TPLP **26** (1-2): 197–214, 2026. © The Author(s), 2026. Published by Cambridge University Press. This is an Open Access article, distributed under the terms of the Creative Commons Attribution licence (https://creativecommons.org/licenses/by/4.0/), which permits unrestricted re-use, distribution and reproduction, provided the original article is properly cited.

doi:10.1017/S1471068426100490 First published online 9 July 2026

*Long-Term Power Grid Planning via Answer Set Programming**

ANTONIO IELO and FRANCESCO DORIA
Dipartimento di Matematica e Informatica, Università della Calabria, Italy
(*e-mails:* antonio.ielo@unical.it, francesco.doria@unical.it)

SANDRA CASTELLANOS-PAEZ
Department of Computer Science, University of Huddersfield, UK
(*e-mail:* s.castellanos@hud.ac.uk)

MARCO MARATEA
Dipartimento di Matematica e Informatica, Università della Calabria, Italy
(*e-mail:* marco.maratea@unical.it)

FRANCESCO PERCASSI and MAURO VALLATI
Department of Computer Science, University of Huddersfield, UK
(*e-mails:* f.percassi@hud.ac.uk, m.vallati@hud.ac.uk)

submitted 26 May 2026 UTC; revised 26 May 2026 UTC; accepted 28 May 2026 UTC

Abstract

The power grid is a critical infrastructure underpinning all aspects of modern society and its services. Maintaining its effectiveness requires continuous adaptations. In particular, addressing sustainability targets, demand patterns, and urbanisation trends requires implementing changes to the network. Actual developments can potentially span over a decade, with supply continuity and service quality that must be preserved throughout by ensuring conformance to several topological and combinatorial invariants. Long-term power grid planning deals with the above process, and although planning languages could be a natural choice, the kind of properties and invariants needed are cumbersome to express in such languages; on the contrary, they can be elegantly and succinctly encoded in answer set programming (ASP). In this paper, we propose the first approach to automate and optimise the long-term power grid planning process using

* This work has been partially supported by the Italian "Ministero delle Imprese e Sviluppo Economico" (MIMIT) under projects "Ei-TWIN - Energy Digital Twin" (CUP B29J2400068000); by Regione Calabria under PR Calabria FESR FSE 2021–2027 through the projects "MOZART - Modelli e tecniche di mOdernizzazione di applicaZioni e data silos a supporto di clienti esterni, field force e impiegati di backoffice basati su AI GeneRativa e apprendimento auTomatico" (CUP J49I24001740005), and "SIGENERA - Large Language Models (LLM) per il controllo di SIstemi informativi, la GENERazione automatica di testo, e l'accesso a basi di conoscenzA" (CUP J29I24001770005). Francesco Percassi and Mauro Vallati were supported by a UKRI Future Leaders Fellowship, grant number MR/Z00005X/1.

https://doi.org/10.1017/S1471068426100490 Published online by Cambridge University Press

ASP. Experimental evaluations conducted on synthetic and real-world grid data confirm the expressive power of the proposed ASP-based approach and demonstrate its effectiveness.

KEYWORDS: answer set programming, application, energy management

1 Introduction

Power grids are the backbone of modern societies, and they are currently experiencing significant structural transformations. The increasing penetration of renewable energy sources (RES) at the distribution level, combined with the rapid uptake of electric vehicles, heat pumps, and the wider electrification of end-uses, is changing operational conditions and load patterns. As distribution networks evolve from passive infrastructures to active systems, their management becomes more complex and uncertain (Fang *et al.* 2012). In this context, long-term planning plays a central role. It refers to the process through which national grid operators define how the current distribution network should evolve over extended time horizons to meet future demand, integrate new technologies, and comply with national and regional energy transition scenarios (Conejo *et al.* 2010). More specifically, the long-term planning problem focuses on identifying the feasible multi-stage evolution path between the initial and the target desired power grid configuration. The two networks may differ significantly in topology, and many sequences of line additions, removals, or switching actions may connect them. However, considering the time span and the role played by the power grids, supply continuity and service quality must be preserved throughout the duration of the evolution (Khator and Leung 1997). Long-term planning decisions are characterised by their scale and long-lasting impact. For instance, the French distribution network spans approximately 1.4 million kilometres, and the cost of a single power line ranges from 50k€ to 200k€ per kilometre, depending on technology and location. Planning horizons typically extend up to 30 years, while infrastructure lifetimes may reach 60 years. As a result, even small inefficiencies in planning decisions can lead to significant economic consequences. At present, long-term power grid planning remains largely manual. National distribution system operators rely on human experts using power-flow and cost-analysis tools, but the construction of multi-stage evolution strategies is still mainly based on expert knowledge and iterative adjustments. This limits reproducibility, scalability, and the systematic exploration of alternative investment trajectories. In particular, the identification of feasible evolution strategies remains a major bottleneck, as it requires exploring a large combinatorial space of possible network transformations while ensuring that all intermediate configurations remain operational. Recent work has called for automated, intelligent decision-support tools that can complement human expertise and address the growing complexity of distribution networks (Castellanos-Paez *et al.* 2022, 2023). Artificial intelligence methods have a long tradition in power systems, ranging from expert systems for operator support (Wollenberg and Sakaguchi 1987) to planning-based approaches for sequential operational tasks (Bertoli *et al.* 2002; Bell *et al.* 2009; Thiébaux *et al.* 2013) to evolutionary and learning-based techniques (Arabali *et al.* 2012; Glavic *et al.* 2017; Vázquez-Canteli and Nagy 2019). Mixed Integer Linear Programming has been used for

