rayleigh curves use $t_0 = 60$ minutes, while the step function uses $t_0 = 120$ minutes. The step response curve, in which evacuees depart at a constant rate after $t_0$ until a region is completely evacuated is the type primarily used in the algorithm.

An evacuation plan is defined to contain the following two components: (a) a set of evacuation paths, each represented by a sequence of connected nodes in the static graph from an evacuation node to a safe node, specifying the route to be taken by residents of each evacuation node to reach safety, and (b) a set of evacuation schedules indicating the number of vehicles that need to depart from each evacuation node at each time step $t \in H$. The Evacuation Planning Problem (EPP) is defined as a problem formulation that produces an evacuation plan that maximizes the number of evacuees reaching safety.
Figures 3 and 3b illustrate, for four types of response curves, the departure rate of evacuees and the cumulative number of departed evacuees over time, assuming an evacuation order issued at 60 minutes. The step function response curve considers a lead time of 60 minutes before evacuees start departing and then assumes a constant rate until all evacuees have departed. The S-shape is a commonly used response curve described by a logistic function $f_S(t) = D_1 + e^{-kt}$, where $D$ and $k$ are parameters. The Rayleigh response curve is defined by the function $f_R(t) = t\sigma^2 e^{-t^2/2\sigma^2}$, where $\sigma$ is a parameter.

3 Related work

According to Hamacher and Tandra [12], evacuation planning can be tackled using either microscopic or macroscopic approaches. Microscopic approaches focus on modeling and simulating the evacuees’ individual behaviors, movements, and interactions. Macroscopic approaches, such as the one presented in this study, aggregate evacuees and model their movements as a flow in the evacuation graph.

3.1 Macroscopic level

To the best of our knowledge, only a handful of studies attempt to design evacuation plans that define both a set of evacuation routes and a departure schedule. Huibregtse et al. [16] propose a two-stage algorithm that first generates a set of evacuation routes and feasible (a) Departure Rates. (b) Cumulative Departures.

Fig. 3: Comparison of response curves

Fig. 4: The number of evacuees departing an evacuation node $k$ as a function of time, $D_k(t, t_0)$, using four different types of response curves (from [18]).

4 Column Generation Algorithm

In this section, we summarize the column generation algorithm introduced in [18] to solve the EPP. It generates an evacuation plan with paths that satisfy a zone-based evacuation constraint, in which all evacuees from an evacuation node are assigned the same evacuation path. It also produces non-preemptive evacuation schedules which differ from preemptive ones in that they do not contain flow interruptions across the time horizon, which may make them easier to enforce.

As its name suggests, the algorithm uses column generation [6], which is a method for solving large-scale linear programs (LPs). It does so by iteratively solving a master problem (MP) containing only a subset of all variables. A pricing subproblem (PSP) introduces additional variables into the MP that can improve its objective value. A negative reduced cost criteria is used to identify new variables that would enter the basis of the MP’s optimal solution. According to LP theory [6], the solution of the MP is optimal when no new variables with negative reduced cost can be found.

When column generation is used to solve the linear relaxation of an integer program (IP), the solution of the final iteration MP provides a lower bound to the optimal solution of the IP (for a minimization problem). Solving the final iteration MP as an IP provides an upper bound, and its distance from the optimal solution can be gauged by calculating the duality gap which is the relative difference between the upper and lower bounds. This is the overall approach adopted by the column generation algorithm.

4.1 Master Problem Formulation

The MP is a formulation intended to select time-response evacuation plans from a subset of feasible plans $\Omega'$ that would maximize the number of evacuees reaching safety and minimize total evacuation time. A time-response evacuation plan, $p = (k, f, P, t_0)$, consists of an evacuation node $k$, a path $P$, a response curve $f \in F$ (where $F$ is a set of predefined response curves), and an evacuation start time $t_0$. The MP uses a binary variable $x_p$ which indicates whether a plan $p \in \Omega'$
should be selected. In the formulation, \( c_p \) denotes the cost of plan \( p \), \( \Omega_k \subseteq \Omega' \) is the subset of plans for evacuation node \( k \), \( \omega(e) \subseteq \Omega' \) is the subset of plans that utilize arc \( e \), \( a_{p,e,t} \) denotes the flow of evacuees along arc \( e \) at time \( t \) induced by plan \( p \), \( \alpha_p \) is the amount of mobilization resources required to execute plan \( p \), and \( Q \) is the total amount of mobilization resources available.

