A multi-stage stochastic integer programming approach for locating electric vehicle charging stations

Ahmed Abdelmoumene Kadri, Romain Perrouault, Mouna Kchaou Boujelben, Céline Gicquel

A multi-stage stochastic integer programming approach for locating electric vehicle charging stations

Ahmed Abdelmoumene Kadri[1] , Romain Perrouault[2] , Mouna Kchaou Boujelben[1] , Céline Gicquel[2]

[1] United Arab Emirates University, Al Ain, United Arab Emirates

[2] Université Paris Saclay, Orsay, France

Abstract : Electric vehicles (EVs) represent one of the promising solutions to face environmental and energy concerns in transportation. Due to the limited range of EVs, deploying a charging infrastructure enabling EV drivers to carry out long distance trips is a key step to foster the widespread adoption of EVs. In this paper, we study the problem of locating EV fast charging stations so as to satisfy as much recharging demand as possible within the available investment budget. We focus on incorporating two important features into the optimization problem modeling: a multi-period decision making horizon and uncertainties on the recharging demand in terms of both the number of EVs to recharge and the set of long-distance trips to cover. Our objective is to determine the charging stations to be opened at each time period so as to maximize the expected value of the satisfied recharging demand over the entire planning horizon. To model the problem, we propose a multi-stage stochastic integer programming approach based on the use of a scenario tree to represent the uncertainties on the recharging demand. To solve the resulting large-size integer linear program, we develop two solution algorithms: an exact solution method based on a Benders decomposition and a heuristic approach based on a genetic algorithm. Our numerical results show that both methods perform well as compared to a stand-alone mathematical programming solver. Moreover, we provide the results of additional simulation experiments showing the practical benefit of the proposed multi-stage stochastic programming model as compared to a simpler multi-period deterministic model.

Keywords: Electric vehicles, Facility location problem, Multi-stage stochastic optimization, Scenario tree, Benders decomposition, Genetic algorithm.

1. Introduction

The increase of CO2 emissions has caused serious environmental issues that may be harmful to human life. According to the technical report published by the International Energy Agency [25], road-transport accounts for around a quarter of the global CO2 emissions related to energy. A promising way to reduce these emissions is to encourage the use of electric vehicles (EVs) instead of conventional gasoline vehicles. In a recent study of the International Energy Agency [26], it was estimated that the sale of EVs will sustain an annual growth rate of more than 25% by 2025.

Two different modes for using EVs are noticed: the first one corresponds to daily short-distance intra-city trips, for which the EV battery can be recharged at home and/or at the workplace. The full recharge of the EV battery requires between 5 and 8 hours using slow or medium charging equipment. The other mode, on which we focus in this paper, corresponds to long-distance inter-city trips. In this case, the distance traveled typically exceeds the range of an EV, i.e. the maximum distance that a fully charged vehicle can travel before its battery runs empty, so that multiple stops will be needed during the trip to recharge the battery. The charging stations have thus to be easily accessible when and where needed, and the charging time should be limited to a few minutes. Currently, the available fast charging technology allows a driver to recharge his battery within 20-30 minutes. However, it is far less widespread than the slow charging technology. For instance, in the USA, only 3% of the 12000 available public chargers are equipped with a fast charging technology [44]. This is mostly explained by the high installation and operating costs of fast charging stations (around 75000 € per station according to [46]).

Electric vehicles are in their earlier stages of development and the technology still needs to be improved to challenge current vehicles, especially as their range is limited between 200 and 450 km [24]. Egbue and Long [10] noticed that travel range, charging infrastructure, and cost are the main concerns for EV buyers. As a consequence, the use of EVs is currently limited to short-distance trips. The most efficient way to make EVs more attractive than conventional gasoline vehicles is to deploy an extensive charging infrastructure enabling drivers to quickly recharge their battery during long-distance trips. Due to the large capital expenditures involved in the deployment of a fast charging infrastructure, it is essential to carefully select the locations of the charging stations in order to maximize demand satisfaction and limit the investment costs.

As will be shown by the literature review provided in Section 2, most published works on the resulting facility location problem have focused on a single-period deterministic setting. Yet, deploying an EV charging infrastructure is not a one-shot decision but rather a step-by-step roll-out process in which location decisions are made dynamically according to the evolution of the demand and the availability of the investment budget. Moreover, as this deployment plan is likely to span several years, there will be significant uncertainties on the problem input data. Not taking these uncertainties into account means neglecting a critical part of the problem and may lead to suboptimal location decisions. Hence, a multi-period strategic station deployment plan taking into account the time dependence and the stochasticity of the problem input data might be very useful to a decision maker.

In this context, we study the problem of locating EV fast charging stations on a road network. We consider a multi-year planning horizon and take into account uncertainties on the recharging demand in terms of both the number of EVs to recharge and the set of long-distance trips to cover. We seek to determine the charging stations to be opened at the beginning of each year so as to maximize the expected value of the recharging demand satisfied. Our contributions are threefold:

1. We propose a multi-stage stochastic integer programming (MSSIP) approach for this stochastic optimization problem based on the use of a scenario tree to represent the uncertainties on the demand. Similar MSSIP approaches have already been investigated for a variety of stochastic multi-period facility location problems related to the design of supply chain networks: see e.g. [12], [37],[41], [49] and [53]. However,

all these works consider that the demand to be satisfied is located at fixed nodes of the network. Thus, to the best of our knowledge, this is the first time a MSSIP approach is developed for a location problem such as the EV fast-charging station location problem in which the demand is modeled as a set of flows going through the network. Moreover, our simulation study shows the practical interest of the proposed stochastic model as compared to a simpler multi-period deterministic model.

2. In order to solve the resulting large-size integer linear program, we develop an exact method based on a Benders decomposition of the problem. This is achieved by extending the approach proposed by Arslan and Karasan [2] for the single-period deterministic problem to our multi-period stochastic problem. Similarly to Arslan and Karasan [2], our numerical results show that the Benders decomposition method significantly outperforms a stand-alone mathematical programming solver. However, contrary to Arslan and Karasan [2], we found that on average, a multi-cut implementation of the method adding a large number of disaggregated cuts at each iteration of the algorithm gives better results than a multi-cut implementation adding a small number of aggregated cuts at each iteration.
3. In view of the numerical difficulty of solving instances of the problem for long planning horizons, we also propose a heuristic approach based on a genetic algorithm. The novelty of our genetic algorithm pertains to three key steps of the general algorithm. We namely propose new strategies to select the individuals in the initial population and a new computationally efficient algorithm to evaluate the fitness function of each individual. Moreover, we propose new crossover and mutation operators ensuring that the obtained solutions are feasible. Our numerical results show that, for the instances involving the longest planning horizon, the genetic algorithm provides good quality solutions in a computation time significantly smaller than the one needed by a Benders decomposition approach.

This paper is organized as follows. Section 2 provides an overview of the relevant literature. Section 3 describes the facility location problem under study, introduces the proposed multi-stage stochastic programming approach and present the integer linear programming (ILP) formulation modeling the problem. In Section 4, we develop an exact solution method based on a Benders decomposition algorithm. We then propose in Section 5 a genetic algorithm in order to solve the large size instances for which the exact method fails to find optimal solutions. In Section 6, we carry out various numerical experiments in order to assess the performance of the proposed solution approaches and to evaluate the benefit from using a stochastic model instead of a deterministic one. Finally, we provide in Section 7 a conclusion and some research perspectives.

2. Related literature

We provide in this section an overview of the related literature. We first consider the modeling of the EV charging station location problem, focusing in particular on multi-period and stochastic extensions. We then discuss the various solution approaches which have been investigated for the Flow Refueling Location Problem before providing a brief state of the art on multi-stage stochastic integer programming.

2.1. *Modeling the EV station location problem*

In traditional facility location problems, the objective is to determine the best locations for facilities in a way to satisfy customer demand at fixed nodes of a network (see [39] for an overview of the facility location literature). However, node-based models implicitly assume that drivers carry out a special round trip to a station to recharge their battery and are thus not always suitable for modeling the EV charging stations location problem. In particular, drivers traveling for long-distance trips recharge their battery while on their way to another destination. In this case, demand should be modeled as a set of origin-destination flows representing the various trips EV drivers are wishing to carry out. Hodgson [18] introduced the flow capturing location model (FCLM), where an origin-destination flow is assumed to be ”captured” if at least one facility is located anywhere on the corresponding path. The main issue when applying the flow capturing model for locating EV charging stations is that it does not take into account the limited range of EVs. Kuby and Lim [29] extended the flow capturing location model to incorporate the limited range constraint and defined the flow refueling location problem (FRLP). In their model, an origin-destination flow is considered as refueled if and only if a set of charging stations are built at carefully chosen locations on the shortest path between the origin and the destination so that an EV driver is able to drive from the origin to the destination and back without running out of fuel. Many extensions of the FRLP have been proposed since this seminal work. Limited charging station capacities were considered e.g. in [45] and [19]. The possibility of using paths slightly deviating from the shortest origin-destination paths was studied by Kim and Kuby [27], [28], Yildiz et al. [52] and Li et al [31]. Two types of charging stations (slow and fast charging stations) were studied by Wang and Lin [47] whereas Arslan and Karasan [2] considered two types (hybrid and pure electric) of vehicles. Flow-dependent charging delays induced by congestion at charging stations were introduced by Ghamami et al. [15].

Multi-period deterministic extensions of the flow refueling location problem were considered by three recent works. Chung and Kwon [7] proposed a multi-period model in which new stations can be opened in a dynamic way but stations built in a given period cannot be relocated until the end of the planning horizon. Their numerical experiments showed the benefit of using a multi-period model as compared to the use of a myopic approach based on the resolution of a series of single-period models. Li et al. [31] studied the multi-period multi-path flow refueling location problem. Their model assumes that drivers are willing to deviate from shortest paths and includes the possibility of relocating stations from period to period. In [55], Zhang et al. investigated the multi-period flow refueling location problem. They considered charging stations with a limited capacity and did not allow relocation of an opened station. They focused on modeling the influence of station siting decisions on the demand growth pattern to model the fact that the adoption of EVs by drivers depends on the charging opportunities.

However, it seems that until now, only a limited attention has been given in the literature to the stochastic nature of the EV charging infrastructure planning problem whereas in practice, there are significant uncertainties on the input parameters of the optimization problem to be solved. De Vries and Duijzer [8] and Lee and Han [30] addressed the uncertainty of the vehicle range. Under a stochastic driving range, the coverage of an origin-destination flow becomes a matter of chance rather than a binary observation. Hence, the problem modeling involves computing the coverage probability of each flow as the joint probability that each portion of the corresponding path comprised between two opened stations will be shorter than the driving range. Other works considered the uncertainty of the recharging demand. Hosseini and MirHassani [20] studied the problem under uncertain traffic flows. They proposed a two-stage stochastic programming model in which portable charging stations can be opened in the second decision stage in order to improve the expected flow coverage. Wu and Sioshansi [48] studied a stochastic flow capturing location problem under recharging demand uncertainty and proposed a two-stage stochastic programming approach. Miralinaghi et al. [35] addressed the uncertainty of the refueling demand while considering a multi-period horizon. They proposed a robust optimization approach in which infrastructure deployment solutions are assessed based on the worst-case demand scenario.

2.2. *Solving the flow refueling location problem*

The flow refueling location problem is a difficult combinatorial problem which can be formulated as a mixed-integer linear/non-linear program (MILP/MINLP) depending on the modeling assumptions. Kuby and Lim [29] proposed a first MILP formulation of the problem relying on an explicit enumeration of all the combinations of stations that can refuel each path. However, since the pre-processing stage used to generate all these combinations is computationally burdensome, the formulation becomes impractical for large-size instances. More efficient MILP formulations relying on an implicit description of these combinations were proposed in [4] and [36]. Capar et al. [4] developed the arc-cover path-cover model, which relies on the idea that a flow is refueled if each arc of the corresponding path is refueled by an opened station. The cover set defining the list of stations that can refuel a given arc can easily be built in a pre-optimization step. MirHassani and Ebrazi [36] proposed a formulation based on the concept of path segment. Basically, a path segment corresponds to a sequence of nodes and arcs such that a vehicle refueled at the first node of the sequence can travel to the last node of the sequence without running out of charge. A path is covered if the round trip between the origin and the destination can be carried out by using a succession of path segments.

Even with a strong and compact MILP formulation, solving large-size instances of the flow refueling location problem can be challenging. Solution approaches based on mathematical programming techniques were recently proposed to tackle this issue. Arslan and Karasan [2] investigated the use of a Benders decomposition method to solve an extension of the arc-cover path-cover model. Their decomposition scheme comprises a master problem in which the stations siting decisions are made and a set of sub-problems, each one focusing on the coverage of a single origin-destination flow. They assessed the performance of their algorithm using single-cut and multi-cut implementations of the Benders decomposition and showed that their algorithm performs much faster than CPLEX solver. Note that a similar decomposition is used by Wu and Sioshansi [48] to solve a stochastic variant of the flow capturing location problem. Miralinaghi et al. [35] also developed a cutting plane (CP) generation algorithm to solve their robust optimization model. Yildiz et al. [52] considered an extension of the path segment formulation for the case where deviations from the shortest path are allowed on each flow. They proposed a branch-and-price (B&P) algorithm in which path segment decision variables are added iteratively to the problem formulation using a column generation strategy. Lee and Han [30] formulated the flow refueling location problem under a stochastic driving range as a MINLP and proposed a Benders-and-Price algorithm (BD&P) combining Benders decomposition and column generation techniques to solve their problem. Hosseini and MirHassani [19] investigated the problem with capacitated refueling stations and developed a heuristic solution method based on the Lagrangian relaxation (LGR) of the capacity constraints.

Heuristic solution approaches were also investigated for several variants of the problem in order to overcome the computational difficulties encountered with real-life instances. Kim and Kuby [28] developed a greedy heuristic (GH) based on an artificial feasible network to solve the problem with deviations. Hosseini and MirHassani [20] proposed a two-step heuristic for the problem with stochastic demand: the first step exploits the linear relaxation solution to select a subset of potentially interesting nodes for locating stations, the second step uses a modified Lagrangian (ML) iterative method to solve the restricted location problem. In order to solve the multi-period variant of the problem, Chung and Kwon [7] and Zhang et al. [55] developed heuristic procedures based on forward or backward myopic methods (Myopic) in which a sequence of single-period sub-problems is solved. Finally, several works proposed solution approaches based on meta-heuristics. A simulated annealing (SA) algorithm was developed by Ghamami et al. [15] to solve their general corridor model formulated as a MINLP. Lim and Kuby [32] developed a genetic algorithm (GA) to solve the single-period deterministic problem introduced by Kuby and Lim [29]: to make sure that all solutions comply with the constraint on the maximum number p of stations that can be opened, they encoded a chromosome as a set of p integers providing the ID number of the opened facilities. Li et al. [31] also used a genetic algorithm to solve the multi-period location problem formulated as a mixed integer linear program. They used a binary matrix to represent how the charging stations are deployed on a spatiotemporal network and incorporated feasibility checks after the crossover and mutation processes to identify and repair infeasible solutions. To take into account driver behavior with respect to travel and refueling time, Miralinaghi et al. [35] investigated a bi-level programming formulation and solved the resulting MINLP with a genetic algorithm. Table 1 summarizes the classification of the above-mentioned works.