https://doi.org/10.1017/S1471068426100490 Published online by Cambridge University Press

network reconfiguration problems focused on power and load flow constraints (Jabr *et al.* 2012; Ajaja and Galiana 2012), typically in static or short-horizon settings where the goal is to compute a feasible configuration rather than model transitions between configurations. Moreover, they are less adequate to express purely combinatorial properties, such as radiality (Ahmadi and Martí 2015), while being more suitable for flow-based constraints. Planning languages could be a natural choice, but graph properties and invariants (e.g., connectedness) are cumbersome to express and are better handled in knowledge representation and reasoning languages. Within this landscape, answer set programming offers a suitable approach, providing a declarative and expressive framework for modelling graph-theoretic constraints, discrete actions, and multi-stage feasibility requirements. Answer set programming (ASP; (Brewka *et al.* 2011)) is a declarative programming language based on stable model semantics (Gelfond and Lifschitz 1988) of logic programmes. Its expressive power and availability of efficient reasoners makes it a popular framework to model knowledge-intensive combinatorial problems. ASP has been successfully applied to academic and industrial problems alike (Erdem *et al.* 2016; Falkner *et al.* 2018).

This paper presents the first approach for automating long-term power grid planning that can handle all grid topological constraints. We propose an approach inspired by recent applications of ASP to *reconfiguration* problems (Yamada *et al.* 2023; Kato *et al.* 2024), where one has to find sequences of *edits* to navigate among solutions of (hard) combinatorial problems. We define an ASP encoding for the long-term power grid planning problem that allows for both sequential and parallel planning. Results of experiments, using the CLINGO ASP solver, executed on synthetic and real-world instances from the French grid, show that our solution is able to efficiently solve instances of a size comparable to those that experts deal with in daily operations when allowing a parallel execution of actions and to solve larger real-world instances within 30 minutes.

2 Background

This section introduces the required preliminary notions about distribution networks and the long-term power grid planning problem that will be useful to understand our solution.

2.1 Distribution network architecture

Urban electrical distribution networks are commonly designed according to a secured feeder architecture (see Figure 1). Primary substations act as the supply points of the medium-voltage distribution network, injecting power into the system through main feeders that interconnect the substations and supply multiple secondary substations. Along these feeders, secondary substations serve local areas and include distribution transformers that step down the medium-voltage level to low voltage for end users, as well as switching devices that control network operation. Each secondary substation is connected to a feeder through two switches, providing operational flexibility: normally closed switches (NCS) allow power flow under normal operating conditions, while normally open switches (NOS) are kept open to ensure radial operation so that each load is supplied by a single primary substation. In the event of a fault, the network can be reconfigured by operating the switches, enabling secondary substations to be resupplied from an alternative primary substation while repair actions are carried out.

https://doi.org/10.1017/S1471068426100490 Published online by Cambridge University Press

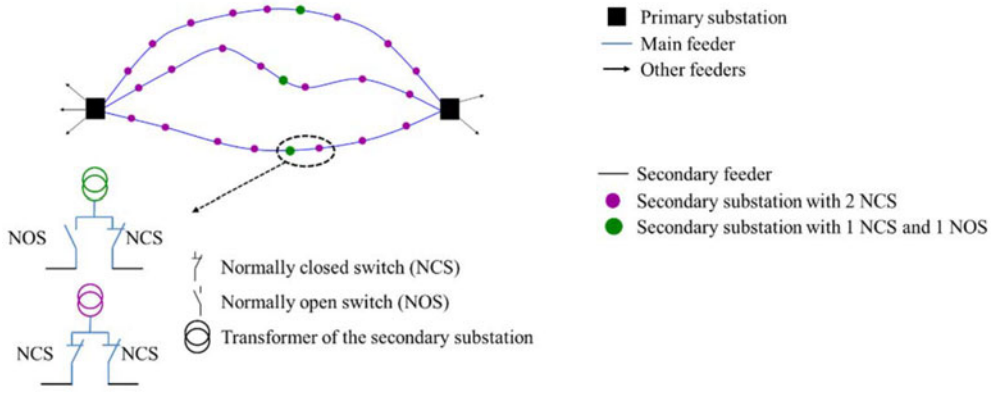


Fig. 1. Medium-voltage distribution network supplied by two primary substations (black squares) feeding interconnected feeders (blue lines) and secondary substations; purple circles represent secondary substations equipped with two normally closed switches (NCS), while green circles denote secondary substations equipped with one NCS and one normally open switch (NOS) used for radial operation and reconfiguration. In this work we treat both types of secondary substations uniformly and represent operational differences through the status of the network edges (open, close). (Figure adapted from Castellanos-Paez *et al.* (2023).).

2.2 Power grid planning

Distribution system operators (DSOs) are responsible for ensuring the safe, reliable, and cost-effective operation of power grids. This mandate includes maintaining existing infrastructures, guaranteeing continuity and quality of supply, and anticipating future developments in demand and distributed generation (Khator and Leung 1997; Conejo *et al.* 2010). In this context, long-term power grid planning is a core activity.