\[
\min \sum_{p \in \Omega'} c_p \cdot x_p \tag{2}
\]

subject to

\[
\sum_{p \in \Omega_k} x_p = 1 \quad \forall k \in \mathcal{E} \tag{3}
\]

\[
\sum_{p \in \omega(e)} a_{p,e_t} \cdot x_p \leq u_{e_t} \quad \forall e \in \mathcal{A}, \forall t \in \mathcal{H} \tag{4}
\]

\[
\sum_{p \in \Omega'} \alpha_p \cdot x_p \leq Q \tag{5}
\]

\[
x_p \geq 0 \quad \forall p \in \Omega' \tag{6}
\]

The cost of plan \( p \), \( c_p \), is defined as a function that applies a linear penalty on the arrival time of evacuees at the safe node and heavily penalizes casualties, i.e. the number of evacuees that cannot reach safety. More precisely, \( c_p \) is defined as:

\[
c_p = \sum_{e \in \mathcal{P}} \sum_{t \in \mathcal{H}} c_{e_t} \cdot a_{p,e_t} + \bar{c} \cdot \bar{a}_p
\]

where \( \bar{a}_p \) denotes the number of casualties that would result from execution of plan \( p \), and \( c_{e_t} \) and \( \bar{c} \) are defined as follows:

\[
c_{e_t} = c_{(i,j)}_t = \begin{cases} \frac{1}{|\mathcal{S}|} & \text{if } j \in \mathcal{S} \\ 0 & \text{otherwise} \end{cases} \tag{8}
\]

\[
\bar{c} = 100 \max_{e \in \mathcal{A}, t \in \mathcal{H}} \{c_{e_t}\} \cdot \max_{k \in \mathcal{E}} \{d_k\} \tag{9}
\]

\( c_p \) essentially causes objective function (2) to be lexicographic; the MP becomes a multi-objective problem that minimizes of the number of casualties first followed by the total evacuation time.

Constraint (3) ensures only one plan is selected for each evacuation node \( k \), (4) enforces all arc capacities, and (5) enforces the availability of mobilization resources. The formulation shown is a linear relaxation of the original MP; \( x_p \) is defined to be continuous to allow application of the column generation procedure. Therefore, once the column generation procedure has terminated, the final MP needs to be solved as an IP by making \( x_p \) binary to obtain the solution of the original MP.
4.2 Reduced Cost Formulation

The algorithm progressively introduces new variables into the MP that can improve its objective value, and such a variable $x_p$ must satisfy the criteria of having negative reduced cost, $r_p$, which is defined as:

$$ r_p = c_p - \pi^T a_p < 0 $$

(10)

where $\pi$ is the vector of dual values from the optimal solution of the MP and $a_p$ is the column of constraint coefficients of $x_p$. Letting $\{\pi_k\}$, $\{\pi_e\}$, and $\pi$ denote the dual variables of (3), (4), and (5) respectively, and substituting (7) into (10), the formula for $r_p$ can be rewritten as:

$$ r_p = \sum_{e \in P} \sum_{t \in \mathcal{H}} c_{et} \cdot a_{p,et} + \bar{c} \cdot \bar{a}_p - \pi_k - \pi \cdot \alpha_p = \sum_{e \in P} \sum_{t \in \mathcal{H}} \pi_e \cdot a_{p,et} $$

$$ = -\pi_k - \pi \cdot \alpha_p + \bar{c} \cdot \bar{a}_p + \sum_{e \in P} \sum_{t \in \mathcal{H}} (c_{et} - \pi_e) \cdot a_{p,et} $$

(11)

4.3 Pricing Subproblem Formulation

The PSP is responsible for identifying a new time-response evacuation plan $p$ that satisfies the negative reduced cost criteria. The key to formulating the PSP is to recognize some key characteristics of the expanded $r_p$ formulation (11): (a) It contains terms which are specific to a single $p$, and since time-response evacuation plans are independent of each other, the PSP can be solved independently for each evacuation node $k \in \mathcal{E}$ and for each predefined response curve $f \in \mathcal{F}$, allowing multiple PSPs to be solved concurrently in parallel. (b) Term $\alpha_p$ is solely dependent on a selected response curve $f$ (hence it is fixed given an $f$). Therefore, finding a $p$ with negative reduced cost is equivalent to finding an evacuation path $P$ and evacuation start time $t_0$ combination that minimizes the last two terms of (11) for each $k \in \mathcal{E}$ and for each $f \in \mathcal{F}$. We denote the last two terms of (11) as Cost($P, t_0$):

$$ \text{Cost}(P, t_0) = \bar{c} \cdot \bar{a}_p + \sum_{e \in P} \sum_{t \in \mathcal{H}} (c_{et} - \pi_e) \cdot a_{p,et} $$

(12)