Article	Problem variant				Formulation	Solving methods	
	SP	MP	Det.	Sto.		Exact	Heuristic
[2]	✓		✓		MILP	Benders	
[4]	✓		✓		MILP	Solver	
[7]		✓	✓		MILP	Solver	Myopic
[8]	✓			R	MILP	Solver	
[15]	✓		✓		MINLP		SA
[19]	✓		✓		MILP		LGR
[20]	✓			D	MILP		ML
[27]	✓		✓		MILP	Solver	
[28]	✓		✓		MILP		GH
[29]	✓		✓		MILP	Solver	
[30]	✓			R	MINLP	BD&P	
[31]		✓	✓		MILP		GA
[32]	✓		✓		MILP		GH, GA
[35]		✓		D	MINLP	CP	GA
[36]	✓		✓		MILP	Solver	
[45]	✓		✓		MILP	Solver	
[47]	✓		✓		MILP	Solver	
[48]	✓			D	MILP	Benders	
[52]	✓		✓		MILP	B&P	
[55]		✓	✓		MINLP		Myopic
Our work		✓		D, P	ILP	Benders	GA

SP: Single-period; MP: Multi-period

Det.: Deterministic environment; Sto.: Stochastic environment

R: stochastic EV Range; D: stochastic recharging demand on each path; P: stochastic set of paths

Table 1: Classification of the related works

2.3. *Multi-stage stochastic integer programming*

Multi-stage stochastic programming is a widely used approach for dynamic decision making under uncertainty. A multi-stage stochastic optimization problem involves a sequential decision making process comprising a set of decision stages and a stochastic process ξ_1, ..., ξ_T representing the future evolution of the random input parameters. At each stage t, the decision maker observes the value of some random parameters ξ_t and makes a decision which may depend on the past observed values of ξ_0,....ξ_t so as to optimize an objective function representing some long-term goals. As explained e.g. by Pflug [40], when the random parameters ξ_0,....ξ_T are continuous random variables, the problem becomes a functional optimization problem, i.e. an infinite dimension optimization problem in which at least one unknown is a function. Such a problem is not solvable in practice except for a few extremely simple and unrealistic cases. Thus, in order to represent uncertainty in a form suitable for computation, an approximate representation of the random parameters based on a discretization of their continuous probability distribution is used. This amounts to building a discrete scenario tree to represent the evolution of the random parameters over time. The scenario tree being given, the stochastic optimization problem becomes a deterministic optimization problem (sometimes called the deterministic equivalent problem) in which deterministic decisions have to be made for each node of the scenario tree. These decisions represent the actions to be taken in case the stochastic process evolves up to the state of the system described by the corresponding node.

In practice, several methods are available to design scenario trees in a multi-stage stochastic programming approach. A first simple method consists in using trees in which only a limited number of possible outcomes is taken into account at each stage. Xie and Huang [49] thus use binary scenario trees with only two possible realizations for the demand (low or high) to be satisfied by their supply chain network. Similarly, Pimentel et al. [41] use ternary scenario trees in which each node has exactly three children (one for the expected value, one for a lower-than-expected value, one for a larger-than-expected value) to model the stochastic demand in their facility location problem. Another possibility is to use a sampling method to discretize a continuous probability distribution: see e.g. [12] where scenario trees are generated using a Latin Hypercube sampling. Other constructive methods not based on sampling are proposed among others in [40] or [21]. Finally, we note that the size of the scenario tree grows exponentially fast with the number of stages and with the number of branches at each stage. In this case, scenario tree reduction techniques such as the ones proposed by [9] and [17] can be used to generate scenario trees of acceptable size while keeping the approximate representation of the stochastic process as accurate as possible.

A multi-stage stochastic integer programming approach often results in the formulation of a huge mixed-integer linear program (MILP) involving a number of variables and constraints which is basically proportional to the one of the scenario tree. Solving such an MILP is a challenging task and several exact and heuristic solution algorithms have been proposed in the literature to tackle the resulting computational difficulties.

Exact solution algorithms relying on MILP formulation strengthening techniques are presented e.g. in [16] and [1]. These algorithms aim at enhancing the efficiency of the Branch & Cut algorithm embedded in MILP solvers by improving the quality of the bounds used at each node of the search tree through valid inequalities or extended formulations. However, they do not scale up very well with the size of the scenario tree. Hence, several alternative decomposition methods have been investigated. A first type of decomposition approach is based on an extensive formulation of the problem in which a set of single-scenario sub-problems are linked together by non-anticipativity constraints ensuring that decisions made at stage h are based only on the information available up to this stage. By relaxing and dualizing these non-anticipativity constraints, the problem can be decomposed into a set of single-scenario independent sub-problems which can be more easily solved. This idea is exploited among others in the progressive hedging algorithm (see e.g. [33], [14] and [22]) and in Lagrangian based heuristics (see e.g. [5] and [38]). A recent work by Escudero et al. [11] propose to relax only a subset of the non-anticipativity constraints, resulting in sub-problems involving small sub-trees of the scenario tree. A second type of decomposition approach starts from a dynamic programming formulation involving nested expected cost-to-go functions. Here, the problem is decomposed into a series of single-period deterministic sub-problems, one for each node of the scenario tree, linked together by dynamic programming equations. Zou *et al.* [57] exploit this idea to devise a sampling-based nested decomposition algorithm called the Stochastic Dual Dynamic integer Programming (SDDiP) algorithm. Note that this algorithm relies on the assumption that the scenario tree displays the stage-wise independency property, i.e. is such that any two nodes of the scenario tree belonging to the same stage have children nodes defined by identical data and conditional probabilities. Finally, Benders decomposition algorithms exploiting the specific structure of the stochastic facility location and capacity sizing problem under study are discussed e.g. in [12] and [42]. Moreover, in view of the numerical difficulties encountered when solving large-size instances, solution algorithms based on metaheuritics, and in particular on genetic algorithms, have also been proposed to tackle multi-stage stochastic optimization problems arising in various application domains such as financial portfolio optimization (Chan et al. [3], Yang [51], Zhang et al. [54]), industrial maintenance planning (Yahyatabar and Najafi [50]) or water resource management ([56]).

2.4. *Position of this work in the literature*

As shown in Table 1, most existing works on the flow refueling location problem consider either deterministic multi-period models or stochastic single-period models. A noticeable exception can be found in Miralinaghi et al. [35] who investigate a multi-period stochastic variant of the FRLP. However, in their model, all station locations have to be made 'once and for all' at the beginning of the planning horizon and cannot be dynamically adjusted over time to the actual realization of the demand. In contrast, the present work considers a multi-stage stochastic decision process spanning several years in which station location decisions are made year after year as more information on the charging demand becomes known. This seems relevant as in practice, the available investment budget is unlikely to be sufficient to allow to deploy in one step all the stations needed to cover the need of all potential EV drivers: a multi-year phasing construction plan taking into account the available budget and the demand year after year will thus be useful for managers. To the best of our knowledge, this is the first time such a simultaneously stochastic and dynamic extension of the flow refueling location problem is investigated.

We propose to model this stochastic combinatorial optimization problem through a multi-stage stochastic integer programming approach in which scenario trees are used to approximately represent the evolution of the underlying stochastic process over time. To solve the resulting large size mixed-integer linear program to optimality, we extend the Benders decomposition approach proposed by Arslan and Karasan [2] for the single-period deterministic problem to our multi-period stochastic problem. This Benders decomposition algorithm exploits some specific features of the FRLP. To this respect, our work belongs to the same line of research as the ones found in [12] and [42]. However, these two works deal with applications arising in supply chain network design and thus use a node-based representation of the demand. Moreover, we would like to point out that the SDDiP algorithm proposed in [57] relies on the assumption that the scenario tree is stage-wise independent. In our problem, the flow on each trip at a given node of the scenario tree depends on the flow on the same trip at the parent node in the tree. Our scenario trees are thus not stage-wise independent so that applying a decomposition approach such as the SDDiP algorithm to solve our problem is not straightforward. Finally, similarly to e.g. [3], [51], [54]), [50] and [56], we propose a genetic algorithm (GA) to find good quality solutions when optimal solutions are difficult to obtain. Note that our GA is based on a fine analysis of the specific features of our problem and seeks to exploit them as best as possible at several critical steps of the algorithm, namely the building of an initial population, the evaluation of the fitness function of each individual and the definition of the crossover and mutation operators. The proposed GA shares some common features with the one developed by Li et al. [31] for the multi-period deterministic FRLP. In particular, the encoding of each individual through a binary vector and the definition of the mutation operator are similar. However, Li et al. [31] formulate their location problem using a set-covering model in which all the charging demand has to be satisfied and relocation of a station from one period to the next is allowed whereas we use a max-covering type model in which we aim at satisfying as much demand as possible with the limited number of stations available and relocation of a station is forbidden. As a consequence, their definition of the crossover operators and the feasibility checks carried out on each individual significantly differ from ours.

3. Mathematical formulation

3.1. Flow refueling location problem (FRLP)

We represent the road network by a directed graph $G = (\mathcal{V}, \mathcal{A})$ where $\mathcal{V}$ denotes the set of vertices and $\mathcal{A}$ denotes the set of arcs. The demand is modeled as a set $\mathcal{Q}$ of predetermined trips to be taken by EV drivers. Each trip q corresponds to traveling from an origin $O^q \in \mathcal{V}$ to a destination $D^q \in \mathcal{V}$ and back. All drivers taking trip q are assumed to follow the shortest path between O^q and D^q and to be unwilling to deviate from it to recharge their battery. We denote $\mathcal{V}^q$ (resp. $\mathcal{A}^q$) the set of vertices (resp. arcs) belonging to the shortest path between O^q and D^q. Due to the limited driving range of EVs, drivers traveling for long distances need to recharge their battery multiple times while on their way to the destination. Accordingly, path q is considered 'covered' or 'refueled' only if the stations opened on the path allow drivers to carry out a round trip between O^q and D^q without running out of charge, otherwise, the path is considered not 'covered'. The objective of the flow refueling location problem (FRLP) is thus to select the best locations for charging stations on the network in a way to maximize the EV flow covered, within a limited investment budget. In our problem modeling, we make the following assumptions which are commonly used in the FRLP-related literature: there is a single type of EV with a limited range R (expressed in miles or kilometers), energy consumption is proportional to the traveling distance and is symmetrical in both directions of each arc, charging stations are uncapacitated and accessible for traffic from both directions and only the vertices of the network are considered as possible locations for the charging stations.

We use the arc-cover path-cover based approach introduced by Capar et al. [4] to model the problem. In this approach, a path q is considered covered if EV drivers are able to travel along each arc of $\mathcal{A}^q$ without running out of charge. In other words, for each arc $(j, k) \in \mathcal{A}^q$, there should be an opened station at at least one vertex i located before k on the path such that a full recharge of the vehicle at node i allows EV drivers to cross arc (j, k) without running out of charge. This means that the distance between i and k should not exceed the range R. In case the arc is linked to the origin or to the destination of the path, assumptions have to be made on the battery initial charging level in order to enable EV drivers to carry round trips on the path. More precisely, if a charging station is located at the origin (respectively destination) of the path, we assume that the battery is fully charged when leaving the origin (respectively destination). However, if no charging station is located at the origin (respectively destination), we assume that the battery should be at least half charged when leaving the origin (respectively destination). This reasoning relies on the idea that if a vehicle could reach the first station after the origin (respectively destination) starting with a half charged battery, it could recharge the battery at that same station on his return trip (respectively outward trip) and return to the origin (respectively arrives to the destination) with a half charged battery. This allows EV drivers on path q to cyclically travel from O^q to D^q to O^q, etc... alternatively crossing each arc belonging to the path in the outward and return direction.

In their arc-cover path-cover model, Capar et al. [4] introduced the arc covering set $\mathcal{K}^{q;(j,l)}$ as the set of candidate vertices situated before vertex l on path q, and within a distance from vertex l shorter than the range R. Hence, if a station is built on at least one of these vertices, the arc (j, l) on path q can be refueled. Figure 1 illustrates a small network with 6 vertices, where the distance between vertices is noted on arcs. Let us consider the path q that starts at vertex 4 then visits vertices 5, 1 and 2 before ending at vertex 3. Assuming that the driving range is 90, $\mathcal{K}^{q;(1,2)} = \{1; 5\}$, i.e. arc $(1, 2)$ can be refueled with stations at either vertex 1 or 5. Vertices 1 and 5 belong to the set because they are located before vertex 2 on the path and within a distance shorter than 90. Similarly, $\mathcal{K}^{q;(5,1)} = \{4; 5\}$, $\mathcal{K}^{q;(4,5)} = \{4; 5\}$ and $\mathcal{K}^{q;(2,3)} = \{1; 2; 5\}$. The arc covering sets have to be defined for arcs in both directions in order to ensure round trips on the path.

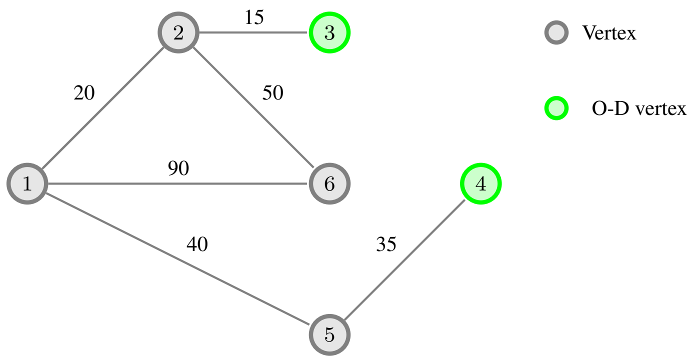

Figure 1: Arc covering example ($R = 90$)

3.2. Uncertainty modeling

In practice, the development of an EV fast charging infrastructure is likely to be a long-term process requiring several years, mostly because the available investment budget will be insufficient to allow to deploy in one step all the stations needed to cover the demand of all potential EV drivers. In order to improve the practical relevancy of the FRLP, we thus consider a multi-period extension in which the dynamic evolution of both the set of trips taken by EV drivers within the road network and the amount of EVs flow on each trip is taken into account. Moreover, since forecasting the exact value of these parameters over a long planning horizon is a very challenging task, their evolution will be subject to uncertainties. This leads to the formulation of a multi-period stochastic flow refueling location problem. To handle this problem, we propose to use a multi-stage stochastic integer programming approach. Our problem modeling relies on the assumption that not all station opening decisions have to be made at the beginning of the planning horizon but rather that some of them can be postponed to a future point in time when more information on the charging demand becomes known. In other words, we assume that deploying an EV charging infrastructure is a multi-stage decision process.