Long-term planning concerns the definition of how a distribution network should evolve over extended horizons, typically spanning multiple years. DSOs construct these plans considering multiple inputs, including consumption forecasts, expected production profiles – including the integration of renewable energy sources – and broader energy transition scenarios. The goal is to identify a future network architecture that remains technically feasible and operationally secure.

Infrastructure modifications in distribution grids are expensive and difficult to reverse. The cost of a single power line may range from 50 k€ to 200 k€ per kilometre, depending on technology (overhead vs. underground), cable type, and local conditions (rural, semi-urban, or urban). At the scale of a national distribution system – such as the French network, which extends over approximately 1.4 million kilometres – even minor changes can result in significant investments. Consequently, DSOs aim to minimise deviations from the existing infrastructure, favouring incremental and targeted interventions (Conejo *et al.* 2010).

A distinctive feature of long-term planning is that it must consider not only the final target network but also the sequence of intermediate networks that will emerge as investment actions are gradually executed over many years. Each intermediate configuration must remain compliant with operational and topological constraints: such intermediate configurations correspond to networks that will be in service for extended periods.

https://doi.org/10.1017/S1471068426100490 Published online by Cambridge University Press

Ensuring their feasibility is therefore critical to maintaining security of supply and service quality over time.

According to the ERDF methodology (ERDF 2008) for the development of distribution network master plans in France, the long-term planning process can be represented as a workflow composed of six main stages: system diagnosis, load forecasting, definition of a long-term target network, identification of development strategies, techno-economic scheduling of investments, and assessment of the resulting quality of supply. Among these stages, the identification of development strategies is a key step, as it determines how the transition from the current network to the target configuration is achieved. More informally, given the current network configuration and the target configuration, this step generates the list of interventions to be performed and the corresponding resulting intermediate networks. The present work focuses on this stage, which remains largely manual and based on expert knowledge in current practice.

To ensure the operationality of intermediate networks during the development strategies step and for minimising the potential for wasting resources in the long-term power grid planning process, a number of implicit rules are usually followed by grid operators. Castellanos-Paez *et al.* (2023) extracted the following set of rules via in-depth interviews with experts from French DSOs that are in charge of the development strategies:

- R1 There is no closed loop between two primary substations; that is, a normally open switch is required to ensure radial operation, meaning that each secondary substation in the network is supplied by exactly one primary substation.
- R2 The network must tolerate the loss of any primary node, ensuring that a radial configuration, in which all secondary nodes remain supplied, can always be restored via switching operations (opening and closing lines).
- R3 For urban areas, the number of connections of secondary substations are between 2 and 3.
- R4 Only lines belonging to the target network can be created, and only lines not belonging to the target network can be removed. These operations are irreversible, for example a line cannot be removed and later re-added.
- R5 There is only one connection between two nodes (distribution/source).

3 Problem formalisation

For our purposes, the power grid is an undirected graph with two types of nodes (primary, secondary) and two types of edges (open, close). One acts upon ("reconfigures") the power grid by changing its graph structure. The *long-term planning problem* (LPP) refers to computing a sequence of physical interventions on the power grid infrastructure that results in a specific *target* configuration in such a way that each intermediate state complies with the DSOs rules as described in Section 2.2. In the remainder of this paper, we use the term LPP to refer to the discrete combinatorial component of the long-term planning process. In this section, we abstract and formalise the problem and the DSOs requirements into graph-theoretic terms.

Notation. In the rest of the section, we adopt the following notation. Let P, S be two finite disjoint sets of *primary* and *secondary* stations, with $V = P \cup S$. A *power grid* is an edge-labelled undirected graph $G(V, E, \lambda)$, with $E \subseteq V \times V$ and $\lambda : E \mapsto \{open, close\}$.

https://doi.org/10.1017/S1471068426100490 Published online by Cambridge University Press

The labelling function λ marks closed and open lines in the power grid. We will refer to the graph $G(V, \{e \in E : \lambda(e) = close\})$ as the *closed network* and $G(V, E)$ as the *open network*. The representation in Figure 1 is consistent with this notation: secondary substations are not distinguished at the node level, and their operational differences are encoded through function λ.

3.1 Acting upon the power grid

There are three possible kinds of actions available to reconfigure the power grid: edge addition, edge removal and edge switching. Each action changes the power grid graph. Whenever applying action a over the power grid G results in G', we denote this by $G \vdash_a G'$. We describe effects of each possible action on G.

Edge addition. Edge addition consists of adding a new edge to the open network. Let $x, y \in V$ with $(x, y) \notin E$. We denote the action of adding (x, y) to G by $add(x, y)$. It results in the power grid $G(V, E', \lambda')$, where $E' = E \cup \{(x, y)\}$ and $\lambda'(e) = \lambda(e)$ for all $e \in E$, and $\lambda'(x, y) = open$.

Edge removal. Edge removal consists of removing an edge from the open network. Let $(x, y) \in E$ such that $\lambda(x, y) = open$. We denote the action of removing (x, y) as $remove(x, y)$. It results in the power grid $G(V, E', \lambda')$, where $E' = E \setminus \{(x, y)\}$ and $\lambda'(e) = \lambda(e)$ for all $e \in E'$.