The path $P$ and evacuation start time $t_0$ that minimizes Cost($P, t_0$) can be obtained from an extended time-expanded graph $\mathcal{G}^x$ with carefully defined arc costs. The extension involves introduction of a virtual super-sink, $v_t$, which all safe nodes $s \in \mathcal{S}^x$ are connected to with arcs $e_t \in \mathcal{A}^x = \{(s, v_t) \mid s \in \mathcal{S}^x\}$. Next, we let $\mathcal{A}^x_\infty$ denote the set of all infinite capacity arcs used to model evacuees waiting at the evacuation nodes. Then for a given evacuation node $k$ and response curve $f$, a path $P^x$ in $\mathcal{G}^x$ from evacuation node $k_0$ (evacuation node $k$ and time 0) to $v_t$ corresponds to a time-response evacuation plan $p = (k, f, P, t_0)$, where $P$ is given by the sequence of nodes visited by $P^x$ excluding $v_t$ and $t_0$ is given by the time of the first non-waiting arc leaving $\mathcal{E}^x$. For instance, path $P^x$ represented by
the red colored arcs in Fig. 5 corresponds to path $P = \langle 0, 1, A \rangle$ and evacuation start time $t_0 = 10:00$.

The cost of this path and evacuation start time combination, $Cost(P, t_0)$, can be calculated by first assigning arc costs $c_{et}^{sp}$ to each arc $e_t \in A^x$ as follows:

$$c_{et}^{sp} = \sum_{t' = t}^{[|H|]} (c_{e_{t'}} - \pi_{e_{t'}}) \cdot f(t' - t) \quad \forall e_t \in A^x \setminus (A^0_w \cup A^0_x)$$ (13)

$$c_{et}^{sp} = \bar{c} \cdot (d_k - F(|H| - t)) \quad \forall e_t \in A^0_x$$ (14)

$$c_{et}^{sp} = 0 \quad \forall e_t \in A^0_w$$ (15)

Equation (13) aggregates future costs of arc $e_t$ should it be selected for a time-response evacuation plan, while (14) accounts for the cost of casualties for time-response evacuation plans that end with that arc.

With these arc cost definitions, $Cost(P, t_0)$ for a path $P$ and evacuation start time $t_0$ combination that corresponds to a path $P^x$ can be calculated using (16):

$$Cost(P, t_0) = \sum_{e_t \in P^x} c_{et}^{sp}$$ (16)

$$= \bar{c} \cdot (d_k - F(|H| - t)) + \sum_{e_t \in P^x \setminus A^0_x} \sum_{t' = t}^{[|H|]} (c_{e_{t'}} - \pi_{e_{t'}}) \cdot f(t' - t)$$ (17)

$$= \bar{c} \cdot \bar{a}_p + \sum_{e_t \in P^x} (c_{e_t} - \pi_{e_t}) \cdot a_{p,e_t}$$ (18)

Equations (17) and (18) show that expansion of (16) will eventually lead to the original equation for $Cost(P, t_0)$ in (12).

Therefore, the goal of the PSP, which is to find a path $P$ and evacuation start time $t_0$ combination that minimizes $Cost(P, t_0)$, can be accomplished by finding a shortest path from $k_0$ to $v_t$ in the extended time-expanded graph for each $k \in \mathcal{E}$ and $f \in \mathcal{F}$. This allows a shortest path algorithm such as the Bellman-Ford algorithm to be applied to solve the PSP in polynomial time.

![Fig. 5: Path $P^x$ in the extended time-expanded graph.](image)
5 Cycle-Free Evacuation Path Extension

The time-expanded graph $G^x$ is by construction acyclic as its arcs only connect nodes from a time step to another in the future. Therefore, shortest paths identified in the PSP will also be acyclic. However, this fact does not preclude the PSP from generating paths that visit the same transit node in $G^x$ at different time steps, as there are no restrictions enforced in the shortest path algorithms used preventing such paths from being generated. While such paths are acyclic in $G^x$, when they are converted to their corresponding counterparts in the static graph $G$, the static path equivalents will contain cycles as they visit the same transit node more than once. Figure 6 shows an example of one such path in $G$ that is generated when the algorithm is applied on a real-world evacuation scenario (more details on this scenario are provided in Sec. 6). This presents a problem as such paths are highly undesirable during evacuations as they would result in evacuees making unnecessary detours instead of going straight to their safe destinations. Our initial experiments on the real-world scenario reveal that approximately 44% of the paths generated by the column generation procedure has this undesirable property, with some ending up being selected by the final IP resulting in an evacuation plan that contains paths with cycles.