In the context of multi-stage stochastic programming, a stage in the decision process can be basically defined as a future point in time at which new information on the stochastic parameters becomes available and decisions based on this newly available information have to be made. In our case, we consider a planning horizon spanning several years. This horizon is divided into a set of H discrete periods, in which each period t corresponds to one year. At the beginning of year t, we have to make decisions on the charging stations to be installed during the forthcoming year. These decisions will be based on the realization of the stochastic charging demand up to year t. In other words, a stage in our model corresponds to one year in a multi-year planning horizon and is a point in time at which new stations will be deployed. Note that we assume that, at the beginning of year t where decisions have to made on the stations to be located during year t, the information of the future charging demand to be satisfied during period t is sufficiently accurate to be considered deterministic. In this sense, our model can be understood as a 'hazard-decision' type model (see e.g. [6] for an introduction on this concept) in which decisions at stage t are made after the realization of the random parameters at this stage. The development of a 'decision-hazard' type model in which decisions for year t have to be made before the actual value of the charging demand for year t is known would require additional research developments and is thus left for future work.

In the proposed approach, the underlying stochastic input process is approximately represented by a discrete scenario tree $\mathcal{T}$ containing T nodes. Each node $n \in \mathcal{T}$ corresponds to a single planning period t_n, has a given probability of occurrence π_n and represents the state of the system that can be distinguished by the information unfolded up to period t_n. At any non-terminal node of the tree, there are k branches to indicate future possible outcomes of the random variables from the current node. An example of scenario tree is given in Figure 2. The first time period of the planning horizon corresponds to the root node of the tree, which has k children nodes in the second time period, corresponding to k possible evolutions of the EV market. Each of these children nodes is parent of k nodes in the third time period, etc... until reaching the tree leaves at the last time period H of the planning horizon. A scenario is defined as a path in the tree from the root node to a leaf node and represents a possible outcome of the stochastic input parameters over the whole planning horizon. Note that the fact that the station location relative to year t can be postponed up to beginning of this year is translated into the problem formulation by the fact that the station location decisions x_n^i are node-dependent rather than period-dependent. However, we do not allow relocation of an opened station, i.e. a charging station opened at a node $n \in \mathcal{T}$ must remain opened at all nodes successors of n in the tree.

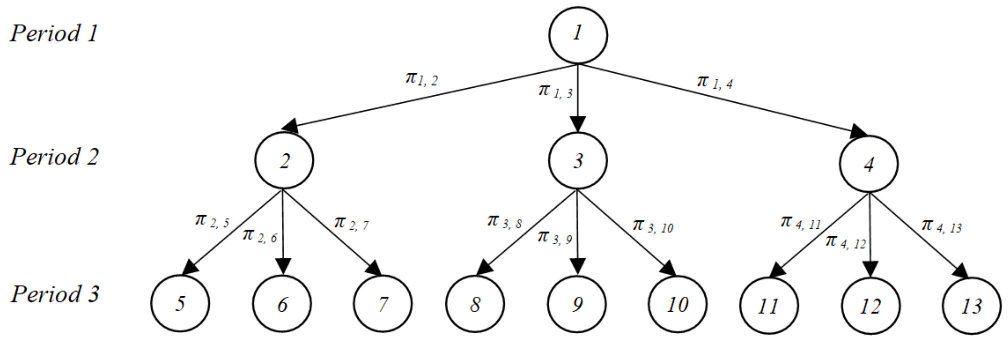

Figure 2: Example of a scenario tree with 13 nodes

3.3. Mathematical formulation

We propose an ILP formulation for the multi-period stochastic problem of locating EV charging stations. Our formulation relies on a scenario tree to describe the evolution of the uncertain parameters of the problem over time. The objective is to maximize the expected EV flow covered on the paths of the road network, computed as the sum of EV flows covered at each node of the tree, weighted by the probabilities of occurrence of the nodes. The notations used in the model are summarized in Table 2. The formulation is given by objective function (1) and constraints (2)-(6).

Parameters	**Description**
$\mathcal{V}$	Set of network vertices, containing V vertices
$\mathcal{A}$	Set of oriented arcs of the network, containing A arcs
$\mathcal{M}_n$	Set of origin/destination vertices at node n of the scenario tree, containing M_n vertices ($\mathcal{M}_n \subseteq \mathcal{V}$)
$\mathcal{M}$	Set of all origin/destination vertices ($\mathcal{M} = \bigcup\limits_n \mathcal{M}_n$)
$\mathcal{T}$	Set of nodes in the scenario tree, containing T nodes ($T = \frac{k^H - 1}{k-1}$)
k	Number of children per parent node in the scenario tree
H	Number of periods in the planning horizon
$\mathcal{Q}_n$	Set of origin/destination paths ($q \in \mathcal{Q}_n$) at node n of the scenario tree, where the total number of paths is: $Q_n = \frac{M_n(M_n - 1)}{2}$
$\mathcal{Q}$	Set of all origin/destination paths ($\mathcal{Q} = \bigcup\limits_n \mathcal{Q}_n$)
$\mathcal{A}^q$	Set of oriented arcs belonging to path q ($\mathcal{A}^q \subset \mathcal{A}$)
$\mathcal{V}^q$	Set of vertices belonging to path q
$\mathcal{K}^{q,\alpha}$	Set of vertices that can refuel arc $\alpha \in \mathcal{A}^q$

t_n	Time period of tree node n ($n \in \mathcal{T}$), with $t_1 = 1$
p_t	Number of stations to be opened in period t
a_n	Parent of node n in the scenario tree
$succ_n$	Set of all successors of node n in the scenario tree
π_n	Probability of occurrence of tree node n
$\pi_{n,n'}$	Probability of transition from node n to its child node n'
R	EV driving range
f_n^q	Traffic flow on path q at tree node n
Variables	**Description**
x_n^i	Equal to 1 if a station is built at vertex i at tree node n, and 0 otherwise
y_n^q	Equal to 1 if the flow on path q is covered at tree node n, and 0 otherwise

Table 2: Model notations

$$\text{maximize} \quad \sum_{n \in \mathcal{T}} \pi_n \sum_{q \in \mathcal{Q}_n} f_n^q y_n^q \tag{1}$$

subject to

$$\sum_{i \in \mathcal{K}^{q,\alpha}} x_n^i \geq y_n^q \qquad \forall n \in \mathcal{T},\ \forall q \in \mathcal{Q}_n,\ \forall \alpha \in \mathcal{A}^q \tag{2}$$

$$\sum_{i \in \mathcal{V}} x_n^i = p_{t_n} \qquad \forall n \in \mathcal{T} \tag{3}$$

$$x_n^i \geq x_{a_n}^i \qquad \forall i \in \mathcal{V},\ n \in \mathcal{T} \tag{4}$$

$$x_n^i \in \{0, 1\} \qquad \forall i \in \mathcal{V},\ n \in \mathcal{T} \tag{5}$$

$$y_n^q \in \{0, 1\} \qquad \forall q \in \mathcal{Q},\ n \in \mathcal{T} \tag{6}$$

The objective function (1) maximizes the total expected EV flow covered. Constraints (2) state that a path q is covered at node $n \in \mathcal{T}$ if all arcs belonging to path q are covered. An arc α is covered if there is at least one station open among the ones capable of covering it, i.e. among the stations belonging to the set $\mathcal{K}^{q,\alpha}$. Constraints (3) limit the number of stations to be opened at each node $n \in \mathcal{T}$ to the maximum number of stations allowed in the corresponding time period. Constraints (4) ensure that a station opened at a node $n \in \mathcal{T}$ remains open at all nodes successors of n in the scenario tree. Constraints (5) and (6) are the binary decision variable restrictions.

Note that problem (1)-(6) is NP-hard as it is a variant of the maximum covering problem. Solving large instances of the problem with a commercial solver thus requires a long computation time and a large memory space, in particular due to the huge number of constraints (2) in the problem formulation. We thus propose in the next section a decomposition approach capable of providing optimal solutions in reduced computation times for medium-size instances.

4. Benders decomposition

In this section, we discuss an exact method aiming at solving problem (1)-(6) to optimality. More precisely, we extend the Benders decomposition approach proposed by Arslan and Karasan [2] for the single-period deterministic case to the multi-period stochastic case.

Benders decomposition is especially appropriate to solve large-size mixed-integer linear programs displaying a particular structure in which a set of independent linear sub-problems are linked together by a set of (continuous or integer) coupling variables. Problem (1)-(6) displays such a structure. Namely, there is a set of independent sub-problems, each one related to the coverage of a single path at a single node of the scenario tree. Note that, even if the coverage variables y_n^q are initially introduced as binary variables, in formulation (1)-(6), they can be relaxed as continuous variables defined over the interval [0,1] without changing the optimal solution. These single-node single-path sub-problems are linked together by the binary location variables x_n^i. We thus develop a Benders decomposition scheme in which the master problem contains all binary variables and each sub-problem contains a single continuous variable y_n^q related to the coverage of path q at node n. In other words, the master problem will decide upon the station deployment strategy over the whole scenario tree while each sub-problem will focus on evaluating the resulting demand coverage for each path at each tree node.

In what follows, we first present the primal and dual formulations of the sub-problems, together with their optimal solutions which can be obtained by inspection. We then investigate two alternative formulations for the master problem, each one leading to a multi-cut implementation of the Benders decomposition algorithm. We then discuss several cut selection strategies aiming at improving the overall performance of the algorithm.

4.1. Benders sub-problems

Given a station deployment plan $\widehat{x} \in \{0,1\}^{|VT|}$, the coverage of each path $q \in \mathcal{Q}_n$ at each node $n \in \mathcal{T}$ is obtained by solving the following sub-problem denoted $SP_n^q(y|\widehat{x})$.

$$\text{maximize} \quad y_n^q \tag{7}$$

$$\text{subject to}$$

$$y_n^q \leq \sum_{i \in \mathcal{K}^{q,\alpha}} \widehat{x}_n^i \qquad \forall \alpha \in \mathcal{A}^q \tag{8}$$

$$y_n^q \leq 1 \tag{9}$$

$$y_n^q \geq 0 \tag{10}$$

Problem (7)-(10) is a linear program involving a single continuous variable. It can be solved by inspection by setting:

$$\widehat{y}_n^q = min(1; \sum_{i \in \mathcal{K}^{q,\alpha}} \widehat{x}_n^i, \alpha \in \mathcal{A}^q) \tag{11}$$

Let $\mu_n^{q,\alpha}$ be the dual variables related to constraints (8) and ν_n^q the dual variable related to constraint (9). The dual of sub-problem $SP_n^q(y|\widehat{x})$, denoted $DSP_n^q(\mu,\nu|\widehat{x})$, can be formulated as follows.

$$\text{minimize} \quad \sum_{\alpha \in \mathcal{A}^q} \left[\left(\sum_{i \in \mathcal{K}^{q,\alpha}} \widehat{x}_n^i \right) \mu_n^{q,\alpha} \right] + \nu_n^q \tag{12}$$

subject to

$$\sum_{\alpha \in \mathcal{A}^q} \mu_n^{q,\alpha} + \nu_n^q \geq 1 \tag{13}$$

$$\mu_n^{q,\alpha} \geq 0 \qquad \forall \alpha \in \mathcal{A}^q \tag{14}$$

$$\nu_n^q \geq 0 \tag{15}$$

As noted by Arslan and Karasan [2], $DSP_n^q(\mu,\nu|\widehat{x})$ is a continuous knapsack problem and can thus also be solved by inspection.

Let $\mathcal{A}_n^{q,0}(\widehat{x})$ (resp. $\mathcal{A}_n^{q,1}(\widehat{x})$) be the set of arcs in $\mathcal{A}^q$ that are not covered (resp. that are covered by a single station) at node n of the scenario tree in the station deployment plan $\widehat{x}$. The set of optimal solutions of $DSP_n^q(\mu,\nu|\widehat{x})$ can be characterized as follows:

- If $\widehat{y}_n^q = 0$

  For each $\alpha_0 \in \mathcal{A}_n^{q,0}(\widehat{x})$, the following solution is optimal:

  $$\widehat{\nu}_n^q = 0 \;\; ; \;\; \widehat{\mu}_n^{q,\alpha_0} = 1 \;\; ; \;\; \widehat{\mu}_n^{q,\alpha} = 0, \forall \alpha \in \mathcal{A}^q \setminus \{\alpha_0\}$$

- If $\widehat{y}_n^q = 1$

  - For each $\alpha_1 \in \mathcal{A}_n^{q,1}(\widehat{x})$, the following solution is optimal:

  $$\widehat{\nu}_n^q = 0 \;\; ; \;\; \widehat{\mu}_n^{q,\alpha_1} = 1 \;\; ; \;\; \widehat{\mu}_n^{q,\alpha} = 0, \forall \alpha \in \mathcal{A}^q \setminus \{\alpha_1\}$$

  - The following solution is also optimal:

  $$\widehat{\nu}_n^q = 1 \;\; ; \;\; \widehat{\mu}_n^{q,\alpha} = 0, \forall \alpha \in \mathcal{A}^q$$

Note that all these solutions correspond to extreme points of the feasible space defined by constraints (13)-(15) and that any convex combinations of these points also provide an optimal solution of $DSP_n^q(\mu, \nu|\widehat{x})$.

4.2. Benders master problem

We now investigate two alternative formulations for the Benders master problem.

Let $\mathcal{D}_n^q$ be the set of extreme points of the polyhedron $\mathcal{P}_n^q \subset \mathbb{R}^{|\mathcal{A}^q|+1}$ defined by constraints (13)-(15) and $\mathcal{D}_n$ be the set of extreme points of the polyhedron defined by $\mathcal{P}_n = \prod_{q \in \mathcal{Q}} \mathcal{P}_n^q$.