Edge switching. Edge switching consists of swapping the state of two edges that are adjacent to the same secondary station. Let $x \in S$, $y, z \in V$ such that $\lambda(x, y) = close$ and $\lambda(x, z) = open$. We denote the switching action by $switch(x, y, z)$. It results in the power grid $G(V, E, \lambda')$, with $\lambda(e) = \lambda'(e)$ for all $e \in E \setminus \{(x, y), (x, z)\}$ and $\lambda'(x, y) = open$, $\lambda'(x, z) = close$.

Non-interfering actions. Executing an action causes a change in the power grid graph. Intuitively, concurrent execution of actions that involve the same edges leads to ambiguities during plans' execution. We formalise the notion of "allowed concurrency" in our domain, inspired by the notion of *non-interference* in planning (see, e.g., (Rintanen *et al.* 2006)). We denote by $\mathrm{Aff}(a)$ the set of edges that is involved in the execution of action a, and in particular we have that $\mathrm{Aff}(p(x, y)) = \{(x, y)\}$ if $p \in \{add, remove\}$ and $\mathrm{Aff}(switch(x, y, z)) = \{(x, y), (x, z)\}$.

We say two actions a_1 and a_2 are *non-interfering* (or *compatible*) if $\mathrm{Aff}(a_1)$ and $\mathrm{Aff}(a_2)$ are disjoint sets. Intuitively, non-interfering actions are "compatible", in the sense that they can be carried out on the graph "at the same time" without ambiguities. A set of actions A is non-interfering if all its elements are pairwise non-interfering. Thus, we extend the notion of affected edges to sets of actions, with $\mathrm{Aff}(A) = \bigcup_{a \in A} \mathrm{Aff}(a)$. We say that $G \vdash_A G'$ if we obtain G' by applying all the actions $a \in A$; if A is non-interfering, there is no ambiguity in the resulting graph G' (as no edge is affected by multiple actions).

3.2 Compliant networks

The DSOs' rules described in Section 2.2 provide properties that power grids must comply with to be considered valid. These can be succinctly expressed in terms of graph-theoretic

https://doi.org/10.1017/S1471068426100490 Published online by Cambridge University Press

properties. We say a power grid $G(P \cup S, E, \lambda)$ is compliant whenever the following properties are satisfied.

Definition 1 (Radial).
A power grid $G(P \cup S, E, \lambda)$ is radial *if its closed network is acyclic and, in this network, each secondary node is connected to exactly one primary node. That is, G is a forest with roots in P.*

Definition 2 (Redundantly connected).
Let $G(P \cup S, E, \lambda)$ be a power grid. Let $s \in S$ be a secondary station connected to $p \in P$ in the closed network. We say that s is redundantly connected *if, in the open network, there exists a path from s to some primary $p' \neq p$ whose internal nodes do not include primary nodes.*

Definition 3 (Reconfigurable).
Let $G(P \cup S, E, \lambda)$ be a power grid. G is reconfigurable *if all its secondary stations $s \in S$ are redundantly connected.*

Definition 4 (Degree-compliant).
A power grid $G(P \cup S, E, \lambda)$ is degree-compliant if it holds that $2 \leq deg(s) \leq 3$ for all $s \in S$.

3.3 Long-term planning problem

Finally, given an initial power grid $G(P \cup S, E, \lambda)$ and a target one $G_T(P \cup S, E_T, \lambda_T)$, we define the *long-term planning problem* as computing a sequence of sets of actions $(A_0, \ldots, A_n)$ such that $G_i \vdash_{A_i} G_{i+1}$, with $G_0 = G$, $G_n = G_T$, and each intermediate network G_i satisfying the radiality, reconfigurability and degree-compliant properties. We denote by $B = E_T \setminus E$ the set of *buildable* edges, that is edges present in the target network but absent from the initial one, and by $R = E \setminus E_T$ the set of *removable* edges, that is edges present in the initial network but not in the target.

Definition 5 (Long-term Planning Problem (LPP)).
Let $G(P \cup S, E, \lambda)$ and $G_T(P \cup S, E_T, \lambda_T)$ be two power grids. $LPP(G, G_T)$ is the problem of computing a sequence of compatible sets of actions $(A_0, \ldots, A_n)$ such that:

- *C1* $G_0 = G$, $G_n = G_T$;
- *C2* $\forall\, i \in \{0, \ldots, n-1\}$, $G_i \vdash_{A_i} G_{i+1}$, *with G_i being radial, reconfigurable, and degree-compliant;*
- *C3* *If* $add(x, y) \in A_i$ *then* $(x, y) \in B$;
- *C4* *If* $remove(x, y) \in A_i$ *then* $(x, y) \in R$;
- *C5* *No* (x, y) *such that* $add(x, y) \in A_i$ *and* $remove(x, y) \in A_j$ *for some* $0 \leq i, j \leq n$.

Relationship with DSOs rules. C1 requires that the network evolves from G to G_T by the end of the horizon. C2 requires every intermediate network G_i to satisfy radiality (R1), fault-tolerance under the loss of any primary node (R2), and the degree compliance (R3). R4 is captured by C3–C5, which constrain admissible line additions and removals and enforce their irreversibility.

https://doi.org/10.1017/S1471068426100490 Published online by Cambridge University Press

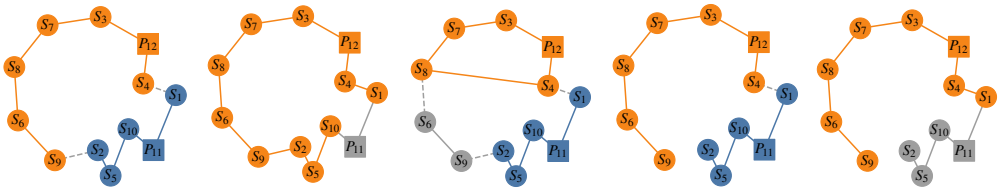


Fig. 2. Example graph configurations illustrating the role of constraints.