To address this issue, we introduce the concept of feasible evacuation paths. A feasible path $P_f^i$ is simply a path in $G$ from an evacuation node to a safe node that is cycle-free. It corresponds to a feasible path $P_{xf}^i$ in $G^x$ that can only visit each transit node in $G^x$ at most once. In other words, $P_{xf}^i$ may not visit the same transit node in $G^x$ at multiple time steps. Instead of simply finding the shortest path in $G^x$ as was proposed by [18], our new PSP now entails finding a

![Fig. 6: Red arrows represent a path originating from an evacuation node (grey square in top left) to a safe node (green triangle in bottom right) containing a cycle (circled in blue) which constitutes an unnecessary detour in the route.](image-url)
feasible shortest path in $G^x$. This problem falls into a class called shortest path problems with resource constraints [11] which are known to be NP-hard [8]. In this problem, the resources are unit “visited” resources associated with each transit node in $G^x$. The set of a particular transit node across all time steps in $G^x$, $\{i_t | i_t \in G^x, i \in T, \forall t \in H\}$, is allocated only one unit of this “visited” resource which is completely consumed if a node from the set were to be visited by path $P^x$. In the following sections, we outline several methods to solve this problem, namely: (a) a $K$-shortest-path-based (KSP) algorithm, (b) a MIP method, (c) a constraint programming (CP) method, and (d) a hybrid algorithm.

5.1 $K$-Shortest-Path-Based Algorithm

The KSP algorithm finds the feasible shortest path in $G^x$ by iteratively enumerating shortest paths in increasing order of length until one that satisfies the feasibility condition is found. Its key component is an implementation of the Recursive Enumeration Algorithm (REA) [12]. REA computes $K$ shortest paths from a source $s$ to a sink $t$ in a graph $G = (N, A)$, where $N$ is the set of all nodes and $A$ is the set of all arcs, in $O(|A| + K|N| \log(|A|/|N|))$ time by recursively solving a set of equations which generalize the Bellman equations for the single shortest path problem.

In the following summary of the REA, $\Gamma^{-1}(j)$ is the set of predecessors of node $j$. $P_k(j)$ denotes the $k$th shortest path from source $s$ to node $j$, while $P_k(i;j)$ denotes a path from $s$ to $j$ formed by appending arc $(i, j)$ to the end of path $P_k(i)$. $C_k(j)$ is the set of candidates for the next shortest path from $s$ to $j$ from which $P_k(j)$ can be chosen, and $L(P)$ denotes the length of path $P$. The REA to compute $K$ shortest paths from $s$ to $t$ in $G = (N, A)$ is outlined in Algorithm 1. It relies on a NextPath($j, k$) procedure which uses recursion to generate $P_k(j)$ using information from $P^1(j), P^2(j), \ldots, P^{k-1}(j)$. This procedure is outlined in Algorithm 2.

NextPath($j, k$) first recursively constructs sets of candidates for the next shortest path from $s$ to each node $j$ in the graph, $C_k(j) \forall j \in N$. It does so by considering paths formed by the next shortest paths to the predecessors of $j$ appended with the arc connecting each predecessor to $j$. $P^k(j)$ is then selected as the shortest path from $C_k(j)$ if $C_k(j)$ is not an empty set, else $P^k(j)$ does not exist. Since REA does not require the desired number of paths $K$ to be fixed a priori, it is used in the KSP algorithm to enumerate shortest paths until the first

---

Algorithm 1 Recursive Enumeration Algorithm

1: Compute $P^1(j) \forall j \in N$ using any algorithm (e.g. Bellman-Ford)
2: $k \leftarrow 1$
3: repeat
4: $\quad k \leftarrow k + 1$
5: $\quad P^k(t) \leftarrow \text{NextPath}(t, k)$
6: until $k = K$ or $P^k(t)$ does not exist
Algorithm 2 Next Shortest Path Algorithm

1: procedure NextPath($j, k$)
2:     if $k = 2$ then
3:         $C^k(j)$ ← $\{P^i(i) \cdot j \mid i \in \Gamma^{-1}(j)\} \setminus P^1(j)$
4:     if $j = s$ and $k = 2$ then
5:         goto 11
6:     Let $i$ be a node and $k' = 0$ be an index such that $P^{k'+1}(i) \cdot j = P^{k-1}(j)$
7:     if $P^{k'+1}(i)$ has not been computed then
8:         $P^{k'+1}(i)$ ← NextPath($i, k'+1$)
9:     if $P^{k'+1}(i)$ exists then
10:        $C^k(j)$ ← $C^k(j) \cup P^{k'+1}(i) \cdot j$
11:    if $C^k(j) \neq \emptyset$ then
12:        $P^k(j)$ ← $\arg \min_{P \in C^k(j)} L(P)$
13:        $C^k(j)$ ← $C^k(j) \setminus P^k(j)$
14:    else
15:        $P^k(j)$ does not exist

one that satisfies the feasibility condition is obtained. The only shortcoming of
this method is that our experiments have shown that there are PSP instances
that require an extremely large amount of shortest paths to be enumerated
$(K > 10^6)$ before a feasible one is found, and in some instances, $K$ is so huge
that our machine runs out of memory before the algorithm terminates.