A first possible way of formulating the Benders master problem consists in introducing continuous variables $\theta_n, n \in \mathcal{T}$, each one representing the coverage of all paths $q \in \mathcal{Q}_n$ at node n. Problem (1)-(6) is then reformulated as the following MILP, denoted MP1.

$$\text{maximize} \quad \sum_{n \in \mathcal{T}} \pi_n \theta_n \tag{16}$$

subject to

$$\sum_{i \in \mathcal{V}} x_n^i = p_{t_n} \qquad \forall n \in \mathcal{T} \tag{17}$$

$$x_n^i \geq x_{a_n}^i \qquad \forall n \in \mathcal{T}, i \in \mathcal{V} \tag{18}$$

$$\theta_n \leq \sum_{q \in \mathcal{Q}_n} f_n^q \left[\sum_{\alpha \in \mathcal{A}^q} \left(\sum_{i \in \mathcal{K}^{q,\alpha}} x_n^i \right) \tilde{\mu}_n^{q,\alpha} + \tilde{\nu}_n^q \right] \qquad \forall n \in \mathcal{T}, \forall (\tilde{\mu}, \tilde{\nu}) \in \mathcal{D}_n \tag{19}$$

$$x_n^i \in \{0,1\} \qquad \forall n \in \mathcal{T}, i \in \mathcal{V} \tag{20}$$

A second possible way consists in introducing disaggregated continuous variables $\theta_n^q, n \in \mathcal{T}, q \in \mathcal{Q}_n$, each one representing the coverage of path q at node n. Problem (1)-(6) is then reformulated as the following MILP, denoted MP2.

$$\text{maximize} \quad \sum_{n \in \mathcal{T}} \pi_n \sum_{q \in \mathcal{Q}_n} f_n^q \theta_n^q \tag{21}$$

subject to

$$\sum_{i \in \mathcal{V}} x_n^i = p_{t_n} \qquad \forall n \in \mathcal{T} \tag{22}$$

$$x_n^i \geq x_{a_n}^i \qquad \forall n \in \mathcal{T}, i \in \mathcal{V} \tag{23}$$

$$\theta_n^q \leq \sum_{\alpha \in \mathcal{A}^q} \left(\sum_{i \in \mathcal{K}^{q,\alpha}} x_n^i \right) \tilde{\mu}_n^{q,\alpha} + \tilde{\nu}_n^q \qquad \forall n \in \mathcal{T}, \forall q \in \mathcal{Q}_n, \forall (\tilde{\mu}, \tilde{\nu}) \in \mathcal{D}_n^q \tag{24}$$

$$x_n^i \in \{0,1\} \qquad \forall n \in \mathcal{T}, i \in \mathcal{V} \tag{25}$$

In both cases, problem (1)-(6) is reformulated as a mixed-integer linear program which involves a huge number of constraints (19) (resp. (24)) and is therefore computationally intractable as such. The idea of the Benders decomposition algorithm is thus to remove all these constraints from the formulation and to iteratively add only a small subset of them over the course of the algorithm.

In a classical implementation, the algorithm starts with the resolution of a restricted master problem, i.e. of problem MP1 (resp. MP2) in which all constraints (19) (resp. constraints (24)) have been removed. At each iteration of the algorithm, given the current master problem solution $\widehat{x}$, the dual sub-problems $DSP_n^q(\mu, \nu|\widehat{x}), \forall n \in \mathcal{T}, \forall q \in \mathcal{Q}_n$, are solved to optimality. This provides extreme points $(\widehat{\mu}, \widehat{\nu})$, which are used to identify violated optimality cuts of type (19) (resp. (24)). If at least one violated cut is found, it is added to the restricted master problem and the process is repeated. If no violated cut is found, the current solution $\widehat{x}$ is optimal for problem (1)-(6) and the algorithm stops.

However, repeatedly solving the restricted master problem to optimality can be computationally slow as it involves solving a series of mixed-integer linear programs of increasing size. Therefore, following the standard practice in mixed-integer linear programming, we instead embed the generation of Benders inequalities within a branch-and-bound algorithm, leading to a branch-and-cut algorithm. In this algorithm, a branch-and-bound tree search is done over the binary variables of the master problem. Each time a potential feasible incumbent solution is found, the cutting plane algorithm is executed and optimality cuts are added to the master problem formulation. In our implementation using CPLEX solver, we use LazyConstraintCallback routines to ensure that we search for cuts when an integer feasible solution is found by the Branch & Bound algorithm.

Note that both formulations of the master problem lead to a multi-cut implementation of the Benders decomposition algorithm as several cuts may be added to the formulation each time a callback routine is used. However, the strategies to generate these optimality cuts will significantly differ. More precisely, each time the callback routine is called, an algorithm based on formulation MP1 will add a moderate number of cuts of type (19) (at most one per each node of the scenario tree) whereas an algorithm based on formulation MP2 will add a large number of cuts (24) (at most one per each node and each path). Hence, the size of the linear program to be used at each node of the Branch & Bound search tree is likely to increase more rapidly when using formulation MP2 than when using formulation MP1, which may negatively impact the overall computation time. However, the disaggregated cuts (24) are likely to be stronger, i.e. to be more effective at representing the objective functions of the sub-problems in the master problem, than the aggregated cuts (19). This may enable an algorithm based on formulation MP2 to converge in a smaller number of iterations, i.e. after a smaller number of calls to the LazyConstraintCallback routine, which may lead to a reduced computation time.

Following this discussion, we note that it is difficult to predict the relative performance of formulations MP1 and MP2 without carrying out numerical experiments. We thus implemented and tested both versions of the algorithm. As will be shown by the numerical results presented in Section 6, it seems that in general, an algorithm based on formulation MP2 outperforms an algorithm based on formulation MP1.

4.3. *Optimality cut generation*

4.3.1. *Pareto-optimal cut*

One potential issue when implementing the Benders decomposition algorithm described above is that there are several alternative dual optimal solutions, i.e. several alternative extreme points $(\widehat{\mu}, \widehat{\nu})$, that may be used to generate optimality cuts: see Subsection 4.1. Arslan and Karasan [2] propose to use the concept of dominating cut introduced by Magnanti and Wong [34] to identify, among all the cuts that could potentially be generated during an iteration, one cut that may help decrease the number of iterations of the algorithm. In what follows, we recall the main ideas underlying the Magnanti-Wong method. We focus on explaining how this method may be used to generate Pareto-optimal disaggregated cuts of type (24) in a Benders decomposition algorithm based on formulation MP2. The method can then be straightforwardly adapted to generate Pareto-optimal aggregated cuts of type (19) in a Benders decomposition algorithm based on formulation MP1.

Let $\mathcal{X} = \{x \in \{0,1\}^{VT}$s.t. (22)-(23)$\}$ be the feasible space of the initial restricted problem and let C_n^q be the function defined as: $C_n^q(x, \mu, \nu) = \left(\sum_{i \in \mathcal{K}^{q,\alpha}} x_n^i\right)\mu_n^{q,\alpha} + \nu_n^q$, for $n \in \mathcal{T}$ and $q \in \mathcal{Q}_n$.

As defined in [34], the cut generated by the dual solution $(\widehat{\mu}, \widehat{\nu})$ is said to dominate the cut generated by the dual solution $(\tilde{\mu}, \tilde{\nu})$ if and only if $C_n^q(x, \widehat{\mu}, \widehat{\nu}) \leq C_n^q(x, \tilde{\mu}, \tilde{\nu})$ for all $x \in \mathcal{X}$ with a strict inequality for at least one point in $\mathcal{X}$. A cut is called Pareto-optimal is no other cuts dominate it.

When applied to our problem, Theorem 1 of Magnanti and Wong [34] states that given an integer feasible solution of the initial restricted master problem $\widehat{x} \in \mathcal{X}$, a core point $\overline{x}$ in the relative interior of the convex hull of $\mathcal{X}$, any optimal solution (μ, ν) of the dual sub-problem $DSP_n^q(\mu, \nu|\widehat{x})$ minimizing $C_n^q(\overline{x}, \mu, \nu)$ provides a Pareto-optimal cut. The proof is basically made by contradiction, i.e. by showing that if a solution (μ, ν) complying with these conditions were not Pareto-optimal, we would obtain two contradictory inequalities. The reader is referred to Magnanti and Wong [34] for more detail.

This means that a Pareto-optimal cut can be obtained by solving the following linear problem program denoted MW_n^q:

$$\text{minimize} \qquad C_n^q(\overline{x}, \mu, \nu) = \sum_{\alpha \in \mathcal{A}^q} \left[\left(\sum_{i \in \mathcal{K}^{q,\alpha}} \overline{x}_n^i \right) \mu_n^{q,\alpha} \right] + \nu_n^q \tag{26}$$

subject to

$$\sum_{\alpha \in \mathcal{A}^q} \mu_n^{q,\alpha} + \nu_n^q \geq 1 \tag{27}$$

$$\mu_n^{q,\alpha} \geq 0 \qquad \forall \alpha \in \mathcal{A}^q \tag{28}$$

$$\nu_n^q \geq 0 \tag{29}$$

$$C_n^q(\widehat{x}, \mu, \nu) = opt(DSP_n^q(\mu, \nu|\widehat{x})) \tag{30}$$

In this optimization problem, constraints (27)-(29) ensure that the obtained vector (μ, ν) is a feasible solution of the dual sub-problem $DSP_n^q(\mu,\nu|\widehat{x})$ whereas constraint (30) guarantees that (μ,ν) belongs to the set of optimal solutions of $DSP_n^q(\mu,\nu|\widehat{x})$. Finally, the objective function (26) aims at identifying, among all the optimal solutions of $DSP_n^q(\mu,\nu|\widehat{x})$, the one(s) minimizing $C_n^q(\overline{x},\mu,\nu)$.

We note that any core point $\overline{x}$ may provide a Pareto-optimal cut. In our experiments, similarly to [2], we initialize the core point $\overline{x}$ as $\overline{x}_n^i = \frac{p_{t_n}}{V}, \forall i \in \mathcal{V}, \forall n \in \mathcal{T}$ and update it at each iteration of the algorithm using $\overline{x} = \frac{\overline{x}+\widehat{x}}{2}$.

As explained by Arslan and Karasan [2], problem MW_n^q is a linear program which can be solved by inspection by using the characterization of the optimal solutions of $DSP_n^q(\mu,\nu|\widehat{x})$ discussed in Subsection 4.1. More precisely, we have:

- If $\widehat{y}_n^q = 0$, let $\alpha_{PO} = \text{argmin}\{\sum_{i\in\mathcal{K}^{q,\alpha}} \overline{x}_n^i, \alpha \in \mathcal{A}_n^{q,0}(\widehat{x})\}$.

  The following solution is optimal for MW_n^q :
  $\widehat{\nu}_n^q = 0$; $\widehat{\mu}_n^{q,\alpha_{PO}} = 1$; $\widehat{\mu}_n^{q,\alpha} = 0, \forall \alpha \in \mathcal{A}^q \setminus \{\alpha_{PO}\}$

- If $\widehat{y}_n^q = 1$
  - if $|\mathcal{A}_n^{q,1}(\widehat{x})| = 0$
    The following solution is optimal for MW_n^q :
    $\widehat{\nu}_n^q = 1$; $\widehat{\mu}_n^{q,\alpha} = 0, \forall \alpha \in \mathcal{A}^q$
  - if $|\mathcal{A}_n^{q,1}(\widehat{x})| > 1$, let $\alpha_{PO} = \text{argmin}\{\sum_{i\in\mathcal{K}^{q,\alpha}} \overline{x}_n^i, \alpha \in \mathcal{A}_n^{q,1}(\widehat{x})\}$.
    * if $\sum_{i\in\mathcal{K}^{q,\alpha_{PO}}} \overline{x}_n^i < 1$
    The following solution is optimal for MW_n^q :
    $\widehat{\nu}_n^q = 0$; $\widehat{\mu}_n^{q,\alpha_{PO}} = 1$; $\widehat{\mu}_n^{q,\alpha} = 0, \forall \alpha \in \mathcal{A}^q \setminus \{\alpha_{PO}\}$
    * else the following solution is optimal for MW_n^q :
    $\widehat{\nu}_n^q = 1$; $\widehat{\mu}_n^{q,\alpha} = 0, \forall \alpha \in \mathcal{A}^q$

In our cut generation scheme, we will use this Magnanti-Wong method to add cuts of type (19) or (24) at each iteration of the Benders decomposition algorithm.

4.3.2. Cut generation strategy

We now provide a detailed description of the corresponding cut generation algorithms, CGAlg1 and CGAlg2: see Algorithms 1 and 2. Note that algorithm CGAlg1 can be understood as the direct extension to the multi-period stochastic case of the Pareto-optimal single-cut implementation presented in [2]. However, algorithm CGAlg2 significantly differs from the multi-cut implementation presented in [2]. Namely, following this implementation, at a given iteration of the algorithm, for each node and each path, all potential cuts of type (24) found when solving the dual subproblems, i.e. one cut for each optimal dual solution identified in the characterization of Subsection 4.1, would be added to the restricted master problem, even if these cuts are not violated by the current master problem solution. In contrast, in the proposed algorithm CGAlg2, for each node and each path, at most one cut is generated (the Pareto-optimal cut), and this only if the cut is violated by the current master problem solution.

Algorithm 1: CGAlg1

```
1 Let $(\widehat{x}, \widehat{\theta})$ be the current solution of the master problem.
2 Set $\Xi = \emptyset$
3 for each $n \in \mathcal{T}$ do
4     for each $q \in \mathcal{Q}$ do
5         compute $\widehat{y}_n^q$ using (11)
6         find the Pareto-optimal cut defined by dual values $\widehat{\mu}^q, \widehat{\nu}^q$ using the procedure described in
            Subsection (4.3.1)
7     if $\widehat{\theta}_n < \sum_{q \in \mathcal{Q}} f_n^q \left[ \sum_{\alpha \in \mathcal{A}^q} \left( \sum_{i \in \mathcal{K}^{q,\alpha}} \widehat{x}_n^i \right) \widehat{\mu}_n^{q,\alpha} + \widehat{\nu}_n^q \right]$, then
8         add the corresponding cut to $\Xi$
```

Algorithm 2: CGAlg2

```
1 Let $(\widehat{x}, \widehat{\theta})$ be the current solution of the master problem.
2 Set $\Xi = \emptyset$
3 for each $n \in \mathcal{T}$ do
4     for each $q \in \mathcal{Q}$ do
5         compute $\widehat{y}_n^q$ using (11)
6         find the Pareto-optimal cut defined by dual values $\widehat{\mu}^q, \widehat{\nu}^q$ using the procedure described in
            Subsection (4.3.1)
7         if $\widehat{\theta}_n^q < \sum_{\alpha \in \mathcal{A}^q} \left( \sum_{i \in \mathcal{K}^{q,\alpha}} \widehat{x}_n^i \right) \widehat{\mu}_n^{q,\alpha} + \widehat{\nu}_n^q$, then
8             add the corresponding cut to $\Xi$
```

As shown by the numerical results presented in Section 6, the Benders decomposition approach discussed in this section is capable of providing optimal solutions of medium-size instances within a computation time significantly shorter than the one needed by a stand-alone mathematical programming solver. However, it fails to provide solutions for large instances, in particular for instances involving a long planning horizon. We thus investigate in the next section a metaheuristic method based on a genetic algorithm in order to find good quality feasible solutions of problem (1)-(6) in reduced computation times.

5. Genetic Algorithm

Genetic Algorithms (GAs) are adaptive heuristic search algorithms based on the evolutionary ideas of the natural selection and genetics. In contrast to local search methods, GAs allow to effectively explore a large search space in order to avoid local optima. In this section, we explain how we implemented the various components of a genetic algorithm to exploit as best as possible the specific features of our problem.

5.1. *Encoding of individuals*

The encoding of solutions is a key element in a genetic algorithm. In our case, each solution is coded as a chromosome that indicates the stations open at each node of the scenario tree. A chromosome is composed of T sub-individuals, each one corresponding to a node of the scenario tree and including V genes. At tree node n, each gene S_n^i is a binary value indicating whether a station is opened at vertex i (1) or not (0). A chromosome thus involves VT binary elements. Figure 3 illustrates a representation of a solution for the scenario tree given in Figure 2.