In the remainder, we refer to the *bounded* version of LPP whenever we assume an integer upper bound $k \in \mathbb{N}$ on the length of the solutions. We refer to the *sequential* version of the problem whenever we add the requirement that $|A_i| = 1$, and *parallel* otherwise (i.e.,, $|A_i| \geq 1$).

Example. Figure 2 illustrates the constraints considered in this work. Figure 2a shows a compliant configuration, where radiality, redundant connectivity, and degree constraints are all satisfied. Figure 2b shows the reconfigurability of the graph from Figure 2a under the failure of P_{11}; in particular, radiality can be restored by closing edges (S_2, S_9) and (S_1, S_4). Figure 2c shows a configuration that violates radiality for two distinct reasons: nodes S_6 and S_9 are isolated, and the feeder associated with P_{12} contains a cycle. Figure 2d shows a graph that is radial but not redundantly connected. Figure 2e shows how the violation of connectivity in the graph provided in 2d implies non-reconfigurability: in particular, in case of failure of P_{11}, node S_1 can be resupplied by closing (S_1, S_4), whereas S_2, S_5, and S_{10} remain isolated.

4 ASP Encoding

In this section, we present our answer set programming (ASP) solution designed to model the LPP problem. We assume the reader to be familiar with ASP and the input language of the CLINGO system, and refer them to standard introductory material (Gebser *et al.* 2012; Lifschitz 2019). We start with a high-level description of our approach.

The idea of the encoding is to model how the *starting network* G evolves subject to a sequence of sets of actions $(A_0, \ldots, A_k)$, with the goal of transforming it into a desired *target network* G_T within a given number of steps k. The approach is inspired by classical ASP applications in planning (Son *et al.* 2023) and bounded reconfiguration (Yamada *et al.* 2023; Kato *et al.* 2024) problems. We first present the data model, and then the proposed ASP encoding.

4.1 Data model

Let $G(P \cup S, E, \lambda)$ and $G_T(P \cup S, E_T, \lambda_T)$ be a pair of power grids. We describe the predicates and facts we use to represent inputs and outputs of the $LPP(G, G_T)$ problem.

Input. To describe the available stations in the power grid, we use atoms of the form $node(x)$ for $x \in P \cup S$. The atom $node_attr(x, primary)$ models that node x is a primary node, $x \in P$. Edges that belong to the start configuration are described by the predicate $start/3$. In particular, if $\lambda(x, y) = o$, we have the atom $start(x, y, o)$, with

https://doi.org/10.1017/S1471068426100490 Published online by Cambridge University Press

$o \in \{open, close\}$. Similarly, edges in the target configuration are described by the predicate $target/3$, with the atom $target(x, y, o)$ whenever $\lambda_T(x, y) = o$. We state that an edge (x, y) may be built or removed by using the atoms $buildable(x, y)$ and $must_remove(x, y)$. As input, we expect an *undirected* representation of the graph, and in particular we assume the fact $start(x, y, o)$ with $x < y$.

Output. A solution $(A_0, \ldots, A_k)$ to the $LPP(G, G_T)$ problem is described by means of the $action/2$ predicate, where the atom $action(t, a)$ models that $a \in A_t$. In particular, a will be function terms of the form $add/2$, $remove/2$ and $switch/3$, mirroring how possible actions are defined in Section 3.1. As an example, $action(3, switch(x, y, z))$ means that the solution includes the action $switch(x, y, z)$ at time 3, that is $switch(x, y, z) \in A_3$.

4.2 Encoding of the problem

We present the ASP encoding used to model the LPP problem. We describe it incrementally, with each paragraph assuming all the rules that precede it. We refer to the i-th line in the encoding by r_i. We present the encoding to compute a bounded solution of length at most k.

***Initial* definitions.** We start with definitions that make the encoding more readable. Let $G(P \cup S, E, \lambda)$ be a power grid. We denote which nodes of our network are respectively primary stations or secondary stations by the $primary/1$ and $secondary/1$ predicates, respectively. We will have the atom $primary(x)$ if $x \in P$, and $secondary(x)$ if $x \in S$. The $opposite/2$ predicate expresses the notion that open line and closed lines are opposite concepts. The $network/4$ predicate stores the "state", with $network(i, x, y, o)$ stating that $\lambda_i(x, y) = o$ in the i-th power grid. We assume $t = -1$ corresponds to the initial state of the network.[1] The $edge/4$ predicate is a "symmetric closure" of $network/4$ over time (that also keeps track of each edge's state) that will be useful to define the effects of the *switch* action (r_{13}–r_{14}).

```
1 primary(X) :- node_attr(X, is_primary).
2 secondary(X) :- node(X), not primary(X).
3 opposite(open,close). opposite(close,open).
4 network(-1,X,Y,A) :- start(X,Y,A).
5 edge(T, X, Y, A) :- network(T, X, Y, A).
6 edge(T, X, Y, A) :- network(T, Y, X, A).
```