5.2 Mixed-Integer Programming Method

The shortcoming of the KSP algorithm compelled us to conceive a MIP formula-
tion for finding the feasible shortest path in $G^*$. In the formulation of (19)–(24),
we use a binary decision variable $x_{e_i}$ to indicate whether arc $e_i$ is selected for
the feasible shortest path, $k_0$ to denote the evacuation node of interest $k$ at time
0, and $\delta^-(i)$ and $\delta^+(i)$ to denote the set of incoming and outgoing arcs of node
$i$ respectively.

Objective function (19) specifies that the objective of the problem is to mini-
mize total cost of the path. Constraint (20) specifies that exactly one path should emanate from source node $k_0$, while (22) ensures the path ends at super sink
node $v_t$. Constraint (21) enforces path continuity at every node other than the
source and super sink. Finally, (23) guarantees that each transit node is visited
by the path at most once throughout the entire time horizon, which corresponds
to the feasibility condition. Our experiments show that this method does not
suffer from out of memory issues encountered by the KSP algorithm under any
circumstances.

$$\min \sum_{e_t \in A^*} c_{e_t} x_{e_t}$$
subject to
\[
\begin{align*}
\sum_{e_i \in \delta^+(k_0)} x_{e_i} &= 1 \\
\sum_{e_i \in \delta^-(i)} x_{e_i} - \sum_{e_i \in \delta^+(i)} x_{e_i} &= 0 \quad \forall i \in \mathcal{N}^x \setminus \{k_0, v_t\} \\
\sum_{e_i \in \delta^-(v_t)} x_{e_i} &= 1 \\
\sum_{i \in \mathcal{H}} \sum_{e_i \in \delta^+(i)} x_{e_i} \leq 1 \quad \forall i \in \mathcal{T} \\
x_{e_i} &\in \{0, 1\} \quad \forall e_i \in \mathcal{A}^x
\end{align*}
\]

5.3 Constraint Programming Method

We also considered CP in finding the feasible shortest path in \(G^x\). Our formulation uses a set of floating point variables \(\text{cost}_{\text{fsp}}[i]\) to store the cost of the feasible shortest path from \(k_0\) to each node \(i \in \mathcal{N}^x\) and a set of boolean variables \(\text{absent}[i]\) to indicate whether node \(i \in \mathcal{N}^x\) is absent or present in the feasible shortest path from \(k_0\) to \(v_t\). We use \(c_{(i,j)}^{\text{sp}}\) to denote the cost of arc \((i, j) \in \mathcal{A}^x\).

\[
\begin{align*}
\min \text{cost}_{\text{fsp}}[v_t] \\
\text{subject to} \\
\text{absent}[v_t] &= 0 \\
\text{cost}_{\text{fsp}}[k_0] &= 0 \\
M &= 10 \cdot \max_{(i,j) \in \mathcal{A}^x} \{c_{(i,j)}^{\text{sp}}\} \\
\text{cost}_{\text{fsp}}[j] &= \min_{i \in \mathcal{P}^{-1}(j)} \left\{ \text{cost}_{\text{fsp}}[i] + c_{(i,j)}^{\text{sp}} + M \cdot \text{absent}[i] \right\} \\
\sum_{i \in \mathcal{P}^{-1}(j)} \text{absent}[i] &\geq |\mathcal{P}^{-1}(j)| - 1 \quad \forall j \in \mathcal{N}^x \setminus \{k_0, v_t\} \\
\sum_{i \in \mathcal{H}} \text{absent}[i] &\geq |\{i \mid \forall t \in \mathcal{H}\}| - 1 \quad \forall i \in \mathcal{T} \\
\text{cost}_{\text{fsp}}[i] &\in \mathbb{R} \quad \forall i \in \mathcal{N}^x \\
\text{absent}[i] &\in \{0, 1\} \quad \forall i \in \mathcal{N}^x
\end{align*}
\]

Objective function (25) specifies that the objective is to minimize cost of the feasible shortest path from \(k_0\) to \(v_t\). Constraint (26) states that \(v_t\) is present in the path, and (27) initializes the cost of the path ending at \(k_0\) to be 0. Equation (28) assigns a really large value to \(M\), while (29) sets cost of the path to node \(j\) equal to the smallest cost of the path to one of its predecessors \(i\) plus the cost of arc \((i, j)\). The last term in the expression, \(M \cdot \text{absent}[i]\), is used to keep track of predecessor \(i\) that led to the smallest cost for the path to \(j\) by making \(i\) present
in the path. Constraint (30) specifies that for each node \( j \), at most only one of its predecessors can be present in the path. Finally, (31) enforces the feasibility condition by ensuring that each transit node can be present at most once in the path throughout the time horizon. Similar to the MIP method, this method does not suffer from out of memory issues.