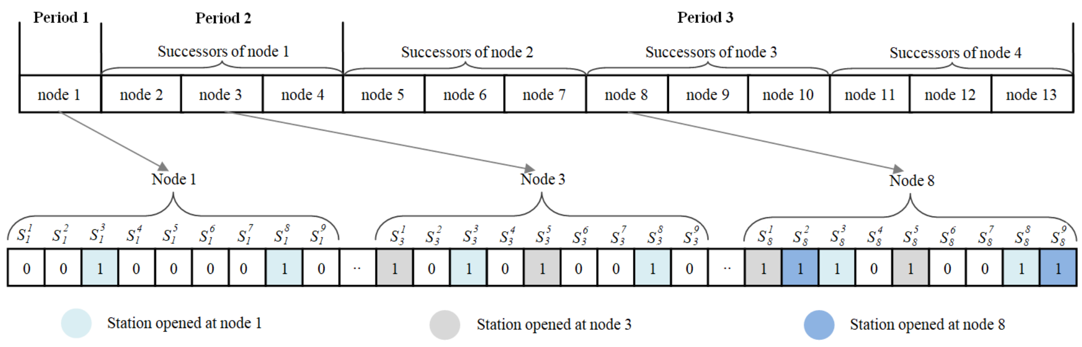

Figure 3: Example of chromosome representation ($k = 3, H = 3, T = 13$)

To ensure the feasibility of each solution, two main conditions should be verified when building a chromosome: 1-the number of open stations at node n must be equal to p_{t_n} 2- if a station is open at node n, it must be maintained open at all nodes successors of n in the tree. In the example shown by Figure 3, the maximum number of open stations is set to 2 in the first period, 4 in the second period and 6 in the third period. One can see that two stations are opened at vertices 3 and 8 for node 1, then two stations are added at vertices 1 and 5 for node 3 and finally two stations are opened at vertices 2 and 9 for node 8. Notice that node 3 is a successor of node 1 (respectively node 8 is a successor of node 3) and thus stations open at node 1 (respectively at node 3) must remain open at node 3 (respectively at node 8).

5.2. *Initial population*

The first step of the genetic algorithm consists of generating an initial population, i.e. a set of feasible solutions for problem (1)-(6). It is an important step as it affects the search for several iterations and often has an influence on both the final solution quality and the total computation time. Several strategies might be used to build an initial population. A first option could be to rely on a pure random generation, which in our case, would consist in randomly selecting, for each individual, the locations to be used at each node among the whole set of V vertices. However, this might lead to very poor quality initial solutions, in particular if some stations are opened at vertices belonging to paths on which the traffic intensity is very low. Hence, we propose in this subsection an improved initial population generation scheme in which for each individual, the locations to be used at each node are randomly chosen among a restricted set of 'promising' vertices.

More precisely, for each node $n \in \mathcal{T}$, we create a subset $LC(n) \subset \mathcal{V}$ of vertices that can be seen as a list of 'promising' potential locations for stations at tree node n. To build $LC(n)$, we start by assigning a score to each vertex v representing the ability of a station located in v to cover the recharging demand. Then, we rank the vertices in the decreasing order of their scores and we select a fixed number $S_{LC(n)}$ of vertices, among those having the highest scores, to be included in the subset $LC(n)$. The size $S_{LC(n)}$ of $LC(n)$ is computed as follows:

$$S_{LC(n)} = p_{t_n} * (t_n + z) \qquad\qquad z \in \mathbb{N} \tag{31}$$

In order to diversify the set of potential locations at a given tree node n, we use four strategies to compute the score assigned to each vertex v:

1. We focus only on the recharging demand at period t_n and consider the total traffic flow going through vertex v at node n. The score of a vertex v is thus computed as the sum of the traffic intensity f_n^q on the origin-destination paths going through vertex v. The score of v is given by equation (32).

$$SCO_1(v) = \sum_{q \in \mathcal{Q}_n | v \in \mathcal{V}^q} f_n^q \tag{32}$$

2. We take into account the total recharging demand on the time interval $[t_n;\ H]$ and consider the total expected traffic flow going through vertex v, at node n and all its successors in the scenario tree. The score of a vertex v is thus computed by equation (33).

$$SCO_2(v) = \sum_{n' \in (n \cup succ_n)} \ \sum_{q \in \mathcal{Q}_{n'} | v \in \mathcal{V}^q} \pi_{n'} f_{n'}^q \tag{33}$$

3. We focus only on the recharging demand at period t_n and consider the traffic on the arcs (j, l) that vertex v can recharge, i.e. on the arcs (j, l) such that it exists at least one path q with $v \in \mathcal{K}^{q;(j,l)}$. The score of a vertex v is computed by equation (34).

$$SCO_3(v) = \sum_{q \in \mathcal{Q}_n} \ \sum_{(j,l) | v \in \mathcal{K}^{q;(j,l)}} f_n^q \tag{34}$$

4. We take into account the total recharging demand on the time interval $[t_n; H]$ and consider the total expected traffic flow on the arcs (j, l) that vertex v can recharge, at node n and all its successors in the scenario tree. The score of a vertex v is computed by equation (35).

$$SCO_4(v) = \sum_{n' \in (n \cup succ_n)} \ \sum_{q \in \mathcal{Q}_{n'}} \ \sum_{(j,l) | v \in \mathcal{K}^{q;(j,l)}} \pi_{n'} f_{n'}^q \tag{35}$$

Each of the four scoring strategies produces a list $LC_\beta(n)$, β=1..4, of potential station locations at tree node n. The four lists are used equitably to generate initial feasible solutions, i.e. each quarter of the initial population is built using a different list. In order to choose the locations of the stations to be opened at a tree node n, we use two approaches. The first approach randomly selects $p_{t_n} - p_{t_{a_n}}$ vertices (i.e. the number of new stations to open in period t_n) from a given list. The second approach selects the first $(p_{t_n} - p_{t_{a_n}})/2$ vertices from the top of the list (i.e. the stations having the highest scores), and the remaining vertices are selected randomly from the same list.

5.3. *Fitness function*

The role of the fitness function is to evaluate the quality of each chromosome, i.e. of each feasible solution, generated by the algorithm. In genetic algorithms, the evaluation of the population usually represents a large portion of the total computational effort. An efficient implementation of the fitness function is thus a key element to ensure the efficiency of the algorithm.

In our case, the quality of a given solution is defined as the corresponding value of the objective function (1), i.e. of the expected coverage of the recharging demand over the whole planning horizon. Thus, computing the fitness function of a chromosome consists in determining, for each node n of the scenario tree, whether each path $q \in \mathcal{Q}_n$ is covered by the stations open at node n or not. We propose two different methods to check the path coverage. In both methods, we exploit the fact that, as the stations open at node n remain open at all nodes in $succ_n$, a path covered at node n is also covered at all nodes in $succ_n$ to avoid checking the coverage of all paths at all tree nodes. A first simple way of computing the fitness function is thus to check the coverage, for each $n \in \mathcal{T}$, of all paths q uncovered at the parent node a_n and such that $f_n^q > 0$. We therefore check whether the distance between two consecutive stations on path q is always less or equal than the range R. To ensure the refueling of round trips, we consider separately the situation where there is no station opened at the origin O^q and the destination D^q. In case there is no station at O^q, we check if the distance from O^q to the first station encountered when traveling to D^q is less or equal than $R/2$. Similarly, in case there is no station at D^q, we check if the distance from D^q to the first station encountered when traveling back to O^q is less or equal than $R/2$. This first method to check the path coverage is detailed in Algorithm 3.

However, this straightforward implementation was found to be very time consuming, leading to a significant reduction in the overall computational efficiency of the algorithm. We thus developed an arc covering based approach in order to decrease the computation time needed to evaluate each chromosome. The main idea is to exploit the fact that the set of opened stations at a given node n contains the set of opened stations at node a_n, predecessor of node n in the scenario tree, to avoid checking the coverage of as much paths as possible at each node of the scenario tree. This can be done as follows. In a pre-optimization step, we build, for each vertex $i \in \mathcal{V}$, the set $\mathcal{L}^i$ of arcs that a station located at i can cover. More precisely, if $i \in \mathcal{K}^{q;(j,l)}$, i.e. if station i can cover the arc (j, l) of path q, we add the element $(q; (j, l))$ to the set $\mathcal{L}^i$. Note that this step has to be carried out only once, before actually running the genetic algorithm. Then, each time a chromosome needs to be evaluated, we carry out the procedure described in Algorithm 4 to compute the fitness value. Note that if the arc (j, l) of path q is covered by the stations opened at node a_n, it will also be covered by the stations opened at node n. Consequently, if a path q is covered at node a_n, it is also covered at node n. At the root node of the scenario tree, we thus check the coverage of each arc of each path and record the information in table $ArcCov$. Then, at each node n different from the root node, we use the information of the predecessor a_n and the precomputed sets $\mathcal{L}^i$ to build the arc coverage table ArcCov$_n$. We then only check the coverage of the paths which are currently not covered and for which at least one arc is newly covered by the stations opened at node n.

Algorithm 3: First approach to compute the fitness function: Distance-based evaluation

```
1  Parameters:
2  FC: Total expected flow covered
3  Stations: Set of vertices on which a station is open at the current node of the tree
4  PathCov$_n^q$ : Binary value representing the coverage of path $q$ at tree node $n$
5  Algorithm:
6  FC := 0
7  for $n = 1 ... T$ do
8      Stations := $\emptyset$
9      for each vertex $i$ such that $S_n^i = 1$, i.e. for each station open at node $n$, do
10         add $i$ to set Stations;
11     for each path $q \in \mathcal{Q}_n$ do
12         if $(f_n^q > 0)$ then
13             if $n = 1$ then
14                 PathCov$_n^q$ $:= 0$
15             else
16                 PathCov$_n^q$ := PathCov$_{a_n}^q$
17             if PathCov$_n^q$ $= 1$ then
18                 FC$+= \pi_n f_n^q$
19             else
20                 Stations$^q$:= Subset of Stations belonging to path $q$
21                 Stations$^q[first]$:= First station encountered on path $q$ on the way from $O^q$ to $D^q$
22                 Stations$^q[last]$:= Last station encountered on path $q$ on the way from $O^q$ to $D^q$
23                 if |Stations$^q$| $> 1$ then
24                     if it exists one pair of consecutive stations located at a distance greater than $R$ from
                         one another then
25                         Go to next path (since the current path $q$ cannot be covered)
26                 if $O^q \notin$ Stations$^q$ then
27                     if distance from $O^q$ to Stations$^q[first]$ is larger than $\frac{R}{2}$ then
28                         Go to next path (since the current path $q$ cannot be covered)
29                 if $D^q \notin$ Stations$^q$ then
30                     if distance from $D^q$ to Stations$^q[last]$ is larger than $\frac{R}{2}$ then
31                         Go to next path (since the current path $q$ cannot be covered)
32                 PathCov$_n^q$ $:= 1$
33                 FC$+= \pi_n f_n^q$
```

Algorithm 4: Second approach to compute the fitness function: Arc-covering based evaluation

```
Parameters:
FC: Total expected flow covered
PathCov$_n^q$ : Binary value representing the coverage of path $q$ at tree node $n$
ArcCov$_n^{q;(j,l)}$ : Parameter indicating the coverage of an arc $(j,l)$ of a path $q$ at a tree node $n$
PathsToBeChecked: The set of paths whose coverage must be checked at the current tree node n
Algorithm:
FC := 0
Initialization at root node ($n = 1$) of the scenario tree:
for each path $q \in \mathcal{Q}_1$ do
    for each arc $(j,l) \in \mathcal{A}^q$ do
        ArcCov$_1^{q;(j,l)}$ := 0
for each vertex $i$ such that $S_1^i = 1$, i.e. for each station opened at root node do
    if the element $(q;(j,l))$ belongs to $\mathcal{L}^i$ then
        ArcCov$_1^{q;(j,l)}$ $:= 1$
    for each path $q \in \mathcal{Q}_1$ do
        if $(f_1^q > 0)$ then
            PathCov$_1^q$ $:= 0$
            if each arc $(j,l) \in \mathcal{A}^q$ is such that ArcCov$_1^{q;(j,l)}$ $= 1$ then
                PathCov$_1^q$ $:= 1$
                FC$+ = f_1^q$
for $n = 2...T$ do
    for each path $q \in \mathcal{Q}_n$ do
        PathCov$_n^q$ := PathCov$_{a_n}^q$
        for each arc $(j,l) \in \mathcal{A}^q$ do
            ArcCov$_n^{q;(j,l)}$ := ArcCov$_{a_n}^{q;(j,l)}$
    PathsToBeChecked = $\emptyset$
    for each vertex $i$ such that $S_n^i = 1$ and $S_{a_n}^i$ = 0, i.e. newly opened station at $n$, do
        for each element $(q;(j,l))$ in the list $\mathcal{L}^i$ do
            if ArcCov$_n^{q;(j,l)}$ $= 0$ then
                ArcCov$_n^{q;(j,l)}$ $:= 1$
                if PathCov$_n^q$ $= 0$ AND path $q$ is not in set PathsToBeChecked then
                    add path $q$ to set PathsToBeChecked
    for each path $q \in \mathcal{Q}_n$ such that PathCov$_n^q$ $= 1$ do
        FC+= $\pi_n f_n^q$
    for each path $q \in$ PathsToBeChecked do
        if $(f_n^q > 0)$ then
            if each arc $(j,l) \in \mathcal{A}^q$ is such that ArcCov$_n^{q;(j,l)}$ $= 1$ then
                PathCov$_n^q$ $:= 1$
                FC += $\pi_n f_n^q$
```

5.4. Genetic operators

A genetic algorithm is an iterative process in which new individuals are generated and evaluated at each iteration. The generation step includes a selection phase and a recombination phase using crossover and mutation operators. When applying these operators, we have to consider two important features of our problem in order to ensure the feasibility of the solutions built. Firstly, the number of stations opened at each node n of the tree must be equal to p_{t_n}. Secondly, if a station is opened at a tree node n, it must be maintained opened at all nodes successors of n in the scenario tree. Therefore, deciding to close or open stations at a node n requires changes to the set of stations opened at the predecessors and successors of n.