Generating action sequences. The following rules generate the search space of possible action sequences. Informally, we first guess "the plan's length" (r_8) by guessing a $time/1$ atom (and propagating backwards existence of previous time-points, up to zero (r_9):

```
7 #const k. step(0..k-1). time(0).
8 { time(T) } :- step(T).
9 time(T-1) :- time(T), T > 0.
```

[1] The encoding assumes actions at time t get applied at time t, and depend on time $t-1$. This is a minor implementation detail, just an index shift wrt the definition of a solution $(A_0, \ldots, A_k)$ of the LPP problem.

https://doi.org/10.1017/S1471068426100490 Published online by Cambridge University Press

Next, the following choice rules impose that edges to be built and removed will be subject to an *add* (r_{12}) and *remove* action (r_{10}), respectively, exactly once within the whole solution. We prohibit the removal of closed edges (r_{11}). The third choice rule (r_{13}–r_{14}) guesses edge switching operations. The last constraint r_{15} imposes that we perform at least one action in each time-point.

```
{ action(T,remove(X,Y)): time(T) } = 1 :- must_remove(X,Y).
:- action(T, remove(X,Y)), network(T-1,X,Y,close).
{ action(T,add(X,Y)): time(T) } = 1 :- buildable(X,Y).

{ action(T,switch(X,Y,Z)): edge(T-1, X, Y, O), edge(T-1, X, Z, O'),
  opposite(O,O'), secondary(X) } :- time(T).
:- time(T), not action(T,_).
```

The atom $action(t, a)$ encodes that at time t the action a is performed, where the action a is a function term as described in Section 4.1. We assume that actions taken at time t act upon the network as available in the previous time-step $t-1$, and take effect at time t.

The $affected_by/4$ predicate (r_{16}–r_{21}) marks networks' edges that are affected by an action, according to the definition of $\mathrm{Aff}(\cdot)$. In particular, the atom $affected_by(t, x, y, a)$ states that $(x, y) \in \mathrm{Aff}(a)$, and that a is performed at time t. Affected edges will either change their state (i.e., whether they are open or close), or be added/removed at the next time-point.

```
affected_by(T,X,Y,add(X,Y)) :- action(T,add(X,Y)).
affected_by(T,X,Y,remove(X,Y)) :- action(T,remove(X,Y)).
affected_by(T,X,Y,switch(X,Y,Z)) :- action(T,switch(X,Y,Z)).
affected_by(T,Y,X,switch(X,Y,Z)) :- action(T,switch(X,Y,Z)).
affected_by(T,X,Z,switch(X,Y,Z)) :- action(T,switch(X,Y,Z)).
affected_by(T,Z,X,switch(X,Y,Z)) :- action(T,switch(X,Y,Z)).
```

Notice that (see r_{10}–r_{12}) we can add an edge (x, y) only if it we have the $buildable(x, y)$ fact, and similarly we can remove it only if we have the $must_remove(x, y)$ fact. Thus, for add and remove actions, we know precisely the affected edge in the ASP representation. On the other hand, the available switch actions are defined on the previous network state (r_{13}), and an edge may be affected by switch actions from different endpoints. As an example, edge (S_6, S_9) in Figure 2 may be affected by $switch(S_6, S_8, S_9)$ but also $switch(S_9, S_6, S_2)$. Thus, the definition of $affected_by/4$ takes this into account (r_{18}–r_{21}). To ensure our solutions consist of compatible action sets, we forbid to guess non-compatible actions (i.e., actions affecting the same edge).

```
:- affected_by(T,X,Y,A), affected_by(T,X,Y,A'), A < A'.
```

Actions' effect. The following rules apply *the effect* of actions taken at time t on the network at time $t-1$, according to the formalisation of Section 3.1. They share the use of atoms $network(t-1, _, _, _)$ and $action(t, _)$ – past network state and action to execute – in the body, and have $network(t, _, _, _)$ – how the network is affected by the action.

https://doi.org/10.1017/S1471068426100490 Published online by Cambridge University Press

Actions of the form $add(x, y)$ simply define that the edge (x, y) will exist at the next time-step:

```
network(T,X,Y,open) :- time(T), action(T,add(X,Y)).
```

The edge switching $switch(x, y, z)$ operation first identifies the correct pair of edges that is involved in the action, then flips their state in the next time-step:

```
network(T,X,Y,O') :- affected_by(T,X,Y,switch(_,_,_)), network(T-1,X,Y,O),
    opposite(O,O').
```

All edges that are not affected by an action are implicitly carried over to the next time-step, with the same state as before. This is a form of *inertia*. The combination of $remove(x, y)$ affecting (x, y) and intertia implicitly removes edges.

```
network(T,X,Y,A) :- time(T), network(T-1,X,Y,A), not affected_by(T,X,Y,_).
```

Encoding radiality I (Connectedness). To enforce the radiality property, as by Definition 1, we require the closed network to be a forest with roots in the primary nodes P. First, we ensure that (i) all secondary $s \in S$ reach at least one primary $p \in P$ in the closed network, and (ii) there are no loops between primaries over the closed network. We start by defining the standard notion of graph reachability over the closed network by means of the predicate $reach/3$ (r_{27}–r_{30}):

```
reach(T,X,Y) :- time(T), network(T,X, Y,close).
reach(T,X,Y) :- time(T), network(T,Y, X,close).
reach(T,X,Y) :- time(T), reach(T,Y,X).
reach(T,X,Z) :- time(T), reach(T,X,Y), reach(T,Y,Z).
```