5.4 Hybrid Algorithm

We conducted a preliminary experiment to evaluate the efficacy of the KSP, MIP, and CP methods by incorporating the new PSP solved by each algorithm into the column generation procedure and applying it on a real-world evacuation scenario, HN80-I1.0 (details on this scenario are presented in Sec. 6). The time spent solving each PSP is recorded together with the \( K \) value required by the KSP algorithm to find the feasible shortest path. Average solve times of each algorithm are shown in Fig. 7. Since the KSP algorithm encounters out of memory issues for large \( K \) values, the algorithm was terminated when \( k \) went beyond \( 10^6 \), which explains why Fig. 7 only includes comparisons for \( K \) ranges up to \( 10^6 \). It can be seen in the figure that the KSP algorithm is consistently the fastest across the various ranges of \( K \) values tested, however its average solve time experiences a sharp increase for \( K > 10^5 \). The MIP method is second fastest, although its time gap with the KSP algorithm narrows significantly for \( K > 10^5 \), and the CP method is the slowest.

Figure 8 shows the fraction of PSP instances solved for each range of \( K \) values (out of a total of 37200 problem instances). The figure indicates that the majority of PSP instances are solvable by the KSP algorithm using small \( K \) values, and approximately 10% of them require \( K > 10^6 \). These observations inspired us to conceive a hybrid algorithm which combines the KSP algorithm and the MIP method. The algorithm first enlists the KSP algorithm to find the feasible shortest path in a PSP up to a fixed \( k \)-threshold. If the KSP algorithm’s \( k \) value reaches this threshold and a feasible path is yet to be found, it is terminated, after which the MIP method is invoked. This strategy lets us harness the speed benefits of the KSP algorithm and also addresses its shortcoming by falling back to the MIP method on more challenging PSPs. Since the majority of PSP instances can be solved by the KSP algorithm with small \( K \) values, we are assured that the hybrid algorithm has to resort to the more expensive MIP method only in a small fraction of cases if the \( k \)-threshold is set appropriately. For instance, average solve times of the hybrid algorithm using a \( k \)-threshold of \( 10^5 \) for scenario HN80-I1.0 are shown in Fig. 7. As expected, its average solve times match those of the KSP algorithm when \( k \) does not exceed the threshold, and they are in the MIP territory when the threshold is exceeded.

6 Experimental Results

We evaluated the column generation algorithm with the feasible shortest path PSP by applying it on a real-world evacuation scenario. It is used to develop
evacuation plans for the Hawkesbury-Nepean (HN) region located north-west of Sydney, Australia. The region’s evacuation graph consists of 80 evacuation nodes, 184 transit nodes, 5 safe nodes, and 580 arcs. The region has a total of 38343 vehicles to be evacuated in its base instance. We also investigated the effect of increasing the population size by linearly scaling the base instance by factor $x \in [1.0, 3.0]$. We will refer to this scenario as HN80-Ix from this point forth with $x$ representing the population scaling factor.

The algorithm is assigned a fixed time horizon of $H = 10$ hours discretized into 5 minute time steps. The set of response curves $\mathcal{F}$ is populated with step response curves with flow rates $\gamma \in \{2, 6, 10, 25, 50\}$ vehicles per time step. The mobilization resource constraint in (5) is relaxed by setting $Q = +\infty$ as was done in [18]. The algorithm is implemented in C++ with Gurobi 6.5.2 being invoked to solve all LPs and MIPs. In each iteration of the column generation procedure, feasible shortest paths for each evacuation node $k \in \mathcal{E}$ and each response curve $f \in \mathcal{F}$ are calculated in parallel using OpenMP. Cplex CP Optimizer 12.6.2 was used to solve CP formulations in experiments involving the CP method. All experiments were conducted on a high-performance computing cluster, with each utilizing 8 cores of a 2.5 GHz Intel Xeon E5-2680v3 processor and 64 GB of RAM. CPU time limits of 96 hours and 24 hours are imposed on the column generation phase and the MP’s final iteration in which it is solved as an IP respectively.