5.4.1. Crossover

The crossover operator selects two parents from the population and recombines them in order to generate two new individuals (the crossovers). We perform a 1-point crossover by considering a single common position CP, index of the period up to which the crossovers will take place. The position CP is generated randomly in the interval [2,H], then, the two parts of parent 1 and parent 2 located before CP are swapped. For example, if CP is set to two, the swapping operation is performed for period 1 only (see example in Figure 4). In order to build the sub-individuals situated after the crossover point CP for crossover 1 (respectively crossover 2), we first copy the stations opened at the nodes located before CP to all their successors, to ensure the feasibility of the solution. Then, to reach the number of stations required at each period for crossover 1 (respectively crossover 2), we complete the set of stations at the tree nodes located after CP according to the order in which stations appear in parent 1 (respectively parent 2). In order to better explain the crossover operation, let us consider a crossover between parent 1 and parent 2 as illustrated in Figure 4. The gray lines show the crossover points CP (CP=2 in the example). To obtain crossovers 1 and 2, we first swap the sub-individuals of parent 1 and parent 2 located before the crossover point ($CP = 2$), then we complete the remaining sub-individuals (located after CP) as follows. For crossover 1 (respectively crossover 2), stations opened at the sub-individual located before CP (i.e. node 1) are copied to all its successors (i.e. node 2, node 3, ..., node 13). Then, in order to obtain the required number of stations at each sub-individual located after CP, we follow the order in which stations appear in parent 1 (respectively parent 2). For the example shown in Figure 4, we illustrate this reasoning using the branch 1$\rightarrow$3$\rightarrow$8 of the tree as node 3 is a successor of node 1, and node 8 is a successor of node 3, while node 8 is a leaf of the tree. Note that since the chromosome is long, it is not possible to display the other branches of the tree on the same figure.

5.4.2. Mutation

The objective of mutations is to provide the genetic algorithm with the necessary diversification for an efficient exploration of the search space. The mutation operator performs a local modification on a single chromosome. For the selected chromosome, a node $n \in \mathcal{T}$ is randomly selected to perform a mutation between two positions $i, j \in \mathcal{V}$, such that S_n^i=1 and S_n^j=0 (i.e. i corresponds to an opened station and j corresponds to

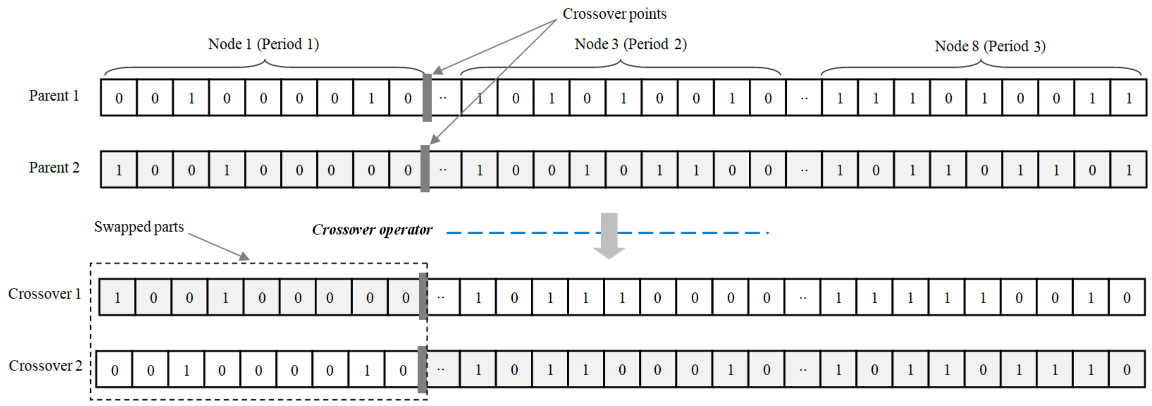

Figure 4: Example of a crossover operation for V=9, T=13, $p_1 = 2$, $p_2 = 4$, $p_3 = 6$, and $CP = 2$

a closed station at node n). Vertex j is randomly selected from one of the lists $LC_\beta(n)$ containing the set of 'promising' locations for node n. To ensure the feasibility of the solution after swapping the binary values at positions i and j for node n, we also swap the values for all nodes predecessors and successors of n. In practice, for successors of n, when moving forward to a leaf of the tree, the values of the two positions might become both equal to one so that swapping is not needed in this case. Similarly, for predecessors of n, when moving back from n to the root node of the tree, the values of the two positions might become both equal to zero so that swapping is not needed in this case. However, notice that for any node n' predecessor of n in the tree, where the swapping operation is performed, we have to ensure that the same operation is also performed at all successors of n' in the tree. In the example given by Figure 5, the mutation is performed at tree node 3 between the values at positions 4 (closed station) and 8 (opened station). After swapping the values at positions 4 and 8 for node 3, the same operation is performed at all nodes predecessors of node 3 where a station is opened at position 8 (in this case at node 1 only) and all nodes successors of node 3 where there is no station opened at position 4 (in Figure 5 the example of node 8 is shown). Then, to ensure the feasibility of the solution obtained, the swapping operation performed at node 1 is also performed at all nodes successors of 1 (i.e. 2,4,5,6,...,13). Note that in Figure 5, since the chromosome is long, we show the swapping operations for three nodes only: node 3, its predecessor (node 1) and only one of its successors (node 8).

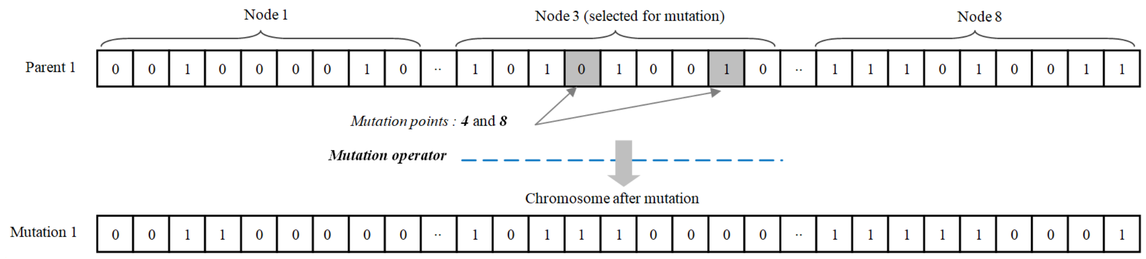
Node 1
Node 3 (selected for mutation)
Node 8
Parent 1
Mutation points : 4 and 8
Mutation operator
Chromosome after mutation
Mutation 1

Figure 5: Example of a mutation operation $V = 9, H = 3, k = 3, P_1 = 2, P_2 = 4, P_3 = 6$

In summary, the proposed genetic algorithm contributes to the existing literature in different ways. First, it investigates different strategies when generating the initial population. Using various approaches to evaluate the ability of each vertex to cover the recharging demand, it selects promising candidates to locate recharging stations. Second, it includes an efficient implementation of the fitness function. This implementation uses an arc covering based approach to significantly decrease the computational effort needed to evaluate the quality of a solution. Thirdly, the crossover and mutation operators are defined in a way to ensure the feasibility of the obtained solutions with regard to the number of stations to be opened in each period and to the constraints forbidding station relocation. As will be shown in Section 6, the proposed GA is capable of providing good quality solutions, in particular for the instances for which the other exact algorithms fail to obtain optimal or feasible solutions.

6. Numerical experiments

In this section, we focus on solving the multi-stage stochastic integer program defined in Section 3. The objective of our numerical experiments is twofold. First, we study the performance of the proposed solution methods, namely the Benders decomposition algorithm and the genetic algorithm, and compare it with the performance of the mathematical solver CPLEX using formulation (1)-(6). Second, we evaluate the benefit from using a multi-stage stochastic model instead of a simpler multi-period deterministic model.

6.1. Test instances

Our numerical experiments are conducted on instances randomly generated as follows.
The road networks are generated according to a procedure similar to the one proposed in [4]. We first generate V nodes whose coordinates are randomly chosen within $[1, 1000]^2$ according to a uniform distribution. The traveling distance between each pair of nodes is computed as the Euclidean distance. We then apply Kruskal algorithm to determine the minimum spanning tree of size $V - 1$: all arcs belonging to this spanning tree are added to the arc set $\mathcal{A}$. We also select V additional arcs to be added to $\mathcal{A}$: these arcs are the shortest potential arcs not yet added to $\mathcal{A}$ and such that the degree of each node stays below four. In our numerical experiments, we consider values for V belonging to $\{250, 500, 750\}$, leading to 3 network sizes. The deterministic range R is set to 250.

The M origin-destination nodes are randomly generated among the nodes belonging to $\mathcal{V}$. This gives a set of $Q = \frac{M(M-1)}{2}$ trips to be covered. The shortest path $(\mathcal{V}^q, \mathcal{A}^q)$ corresponding to origin-destination pair q is determined using Dijkstra algorithm. In our numerical experiments, M is varied in $\{60, 100, 150, 250, 500\}$. The population at each origin/destination node is randomly generated within $[10, 100]$ according to a uniform distribution. The flow f^q of EVs on a path q is determined using the gravity model [13]: $f^q = \frac{P^{O^q} P^{D^q}}{d(O^q, D^q)^2}$ where P^{O^q} and P^{D^q} are the populations at the origin and destination vertices, respectively, and $d(O^q, D^q)$ is the length of the shortest path between the origin O^q and the destination D^q.

We consider a planning horizon of $H \in \{2, 3, 4, 5, 6\}$ periods. The maximum number of stations that can be open at period t is set to $p_t = 3t$. The scenario tree used to represent the stochastic evolution of the set of paths taken by EVs on the road network and the traffic intensity on each path is built as follows. At the root node of the scenario tree, we consider a subset $\mathcal{M}_0$ of the set $\mathcal{M}$ of origin-destination vertices containing the $M_0 = \lfloor \frac{M}{H} \rfloor$ cities with the largest populations. We determine the traffic intensity on each corresponding path, f_0^q, by using the gravity model mentioned above. Then, the set of origin-destination vertices $\mathcal{M}_n$ to be considered at a given node n of the scenario tree includes the set $\mathcal{M}_{a_n}$ considered at its parent node together with a fixed number $\lfloor \frac{M}{H} \rfloor$ of new origin-destination vertices, that are randomly chosen amongst the largest cities remaining in the set $\mathcal{M} \setminus \mathcal{M}_{a_n}$. For the paths q which were already considered in the set $\mathcal{M}_{a_n}$, the flow f_n^q is obtained by increasing $f_{a_n}^q$ by a percentage randomly generated in the interval [0%, 30%]. For the paths newly appeared at node n, the flow f_n^q is set to the value provided by the gravity model mentioned above. This approach is applied at all nodes of the scenario tree related to periods 2 to $H - 1$. For the leaf nodes related to period H, we set $\mathcal{M}_n = \mathcal{M}$ and use the same procedure as for the other interior nodes to determine the flows f_n^q. Note that in our numerical experiments, the number of immediate successors of a node, k, is set to 3 and the transition probability from a parent node to each of its successors is set to $\frac{1}{3}$. Hence, the probability of occurrence of node n is given by: $\pi_n = (\frac{1}{3})^{t_n - 1}$.

We first define a reference set of 10 randomly generated instances in which $V = 250$, $M = 60$ and $H = 3$. Then, in order to assess the impact of some of the problem parameters on the performance of the studied algorithms, we create 11 additional sets of 10 instances, each time varying the value of a single parameter. This leads to a total of 120 instances.

6.2. *Numerical results*

In this subsection, we report the results of numerical experiments conducted on a large set of randomly generated instances. We aim at comparing the performance of the proposed solution approaches, in terms of solution quality and computation time, with the one of the mathematical solver CPLEX using the ILP formulation (1)-(6). We also seek to evaluate the impact on the performance of our algorithms of three problem input parameters which have a direct influence on the size of formulation (1)-(6), namely the road network size V, the number of origin-destination vertices M and the length of the planning horizon H.

In order to achieve this, we solve each considered instance with the following solution algorithms:

- 'CPLEX solver' corresponds to the direct resolution of formulation (1)-(6) by the solver CPLEX,
- 'Benders 1' corresponds to an implementation of the Benders decomposition algorithm discussed in Section 4 in which the cut generation algorithm CGAlg1 is used to add at most one optimality cut per tree node at each iteration of the algorithm,
- 'Benders 2' corresponds to an implementation of the Benders decomposition algorithm discussed in Section 4 in which the cut generation algorithm CGAlg2 is used to add multiple optimality cut per tree node

(more precisely at most one per path to be covered) at each iteration of the algorithm,

- 'GA$_1$' corresponds to the genetic algorithm in which the fitness value of each individual is evaluated using Algorithm 3,

- 'GA$_2$' corresponds to the genetic algorithm in which the fitness value of each individual is evaluated using Algorithm 4.

All solution algorithms were coded in C++ language. The resolution of the ILP and MILP models was implemented using the Concert Technology and was carried out by CPLEX version 12.6.0 with the solver default settings. Note that the setting of a genetic algorithm is itself an NP-hard problem. Based on the results of our preliminary experiments, we fixed the parameters of our GAs as follows: the parameter z used in equation (31) is set to 5, the population size to 200, the crossover rate to 100%, the mutation rate to 10% and the maximum number of iterations to 200. For all the algorithms, we set the CPU time limit to 3000 seconds. All tests were carried out on a PC with Intel Core i7-4800MQ CPU (2.70 GHz) and 8GB of RAM, running under Windows 10.

The corresponding results are displayed in Tables 3 to 5. For each set of 10 instances, we report the following information.

- For all algorithms, we provide $\#Feas$, the number of instances for which a feasible solution could be found within the time limit, and CPU, the average computation time of the algorithm. In case the algorithm did not converge within the time limit, the value 3000s was used to compute the average.

- For the direct resolution by CPLEX solver, we report $\#Opt$, the number of optimal solutions found within the time limit, $\#Cst$ the average number of arc covering constraints of type (2) in formulation (1)-(6) and $MIPGap$, the average remaining MIP gap, i.e. the average relative difference between the best integer feasible solution and the best upper bound found by CPLEX solver within the time limit.

- For the Benders decomposition algorithms, in addition to the performance metrics $\#Opt$ and $MIPGap$, we report $\#\ Cuts$, the average number of optimality cuts of type (19) or (24) added to the formulation, and $\#Callbacks$, the average number of lazy constraints callbacks which can be understood as the average number of iterations of the Benders decomposition algorithm.

- For the genetic algorithms, we report GAP$_{opt}$, the relative difference between the value of the optimal integer solution and the value of the best solution found by the GA. In case the optimal integer solution could not be found within the time limit, we report GAP$_{int^*}$ (resp. GAP$_{ub^*}$), the relative difference between the value of the best integer solution (resp. best upper bound) known for the instance and the value of the best solution found by the GA.

6.2.1. Numerical comparison of the solution algorithms

Results from Tables 3 to 5 first show that solving formulation (1)-(6) directly with CPLEX solver is possible only for medium-size instances of the problem. Namely, using this solution approach, we could find within the time limit a feasible solution for only 78 instances and an optimal solution for only 68 instances out of the 120 instances considered. In particular, no feasible solution could be obtained for instances involving a large number ($M = 500$) of origin-destination vertices or a long planning horizon ($H = 6$) due to out-of-memory issues. The main reason for this is that, even if the arc-cover path-cover formulation is a strong and compact formulation of the problem, the number of path covering constraints (2) becomes huge (greater than 10 millions) when the number of paths to be covered or the number of nodes in the scenario tree increases.