Then, we use the atom $reaches_primary(t, x, y)$ to model that node x is connected to the primary station y at time t. The constraints model, respectively, that two distinct primaries should not reach each other (r_{32}), and that all secondary stations should be connected to at least one primary station (r_{33}).

```
reaches_primary(T,X,Y) :- time(T), reach(T,X,Y), primary(Y).
:- primary(X), primary(Y), time(T), X < Y, reach(T,X,Y).
:- secondary(Y), time(T), not reaches_primary(T,Y,_).
```

Encoding radiality II (Acyclicity). Since the problem deals with undirected graphs and the previous rules (r_{32}–r_{33}) enforce that each secondary station is connected to exactly one primary station in the closed network (recall that if it were connected to two there would be a loop between primaries, which we forbid by means of a constraint (r_{32}), it is sufficient to enforce that the network at time t has *the correct number of edges* to enforce acyclicity. Since we are looking for $|P|$ disjoint trees and the number of edges in a tree of q nodes is $q - 1$, we want our network to have exactly $|V| - |P|$ edges. The predicates $num_primaries/1$ and $num_nodes/1$ store the number of primary nodes and overall number of nodes in the network, while the constraint (r_{36}–r_{37}) enforces the correct number of overall edges.

https://doi.org/10.1017/S1471068426100490 Published online by Cambridge University Press

```
num_primaries(R) :- R=#count{Z: primary(Z)}.
num_nodes(X) :- X=#count{Z: node(Z)}.
:- num_primaries(R), num_nodes(Q), time(T),
    #count{X, Y: network(T,X,Y,close)} != (Q-R).
```

Encoding *reconfigurability.* Secondary stations must be *redundantly connected* (Definition 2) to an extra primary substation over the open network, distinct from the one they are connected to in the closed network. Recall that a secondary station cannot be connected to two primary stations in the closed network, as this would create a loop between primaries and violate radiality.

```
redundancy_reach(T,X,Y) :- time(T), network(T,X,Y,_).
redundancy_reach(T,X,Y) :- time(T), redundancy_reach(T,Y,X).
redundancy_reach(T,X,Z) :- time(T), redundancy_reach(T,X,Y),
    redundancy_reach(T,Y,Z), not primary(Y).

redundancy(T,S) :- secondary(S), time(T), redundancy_reach(T, S, X),
    primary(X), not reach(T, S, X).
:- secondary(S), time(T), not redundancy(T,S).
```

The predicate *redundancy_reach*/3 models reachability over the open network (r_{38}–r_{41}), with the atom *redundancy_reach*(t, x, y) modelling that node x is connected to node y over the open network *with a path that does not traverse any primary substation* – notice the `not primary(Y)` atom in the body of the recursive definition of *redudancy_reach*/3 (r_{41}). This ensures the only way networks' edges involving primaries are taken into account in computing *redundancy_reach*/3 is via direct edges. Finally, *redundancy*(t, s) denotes that at time t the secondary station s reaches a primary substation over the open network, and we prevent a secondary station that does not (r_{44}), ensuring reconfigurability as by Definition 3, that is that all secondary nodes are redundantly connected.

Encoding degree-compliance. We use aggregates to count the number of adjacent edges to each node, with the atom *degree*(t, x, d) meaning that node x has degree d in the network at time t. The constraints (r_{46}–r_{47}) enforce that $d \in [2, 3]$, as by Definition 4:

```
degree(T,X,D) :- time(T), node(X), D=#count{Y: edge(T,X,Y,_)}.
:- secondary(S), time(T), degree(T,S,D), D < 2.
:- secondary(S), time(T), degree(T,S,D), D > 3.
```

Final configuration. We encode that the final network obtained by applying the guessed actions sequence is exactly the target network G_T, as in Definition 5: it contains the same edges as G_T and in the same state (r_{49}–r_{50}), and all edges to be removed have been removed (r_{51}).

```
final(T) :- time(T), T = #max{ T' : time(T') }.
:- final(T), target(X,Y,A), not network(T,X,Y,A).
:- final(T), network(T,X,Y,A), not target(X,Y,A).
:- final(T), must_remove(X,Y), network(T,X,Y,_).
```

Preferred solutions. Answer sets of the above logic programme encode sequences of (sets of) actions that reconfigure G into G_T. As customary in ASP applications, multiple

https://doi.org/10.1017/S1471068426100490 Published online by Cambridge University Press

answer sets may exist, that is multiple ways to reconfigure G into G_T. ASP optimisation (via weak constraints) enables us to single out *preferred solutions*, according to different criteria. For the purposes of this paper, we are interested in reducing the overall number of actions executed in the plan, as well as its length (lexicographically). We can express this by the constraints:

```
52 :∼ action(T,switch(X,Y,Z)). [1@2,T,A]
53 :∼ time(T). [1@1,T]
```

The weak constraint r_{52} minimises the total number of actions,[2] while the weak constraint r_{53} favours shorter plans.

5 Experimental analysis

The empirical evaluation aims to assess the capabilities of the proposed approach. We start by describing the experimental settings and, then, in Section 5.1, we present the results.