In order to identify a suitable $k$-threshold for the hybrid algorithm, we first ran only the column generation phase with $k$-threshold $\in \{10^3, 10^4, 10^5, 10^6\}$ on HN80-Ix instances with $x \in \{1.0, 1.1, 1.2, 1.4\}$, and recorded the total time spent solving only the PSP in each iteration. Figure 9 shows the average per iteration PSP solve times for each instance and each $k$-threshold tested with error bars corresponding to standard errors. It can be seen for instances HN80-I1.0 and
HN80-I1.2, the differences in average solve times across various threshold values are not statistically significant. However, in the other instances, a $k$-threshold of $10^5$ resulted in either the smallest or among the smallest average solve times. Therefore, this threshold value is used for the hybrid algorithm in all subsequent experiments.

Figure 10 provides a closer look at the total time spent on solving the PSP in each iteration of the column generation phase of instance HN80-I1.0. 400 instances of the PSP are solved in each iteration. In earlier iterations, the hybrid algorithm is able to find feasible shortest paths without having its $k$-threshold exceeded, which explains its short solve times as the fast KSP algorithm is being primarily utilized. However as the iterations increase, so does the number of PSP instances that exceed the $k$-threshold, leading to more instances being solved using the more expensive MIP method and correspondingly increased solve times. The original shortest path PSP does not experience this time increase as the polynomial time Bellman-Ford algorithm is being used in all iterations.

Figures 11 and 12 illustrate the evolution of the algorithm’s objective value over time with the original shortest path PSP and the new feasible shortest path PSP respectively. The key takeaway is that both formulations result in the same convergence characteristic, producing the same final objective value in the column generation phase. In fact, the only major difference is that the original formulation’s column generation phase converged earlier, which is to be expected since its PSP is solved using a polynomial time algorithm. The final IP of both formulations reached their 24 hours time limit, producing approximately the same final objective value and duality gap.

Complete results of the column generation phase of the algorithm with original and new PSP formulations on all HN80-Ix instances are summarized in Table 1. It shows the number of iterations, number of columns generated, final objective values, and CPU time consumed during the column generation phase of both algorithms. The most important result is that the new formulation produces final objective values which are the same as the original in almost all instances. This
implies that no quality is lost in the optimal solution of the column generation

Fig. 10: Total time spent on solving the new PSPs per iteration increases steadily with the number of iterations.

Fig. 11: Evolution of solution quality over time of algorithm with original shortest path PSP on the HN80-I1.0 instance.

Fig. 12: The algorithm incorporating the new feasible shortest path PSP exhibits similar convergence characteristics to that using the original PSP.
Table 1: Results of column generation phase using original shortest path and new feasible shortest path PSP formulations.

<table>
<thead>
<tr>
<th>Instance</th>
<th>Original Shortest Path PSP</th>
<th></th>
<th>New Feasible Shortest Path PSP</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Iter</td>
<td>Column</td>
<td>Final obj. val.</td>
<td>CPU time (mins)</td>
</tr>
<tr>
<td>HN80-I1.0</td>
<td>79</td>
<td>12251</td>
<td>8816</td>
<td>39</td>
</tr>
<tr>
<td>HN80-I1.1</td>
<td>104</td>
<td>15072</td>
<td>10405</td>
<td>136</td>
</tr>
<tr>
<td>HN80-I1.2</td>
<td>229</td>
<td>22571</td>
<td>12116</td>
<td>799</td>
</tr>
<tr>
<td>HN80-I1.4</td>
<td>152</td>
<td>20184</td>
<td>15935</td>
<td>690</td>
</tr>
<tr>
<td>HN80-I1.7</td>
<td>178</td>
<td>21871</td>
<td>22635</td>
<td>1312</td>
</tr>
<tr>
<td>HN80-I2.0</td>
<td>197</td>
<td>31418</td>
<td>30490</td>
<td>5760</td>
</tr>
<tr>
<td>HN80-I2.5</td>
<td>121</td>
<td>22806</td>
<td>46189</td>
<td>5760</td>
</tr>
<tr>
<td>HN80-I3.0</td>
<td>132</td>
<td>31726</td>
<td>1.96 × 10^1</td>
<td>5760</td>
</tr>
</tbody>
</table>

phase when the new formulation is used. The only instance where the values are not equal, HN80-I2.5, is due to the phase being terminated before convergence as its 96 hours CPU time limit was reached. Even then the values from both formulations are very close. CPU time consumption by the new formulation is consistently higher in almost all instances, which is expected since the new PSP involves solving an NP-hard problem. Even though the hybrid algorithm is very efficient at identifying feasible shortest paths, it cannot compete with the polynomial time Bellman-Ford algorithm used in the original PSP. Another interesting observation is that for instances where the phase converged before its CPU time limit was reached (x ∈ [1.0, 1.7]), the new formulation converges in equal or less number of iterations and produces less number of columns than the original.