Algorithm	**Metrics**	$V = 250$			$V = 500$		
		M			M		
		60	100	250	150	250	500
CPLEX	$\#Cst$	508112	1460117	9044773	4361751	12113034	-
	$\#Feas$	10	10	7	5	0	0
	$\#Opt$	10	10	3	5	0	0
	$MIPGap$ (%)	0	0	62,8	-	-	-
	CPU (s)	26,5	147,4	2633,6	2456,1	> 3000	Out of memory
Benders 1	$Cuts$	889	674	702	748	814	849
	$\#Callbacks$	68	52	54	55	63	66
	$\#Feas$	10	10	10	10	10	10
	$\#Opt$	8	9	8	6	9	7
	$MIPGap$ (%)	0,05	0,1	0,07	0,09	0,02	0,02
	CPU (s)	1005,3	528,7	804,7	1480,3	664,2	1832
Benders 2	$Cuts$	32101	68092	430448	146833	438283	-
	$\#Callbacks$	17	13	9	13	8	-
	$\#Feas$	10	10	10	10	10	0
	$\#Opt$	**10**	**10**	**10**	**10**	**10**	0
	$MIPGap$ (%)	0	0	0	0	0	-
	CPU (s)	**7,1**	**16,2**	**394,1**	**62,3**	**520,8**	> 3000
GA$_1$	$\#Feas$	10	10	0	0	0	0
	GAP$_{opt}$ (%)	4,3	4,4	-	-	-	-
	CPU (s)	1080,2	2220,4	> 3000	> 3000	> 3000	> 3000
GA$_2$	$\#Feas$	**10**	**10**	**10**	**10**	**10**	**10**
	GAP$_{opt}$ (%)	4,4	4,7	5,2	5,3	6,6	5,3
	CPU (s)	**14,1**	**44,2**	**284,1**	**116,2**	**364,1**	**1465**

Table 3: Average performances of algorithms for different sizes of O-D vertices (M) ($V = \{250, 500\};\ H = 3$)

Algorithm	**Metrics**	V		
		250	500	750
CPLEX solver	$\#Cst$	508112	706343	831658
	$\#Opt$	**10**	**10**	**10**
	CPU (s)	26,5	55,8	90,1
Benders 1	$Cuts$	889	898	870
	$\#Callbacks$	68	69	67
	$\#Feas$	10	10	10
	$\#Opt$	8	8	7
	$MIPGap$ (%)	0,05	0,12	0,08
	CPU (s)	1005,3	662,5	1074,9
Benders 2	$Cuts$	32101	37536	34764
	$\#Callbacks$	17	21	20
	$\#Opt$	**10**	**10**	**10**
	$MIPGap$ (%)	0	0	0
	CPU (s)	**7,1**	**11,7**	**13,6**
GA$_2$	$\#Feas$	**10**	**10**	**10**
	GAP$_{Opt}$ (%)	4,4	4,1	5,6
	CPU (s)	**14,1**	**20,5**	**25,8**

Table 4: Average performances of algorithms for different sizes (V) of the network ($M = 60$; $H = 3$)

Results from Tables 3 to 5 also show that Benders 2 algorithm outperforms the resolution by CPLEX solver. Namely, this algorithm could find a feasible solution for 100 instances and an optimal one for 90 instances out of the studied 120 instances. Moreover, the average computation time is significantly reduced from an average of 1462s with CPLEX solver to an average of 843s with Benders 2 algorithm. This can be explained mainly by the fact that the number of cuts of type (24) explicitly added to the formulation by the algorithm is much smaller than the initial number of constraints (2) in the ILP formulation: it namely represents on average less than 5% of the total number of constraints (2).

In contrast, the results related to Benders 1 algorithm are rather negative as, for most instances, it exhibits a poorer performance than Benders 2 algorithm both in terms of the number of feasible/optimal solutions found and in terms of computation time. Namely, Benders 1 algorithm could find an optimal solution for only 72 instances and its average computation time of 1095s is significantly larger than the one of Benders 2 algorithm (843s). A reason of this could be that, in our case, the aggregated cuts are less effective than the disaggregated cuts at representing the objective functions of the sub-problems in the master problem. Thus, even if the number of cuts of type (19) explicitly added to the formulation is much smaller than the number of cuts of type (24), this is not sufficient to reduce the overall computation time as more iterations (i.e. more calls to the callback routine) will be needed to obtain an accurate representation of these objective functions in the master problem.

Benders 1 algorithm performed, however, better than Benders 2 algorithm in a few cases corresponding to the 20 largest instances considered in our numerical experiments. Namely, for the 10 instances involving the largest number of origin-destination vertices (see Table 3 for $V = 500$ and $M = 500$) and the 10 instances involving the longest horizon (see Table 5 for $H = 6$), Benders 1 algorithm provided feasible solutions for all instances whereas Benders 2 algorithm did not provide any feasible solution. Moreover, out of these solutions, 7 were optimal. This might be explained by the fact that for such huge instances (involving up to 14 millions of constraints (2) in the initial MILP formulation - see line 4 of Table 5), even if the disaggregated cuts (24) generated by Benders 2 algorithm represent only a small percentage of the original constraints, their number is still too large to enable CPLEX solver to solve the obtained MILP within the computation time limit. This can be seen e.g. for the instances corresponding to $H = 6$ (see Table 5) for which, on average, Benders 2 algorithm generated 780000 cuts after only two calls to the callback routine. Hence, for these very large instances, even if Benders 1 algorithm leads to a larger number of calls to the routine than Benders 2 algorithm, as the number of generated cuts is much smaller, the size of the MILP to be solved by CPLEX remains tractable so that at least feasible (and sometimes optimal) solutions can be found.

Regarding the genetic algorithms, we first note that the simple path-coverage checking procedure defined in Algorithm 3 is much more time consuming than the arc-coverage path-coverage checking procedure detailed in Algorithm 4. Even for the small instances involving $V = 250$ nodes, $M = 60$ origin-destination vertices and $H \in \{3, 4\}$ periods, the computation time needed by algorithm GA$_1$ is around 60 times larger than the computation time needed by algorithm GA$_2$. This shows the crucial importance of devising a computationally efficient fitness function evaluation such as the one proposed in Algorithm 4 to ensure the practical performance of a genetic algorithm.

Moreover, the results reported in Tables 3 to 5 first show that for instances of medium size, the exact solution methods (especially CPLEX and Benders 2) outperform algorithm GA$_2$ as, within a similar computation time, they provide optimal solutions of the problem whereas GA$_2$ only obtains heuristic solutions within an average optimality gap greater than 4%. However, these results also show that, for the instances involving a long planning horizon ($H \in \{5, 6\}$), CPLEX solver and the Benders decomposition algorithms all struggle at finding optimal or even feasible solutions of the problem within the time limit whereas GA$_2$ provides feasible solutions of the problem within less than 7 minutes. GA$_2$ thus seems to be an interesting alternative in this case as obtaining within a few minutes approximate solutions within an optimality gap of around 4% when no other solution is available might be of great usefulness to managers.

6.2.2. Influence of the number of origin-destination vertices

The significant impact of the number of origin-destination vertices M on the number $Q = \frac{M(M-1)}{2}$ of paths to be covered and thus on the numerical difficulty of the problem can be observed from the results presented in Table 3.

Namely, the computation time required by CPLEX solver to find optimal solutions increases sharply with the value of M. For example, for a network with $V = 250$ vertices, the solver requires 147.4 seconds on average to find optimal solutions when $M = 100$ and $Q = 4950$ whereas it needs more than 2633.6 seconds on average when $M = 250$ and $Q = 31125$. A similar augmentation of the computation time is observed for Benders 2 algorithm. This is explained by the sharp increase in the number of arc covering constraints (2) and, to a lesser extent, in binary coverage decision variables y_n^q when M increases.

Similarly, the performance of our genetic algorithm is also negatively impacted by an increase of M. The resulting growth of the search space induces a deterioration of the quality of the solution provided by GA$_2$ as it becomes more difficult to identify good solutions. Moreover, the average computation time also increases, mainly due to an increase in the time needed to evaluate the fitness value of each individual.

Algorithm		Size of the scenario tree				
	H	2	3	4	5	6
	T	4	13	40	121	364
CPLEX solver	$\#Cuts$	157558	508112	1592793	4890024	14467415
	$\#Feas$	10	10	10	10	0
	$\#Opt$	**10**	**10**	**10**	0	0
	$MIPGap$ (%)	0	0	0	1,7	-
	CPU (s)	**4,7**	**26,5**	**136**	>3000	Out of memory
Benders 1	$\#Cuts$	79	889	8501	33880	32801
	$\#Callbacks$	20	68	179	280	90
	$\#Feas$	10	10	10	10	10
	$\#Opt$	10	8	0	0	0
	$MIPGap$ (%)	0	0,05	0,4	4,6	18,6
	CPU (s)	1,4	1005,3	>3000	>3000	>3000
Benders 2	$\#Cuts$	7565	32101	131963	593094	780978
	$\#Callbacks$	9	17	27	37	2
	$\#Feas$	10	10	10	10	0
	$\#Opt$	**10**	**10**	**10**	0	0
	$MIPGap$ (%)	0	0	0	0,5	-
	CPU (s)	**0,7**	**7,1**	**92,5**	>3000	>3000
GA$_2$	$\#Feas$	**10**	**10**	**10**	**10**	**10**
	GAP$_{opt}$ (%)	3,5	4,2	4,4	-	-
	GAP$_{int*}$ (%)	-	-	-	5,5	1,4
	GAP$_{ub*}$ (%)	-	-	-	6,1	12,2
	CPU (s)	**4,9**	**14,1**	**46,3**	**136**	**392**

Table 5: Average performance of algorithms for different values of the horizon H ($V = 250;\ M = 60$)

6.2.3. *Influence of the graph size*

Regarding the impact of the graph size (V), results from Table 4 show that increasing the network size, and in particular the number of potential locations for charging stations, negatively impacts the performance of our algorithms but in a much less significant way than an increase of M.

Again, Benders 1 algorithm shows poorer performance than Benders 2 algorithm in terms of the number of optimal solutions found and the average computation times. Namely, Benders 1 algorithm could find an optimal solution for only 23 instances among of 30 instances optimally solved by Benders 2 algorithm. Moreover,Benders 1 algorithm requires significantly larger average computation time (914s) compared to Benders 2 algorithm requiring only 11s on average.

6.2.4. Influence of the horizon size

In order to assess the influence of the horizon size H, we carried out experiments on instances in which H varies in the set $\{2, 3, 4, 5, 6\}$ while all other parameters of the problem remain unchanged. Table 5 displays the corresponding results.

Results from Table 5 show that CPLEX solver and Benders 2 algorithm are both capable of providing optimal solutions for a planning horizon up to 4 periods and feasible solutions for a planning horizon up to 5 periods. However, for a planning horizon of 6 periods, both methods fail at providing feasible solutions for the problem. The main reason for this is the fact that the number of nodes T in the scenario tree grows exponentially fast with the number of periods in the planning horizon and that the number of binary variables and constraints in the MILP formulation is proportional to T. For Benders 1 algorithm , optimal solutions were found in only 18 out of 50 instances, related to problems with a small planning horizon ($H \leq 3$). However, feasible solutions were obtained for all instances, including the ones for which the planning horizon involves 6 time periods.

Results from Table 5 also show that, when the horizon length stays below 4 periods, algorithm GA$_2$ is capable of providing feasible solutions within an average gap GAP$_{opt}$ of 4.0% from the optimal solutions, which seems acceptable in view of the problem difficulty. Moreover, GA$_2$ is capable of providing feasible solutions within a reduced computation time (as compared to the other exact solution algorithms) when H is set to 5 or 6 time periods. Note that, for these two cases, it is not possible to compute GAP$_{opt}$ since no guaranteed optimal solutions are provided by the exact approaches. Therefore, in order to get a preliminary assessment of the quality of the GA$_2$ solutions, we compute GAP$_{int^*}$ (resp. GAP$_{ub}$) the gap between the value of the heuristic solution and the value of the best known feasible solution (resp. the value of the best known upper bound) provided by the exact methods. By definition, GAP$_{opt}$ lies within the range $[$GAP$_{int^*}$, GAP$_{ub}]$. However, we note that the value of GAP$_{int^*}$ and GAP$_{ub}$ should be taken carefully as they strongly depend on the quality of best known lower and upper bounds found by the exact methods. For the instances involving $H = 5$ periods, these values are provided by the solution obtained with Benders 2 algorithm. In this case, the average MIP gap Gap$_{MIP}$ is only 0.5%, which means that both the lower and upper bounds are of good quality. We thus get a rather accurate estimation of GAP$_{opt}$ through the range $[$GAP$_{int^*}$, GAP$_{ub}] = [5.5\%, 6.1\%]$. However, for the instances involving $H = 6$ periods, the values of GAP$_{int^*}$ and GAP$_{ub}$ are computed based on the results of Benders 1 algorithm, for which the average MIP Gap is above 18%. The large range $[$GAP$_{int^*}$, GAP$_{ub}] = [1.4\%, 12.2\%]$ obtained in this case makes it difficult to properly estimate the quality of the heuristic solution provided by GA$_2$.

6.2.5. Value of the stochastic solution

The aim of developing a multi-stage stochastic program for the EV charging station location problem is to take into account the uncertainties that may occur over time in the problem modeling in order to make better station location decisions. However, this significantly enhances the difficulty of its numerical resolution, mainly because the resulting combinatorial optimization problem to be solved becomes much larger. It would thus be interesting to evaluate the practical benefit of using the multi-stage stochastic model instead of a simpler multi-period deterministic model. We propose in what follows to obtain a first estimation of this benefit through a rolling horizon simulation.

Each experiment starts with the generation of a 'real' scenario representing the actual realization of the random parameters over the simulation horizon. The experiment then consists in simulating the application of the first-stage planning decisions over H time periods and in comparing the total flow coverage obtained when applying the planning decisions established by the deterministic or the stochastic model. More precisely, at each time period θ of the rolling horizon framework, we solve two problems on an optimization horizon of length H: a deterministic variant of problem (1)-(6) in which the scenario tree is reduced to a single deterministic scenario, and a stochastic variant of problem (1)-(6) based on a scenario tree representing the possible outcomes of the stochastic input parameters. For each model, the station location decisions relative to the first planning period are implemented and the corresponding flow coverage is assessed based on the value of the flows in the 'real' scenario generated at the beginning of the simulation process. The location decisions pertaining to the current period are then introduced as additional constraints in the deterministic and stochastic problems to be solved for the next period and the process is repeated till we reach the end of the simulation horizon.

In our numerical experiments, the 'real' scenario used to assess the location decisions is generated using the same procedure than the one described in Subsection 6.1 to generate the scenario trees. The single scenario used by the deterministic model is built by adding to the set $\mathcal{M}_n$, the origin-destination vertices considered at period t_n, the $\lfloor \frac{M}{H} \rfloor$ cities with the highest population that are not yet considered at the previous periods and by setting the evolution rate of the demand on each path at 15%.