Results of the final IP are summarized in Table 2 for both original and new formulations. It shows the duality gap, evacuation percentage, evacuation end time, and total CPU time (column generation time plus final IP time) consumed by both formulations on all HN80-Ix instances. The duality gap is calculated using \( \frac{z_{\text{MP,IP}} - z_{\text{MP,LP}}}{z_{\text{MP,IP}}} \) where \( z_{\text{MP,LP}} \) and \( z_{\text{MP,IP}} \) are the final objective values of the MP LP and MP IP respectively, whereas the evacuation end time is simply the time at which the last evacuee reaches its safe destination. Firstly, it must be noted that the 24 hours time limit for the final IP was reached in all instances of both formulations. Aside from the obvious CPU time advantage held by the original formulation, the results across both formulations are approximately even. Both formulations attained 100% evacuation in the first four instances, producing duality gaps of less than 15% in the first three. Both formulations had 100% duality gaps for x ∈ [1.7, 2.5], which can attributed to the huge penalty incurred in the objective value of the final IP due to not being able to evacuate every-
Table 2: Results of entire column generation algorithm using original shortest path and new feasible shortest path PSP formulations.

<table>
<thead>
<tr>
<th>Instance</th>
<th>Original Shortest Path PSP</th>
<th>New Feasible Shortest Path PSP</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Duality gap (%)</td>
<td>Evac. perc. (%)</td>
</tr>
<tr>
<td>HN80-I1.0</td>
<td>13.7</td>
<td>100.0</td>
</tr>
<tr>
<td>HN80-I1.1</td>
<td>13.6</td>
<td>100.0</td>
</tr>
<tr>
<td>HN80-I1.2</td>
<td>14.1</td>
<td>100.0</td>
</tr>
<tr>
<td>HN80-I1.4</td>
<td>15.1</td>
<td>100.0</td>
</tr>
<tr>
<td>HN80-I1.7</td>
<td>100.0</td>
<td>98.3</td>
</tr>
<tr>
<td>HN80-I2.0</td>
<td>100.0</td>
<td>89.9</td>
</tr>
<tr>
<td>HN80-I2.5</td>
<td>100.0</td>
<td>94.7</td>
</tr>
<tr>
<td>HN80-I3.0</td>
<td>69.1</td>
<td>77.3</td>
</tr>
</tbody>
</table>

one, whereas its linear relaxation did not produce any casualties. An interesting observation is that the new formulation manages to complete evacuation earlier in two instances, HN80-I1.0 and HN80-I1.2. Most importantly, the new PSP formulation produces evacuation paths that are completely cycle-free.

7 Conclusion

In this paper, we introduce a column generation algorithm that produces an evacuation plan with a cycle-free evacuation path and a non-preemptive evacuation schedule for each residential zone. The algorithm improves upon the column generation approach for joint mobility and evacuation planning proposed in [18] by addressing a critical shortcoming in its pricing subproblem which may result in undesirable evacuation paths containing cycles.

The original subproblem identifies variables with negative reduce cost by finding the shortest path in a time-expanded graph. Unfortunately, some of these shortest paths may visit a transit node in the time-expanded graph at different time steps, resulting in an evacuation path that contains cycles when the shortest path is collapsed onto a corresponding static graph. Our algorithm solves this problem by utilizing a different pricing subproblem which finds feasible shortest paths in the time-expanded graph. We define a feasible shortest path as a path with minimal cost that satisfies a feasibility constraint which allows the path to visit each transit node in the time-expanded graph at most once. The constraint directly addresses the root cause of the problem present in the original formulation by considering only shortest paths that would not contain cycles when they are collapsed onto the static graph when searching for variables with negative reduced cost.
We also propose a hybrid algorithm which combines a $K$-shortest-path-based algorithm with a MIP method to efficiently solve the new subproblem. Experiments applying algorithms utilizing the original and new subproblems on a real-world evacuation scenario show that the only drawback of the new algorithm is the larger CPU time consumption of its column generation phase. Otherwise, the new algorithm produces solutions that match the objective value and duality gap of the original while ensuring the evacuation paths it generates are cycle-free. We see potential in extending the algorithm to produce evacuation plans that utilize contraflow, as well as those that produce convergent paths similar to those introduced in [7] while still maintaining a non-preemptive schedule.

References