We perform numerical tests on instances randomly generated following the procedure defined in Subsection 6.1. We set $L = 1000$, $V = 250$, $M = 60$, $H = 3$, $R = 250$, $k = 3$, $p_t = 3t$ and consider values of $H \in \{2, 3, 4, 5\}$. For each value of H, we carry out 20 simulation replications and report in Table 6 the average value of the flow coverage obtained when applying the solution of the deterministic model and the solution of the stochastic model. Note that this value is not the objective function of each optimization model, but the sum over time of the true coverage obtained when applying the first-stage decision planning over the 'real' scenario. We also report the value of the flow coverage obtained when applying the deterministic model on the 'real' scenario in order to assess the coverage that would have been obtained if we had had in advance a perfect information on the realization of the stochastic input parameters.

Results from Table 6 enable us to obtain a first assessment of the overall benefit from using a stochastic model. Namely, the actual flow coverage obtained when implementing the solution in a stochastic environment is significantly increased from an average of 63.7% with the deterministic model to an average of 72.9% with the stochastic model. This shows that a station deployment plan established while taking into account the uncertainty on the recharging demand will perform significantly better in a stochastic environment than a deployment plan established while assuming deterministic input data. However, the rather large remaining gap between the stochastic solution value and the perfect information solution value (around 9% of coverage) shows that there is still room for improvement in the stochastic solution quality, which could be potentially obtained e.g. by considering larger scenario trees, i.e. scenario trees with more than 3 children per nodes, in the stochastic model.

	H	**2**	**3**	**4**	**5**
Covered flow (%)	Perfect information solution	78,2	80,1	84,4	85,1
	Deterministic model solution	59,5	61,6	63,1	70,7
	Stochastic model solution	67,8	73,4	71,3	78,9

Table 6: Average performance of the multi-stage stochastic approach using the value of the stochastic solution

7. Conclusion

We studied the problem of locating fast charging stations in electric vehicles systems and focused on taking into account in the problem modeling the dynamic and stochastic aspects of the recharging demand. We proposed a multi-stage stochastic integer programming approach in which the uncertainties on the set of trips taken by EV drivers and on the flow of EVs on each trip are represented through a scenario tree. This led to the formulation of an integer program which can be seen as the extension of the arc-cover path-cover formulation proposed by Capar *et al.* [4] for the single-period deterministic problem to the multi-period stochastic case. In view of the numerical difficulties encountered while solving the problem for medium-size instances, we developed two solution algorithms: an exact solution method based on a Benders decomposition algorithm and a heuristic approach based on a genetic algorithm. Our numerical results showed that both methods performed well as compared to a stand-alone mathematical programming solver both in terms of solution quality and computation time. Moreover, the results of our simulation experiments showed the practical benefit of the proposed multi-stage stochastic programming model as compared to a simpler multi-period deterministic model. Namely, the stochastic programming model provides a charging infrastructure which, when confronted to a stochastic environment, leads to an actual coverage of the recharging demand significantly improved as compared to one obtained with the infrastructure provided by a deterministic model.

This paper provides an ample potential for further study. First, it might be interesting to develop solution approaches capable of solving instances of the problem with larger scenario trees, in particular with scenario trees in which each node has a larger number of children. This would allow to obtain a better approximation of the stochastic input process and would thus lead to charging infrastructures providing a coverage of the demand closer to the one that would be obtained in the perfect information situation. Solving the resulting MILP could be done e.g. by combining the Benders decomposition algorithm proposed in the present paper with a cluster-based decomposition algorithm such as the one recently proposed by Escudero [11].

Second, we would like to point out that the proposed solution algorithms (both the Benders decomposition algorithm and the genetic algorithm) could rather easily be adapted to solve other variants of the stochastic multi-period flow refueling location problem involving realistic complicating features such as multiple types of EVs, multiple types of charging stations or drivers willing to deviate from their shortest paths. These extensions will lead to an increase in the problem size and may thus enhance the numerical difficulty of the problem. But the key property that, for a given charging infrastructure (described by the restricted master problem solution in the BD algorithm and by the chromosome in the GA), the coverage of a trip can be computed independently of the one of the others trips is conserved. As a result, the two proposed algorithms might be quite easily extended to consider these features. However, extending the problem modeling to take into account the limited capacity of the charging stations (expressed e.g. in terms of the maximal number of vehicles that can be charged per hour) and congestion problems at the stations will be less straightforward as in this case, the coverage of a given trip depends on how the other trips use the installed station capacity. An interesting direction for further research would thus be to develop solution algorithms capable of handling this inter-trip dependency in a stochastic environment.

8. References

References

[1] Ahmed, S., King, A. J., Parija, G., 2003. A Multi-Stage Stochastic Integer Programming Approach for Capacity Expansion under Uncertainty. Journal of Global Optimization, 26, 3–24.

[2] Arslan, O., Karasan, O. E., 2016. A Benders decomposition approach for the charging station location problem with plug-in hybrid electric vehicles. Transportation Research Part B, 93, 670-695.

[3] Chan, M., Wong, C., Luo, W., Cheung, B., 2005. Investment Stock Portfolio with Multi-Stage Genetic Algorithm Optimization. In: Abraham A., Dote Y., Furuhashi T., Köppen M., Ohuchi A., Ohsawa Y. (eds) Soft Computing as Transdisciplinary Science and Technology. Advances in Soft Computing, vol 29. Springer, Berlin, Heidelberg.

[4] Capar, I., Kuby, M., Leon, V. J., 2013. An arc cover-path-cover formulation and strategic analysis of alternative-fuel station locations. European Journal of Operational Research, 227(1), 142-151.

[5] Caroe, C.C. and Schultz, R., 1999. Dual decomposition in stochastic integer programming. Operations Research Letters, 24, 37-45.

[6] Carpentier P., Cohen G., Culioli JC., 1995. Stochastic Optimal Control and Decomposition-Coordination Methods Part I: Theory. In: Durier R., Michelot C. (eds) Recent Developments in Optimization. Lecture Notes in Economics and Mathematical Systems, vol 429. Springer, Berlin, Heidelberg

[7] Chung, S. H., Kwon, C., 2015. Multi-period planning for electric car charging station locations: a case of Korean expressways. European Journal of Operational Research, 242(2), 677-687.

[8] De Vries, H., Duijzer, E. 2017. Incorporating driving range variability in network design for refueling facilities. Omega, 69, 102-114.

[9] Dupacová, J., Gröwe-Kuska, N., Römisch, W., 2003. Scenario reduction in stochastic programming. Mathematical Programming, 95 (3), 493–511 .

[10] Egbue, O., Long, S., 2012. Barriers to widespread adoption of electric vehicles: an analysis of consumer attitudes and perceptions. Energy Policy, 48, 717-729.

[11] Escudero, L. F., Garin, A. , Unzeuta, A., 2016. Cluster lagrangean decomposition in multistage stochastic optimization. Computers & Operations Research, 67,48–62.

[12] Fattahi, M., Govindan, K. and Keyvanshokooh, E., 2018. A multi-stage stochastic program for supply chain network redesign problem with price-dependent uncertain demands. Computers and Operations Research, 100, 314-332.

[13] Fotheringham, A. S., O'Kelly, M. E., 1989. Spatial Interaction Models: Formulations and Applications. Kluwer Academic Publishers.

[14] Gade D., Hackebeil G., Ryan S.M., Watson, J.P., Wets, R.J-B., Woodruff, D.L., 2016. Obtaining lower bounds from the progressive hedging algorithm for stochastic mixed-integer programs. Mathematical Programming, Series B, 157:47–67.

[15] Ghamami, M., Zockaie, A., Nie, Y.M., 2016. A general corridor model for designing plug-in electric vehicle charging infrastructure to support intercity travel. Transportation Research Part C, 68, 389-402.

[16] Guan, Y., Ahmed, S., Nemhauser G. L., 2009. Cutting planes for multistage stochastic integer programs, Operations research, 57(2), 287–298.

[17] Heitsch, H., Römisch, W, 2009. Scenario tree modeling for multistage stochastic programs. Mathematical Programming, 118, 371–406.

[18] Hodgson, M. J., 1990. A Flow Capturing Location Allocation Model. Geographical Analysis, 22(3), 270-279.

[19] Hosseini, M., MirHassani, S. A., 2017. A heuristic algorithm for optimal location of flow-refueling capacitated stations. International Transactions in Operational Research, 24, 1377-1403.

[20] Hosseini, M., MirHassani, S., 2015. Refueling-station location problem under uncertainty. Transportation Research Part E, 84, 101-116.

[21] Hoyland, K., Wallace S.W., 2001. Generating Scenario Trees for Multistage Decision Problems. Management Science, 47(2), 295-307.

[22] Hu, S., Han, C., Dong, Z. S., Meng, L., 2019. A multi-stage stochastic programming model for relief distribution considering the state of road network. Transportation Research Part B, 123, 64-87.

[23] Huang, K., Ahmed, S., 2008. On a Multi-stage Stochastic Programming Model for Inventory Planning. INFOR, 46(3), 155–163.

[24] http://www.ieee-holm.org/h2017/Morton%20Antler%20IEEE%20Conference.%20Electrified%20and%20Autonomous%20(2)%20%20distributed.pdf

[25] International Energy Agency, 2009. Transport energy and CO2: moving toward sustainability. Technical report.

[26] International Energy Agency, 2016. Global EV Outlook 2016. Technical Report.

[27] Kim, J. G., Kuby, M., 2012. The deviation-flow refueling location model for optimizing a network of refueling stations. International journal of hydrogen energy, 37(6), 5406-5420.

[28] Kim, J. G., Kuby, M., 2013. A network transformation heuristic approach for the deviation flow refueling location model. Computers and Operations Research, 40(4), 1122-1131.

[29] Kuby, M., Lim, S., 2005. The flow-refueling location problem for alternative-fuel vehicles. Socio-Economic Planning Sciences, 39(2), 125-145.

[30] Lee, C., Han, J., 2017. Benders-and-price approach for electric vehicle charging station location problem under probabilistic travel range. Transportation Research Part B, 106, 130-152.

[31] Li, S., Huang, Y., Mason, S. J., 2016. A multi-period optimization model for the deployment of public electric vehicle charging stations on network. Transportation Research Part C, 65, 128-143.

[32] Lim, S., Kuby, M., 2010. Heuristic algorithms for siting alternative-fuel stations using the flow-refueling location model. European Journal of Operational Research, 204(1), 51-61.

[33] Lokketangen, A., Woodruff D.L., 1996. Progressive Hedging and Tabu Search Applied to Mixed Integer (0, 1) Multistage Stochastic Programming. Journal of Heuristics, 2, 111-128 (1996)

[34] Magnanti, T. L., Wong, R. T., 1981. Acceleration Benders Decomposition: Algorithmic Enhancement and Model Selection Criteria. Operations research, 29(3), 464-484.

[35] Miralinaghi, M., Lou, Y., Keskin, B.B., Zarrinmehr, A., Shabanpour, R., 2017. Refueling station location problem with traffic deviation considering route choice and demand uncertainty. International Journal of Hydrogen Energy, 42 (5), 3335-3351.

[36] MirHassani, S. A., Ebrazi, R. A., 2012. Flexible reformulation of the refueling station location problem. Transportation Science, 47(4), 617-628.

[37] Nickel, S., Saldanha-da-Gama, F., Ziegler, H., 2012. A multi-stage stochastic supply network design problem with financial decisions and risk management. Omega, 40(5), 511-524.

[38] Nowak, M. P., Römisch W., 2000. Stochastic lagrangian relaxation applied to power scheduling in a hydro-thermal system under uncertainty. Annals of Operations Research, 100(1-4):251–272.

[39] Owen, S. H., Daskin, M. S., 1998. Strategic facility location: A review. European journal of operational research, 111(3), 423-447.

[40] Pflug, G., 2001. Scenario tree generation for multiperiod financial optimization by optimal discretization. Mathematical Programming, Ser. B, 89,251–271.

[41] Pimentel, B. S., Mateus, G. R., Almeida, F. A., 2013. Stochastic capacity planning and dynamic network design. International Journal of Production Economics, 145, 139–149.

[42] Taghavi, M., Huang, K., 2016. A multi-stage stochastic programming approach for network capacity expansion with multiple sources of capacity. Naval Research Logistics, 63(8), 600-614.

[43] Taha, W., 2003. A gentle introduction to multi-stage programming. In: Lengauer C., Batory D., Consel C., Odersky M. (eds) Domain-Specific Program Generation. Lecture Notes in Computer Science, vol 3016. Springer, Berlin, Heidelberg.

[44] Tweed, K., 2013. Fast Charging Key to Electric Vehicle Adoption, Study Finds, https://www.greentechmedia.com/articles/read/fast-charging-key-to-electric-vehicle-adoption-study-finds, November, 17th 2013.

[45] Upchurch, C., Kuby, M., 2009. A Model for Location of Capacitated Alternative-Fuel Stations. Geographical Analysis, 41(1), 85-106.

[46] US Department of Energy, 2015. Costs associated with non-residential electric vehicle supply equipment. Technical report.

[47] Wang, Y. W., Lin, C. C., 2013. Locating multiple types of recharging stations for battery-powered electric vehicle transport. Transportation Research Part E, 58, 76-87.

[48] Wu, F., Sioshansi, R., 2017. A stochastic flow-capturing model to optimize the location of fast-charging stations with uncertain electric vehicle flows. Transportation Research Part D, 53, 354-376.

[49] Xie, F., Huang, Y., 2018. A multistage stochastic programming model for a multi-period strategic expansion of biofuel supply chain under evolving uncertainties. Transportation Research Part E, 111, 130-148.

[50] Yahyatabar, A., Najafi, A.A., 2018. Condition based maintenance policy for series-parallel systems through Proportional Hazards Model: A multi-stage stochastic programming approach. Computers & Industrial Engineering, vol. 126, 30–46.

[51] Yang, X., 2006. Improving portfolio efficiency: A genetic algorithm approach. Computational Economics, 28, 1-14.

[52] Yildiz, B., Arslan, B., Karaan, O. E., 2016. A branch and price approach for routing and refueling station location model. European Journal of Operational Research, 248(3), 815-826.

[53] Zeballos, L.J. ,Mendez, C.A., Barbosa-Povoa, A.P. , Novais, A.Q., 2014. Multi-period design and planning of closed-loop supply chains with uncertain supply and demand. Computers & Chemical Engineering, 66, 151-164.

[54] Zhang, K. C., Zhang, X. L., 2009. Using a genetic algorithm to solve a new multi-period stochastic optimization model. Journal of Computational and Applied Mathematics, 231, 114-123.

[55] Zhang, A., Kang, J., and Kwon C., 2017. Incorporating demand dynamics in multi-period capacitated fast-charging location planning for electric vehicles. Transportation Research Part B, 103, 5-29.

[56] Zhang, H., Ha, M., Zhao, H., Song, J, 2017. Inexact multistage stochastic chance constrained programming model for water resources management under uncertainties. Scientific Programming, 2017.

[57] Zou, J., Ahmed, S., Sun, X. A., 2019. Stochastic Dual Dynamic Integer Programming. Mathematical Programming, 175, 461-502.