Execution environment. Experiments were executed on a 64-bit Linux machine equipped with an Intel i7-13700 @ 5.2 GHz with 32 GB of RAM and 16 physical cores, running up to 8 concurrent jobs via GNU Parallel. Each run was limited to a timeout of 1800 CPU seconds and 2 GB of memory. We use the CLINGO solver (version 5.4.1) in default configuration.

Implementation details. In real scenarios, a bound on reconfiguration's length is unknown. We progressively expand such bound by means of multiple solver calls. In particular, we solve the bounded version of the problem over the interval $[2^h, 2^{h+1}]$ (i.e., we search for a plan of length at least 2^h and at most 2^{h+1}) starting with $h = 0$, and increasing it up to the point where we find a solution (or hit resource limits). To enforce a lower bound 2^h on plan's length, we add the fact $time(2^h)$, while to enforce an upper bound 2^{h+1} we set the runtime constant k to 2^{h+1}.

Data. Our test set is composed by a collection of synthetic and real-world instances. The 7 real instances were obtained using data from a French urban network consisting of two source stations (primary nodes). This data has been anonymised and used to create smaller subnetworks that comply with a secure feeder architecture and correspond to French distribution network design practices in terms of topology, feeder capacity constraints, and balancing criteria. The instances range in size from 6 to 40 secondary nodes, and all have 2 primary nodes. Further, we generated synthetic instances by varying the number of nodes $|V| \in \{8, 12, 15, 18, 22, 30, 40, 50\}$, fixing the number of primary nodes to $|P| = 2$. This size matches what experts deal with in daily operations. For each size, we sampled five random compliant power grids. Starting from each grid G, we constructed a sequential plan by applying valid random actions that preserve compliance, leading to G_T. In doing so, we avoid trivial redundancies, that is addition-removals of the same edge, or "opposite switches" in adjacent time-steps. Motivated by the empirical observation

[2] Edges cannot be added (removed) twice, hence minimising actions is equivalent to minimising the number of switches.

https://doi.org/10.1017/S1471068426100490 Published online by Cambridge University Press

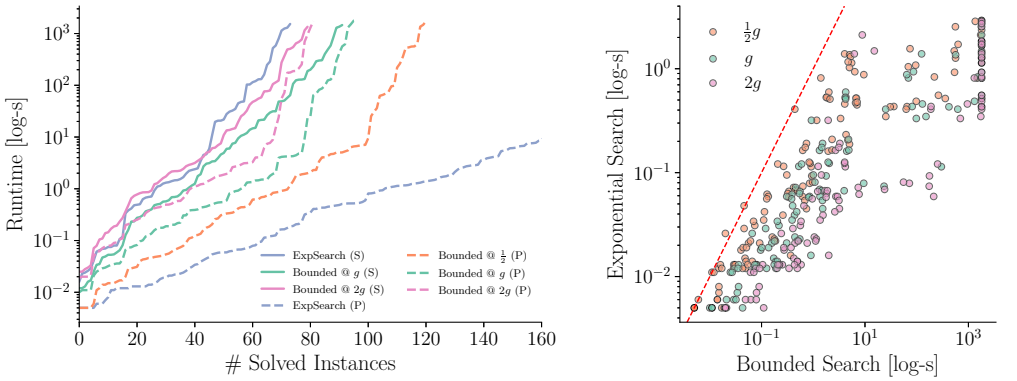


Fig. 3. (a) A point (i, t) indicates that i instances are solved in t seconds. (b) A point (x, y) denotes a synthetic instance solved in y seconds by exponential and in x by bounded. Points below the bisector indicate faster exponential search.

that, in real instances, solution length is approximately equal to the number of edges, we considered plan depths of the form $\alpha \cdot |E|$ with $\alpha \in \{0.2, 0.6, 1.0, 1.4, 1.8\}$. Executing these action sequences yields pairs of compliant configurations (G, G_T), which constitute valid instances of the LPP. Overall, this process generates 175 synthetic LPP instances. Moreover, $\alpha \cdot |E|$ is an upper bound for a sequential solution's length. We will refer to this (instance-wise) upper bound as g, and experiments will consider $\frac{1}{2}g$, g and $2g$ as possible bounds.

5.1 Results

Below we present the results of our experiments. All solutions have been validated using an external purposely-designed tool to assess the satisfaction of the DSO rules.

Synthetic instances. In order to select the best approaches to be employed on real data, we first test our encoding in the bounded and exponential search version, searching for sequential and parallel plans, on the synthetic instances. We report our results in terms of the cactus plot and scatterplot in Figure 3. First, we can observe in Figure 3a that we are able to solve more problem instances dealing with parallel plans wrt sequential plans. Intuitively, this is due to non-interfering actions "compressing" the solutions' length, which translates to dealing with smaller logic programmes. In ASP planning, this typically correlates with *easier* problems, given that long plans are a known pain point also for ASP-based planning (see, e.g., (Son *et al.* 2023)). Exponential search performs worse than bounded search over sequential plans, as it incurs in overhead due to multiple solver calls that are required to expand the search horizon, while the bounded encoding operates directly on it. In the parallel setting, the exponential search approach dominates, solving 54 more instances wrt the bounded at horizon $\frac{1}{2}g$, 78 wrt g and 93 wrt $2g$, with all instances being mostly trivial for the parallel approach. The scatterplot in Figure 3b shows that for parallel plans, all instances slow down as bounds increase. In particular, we can observe (top-right corner of the scatterplot) that these synthetic instances – despite being trivial whenever they admit a short solution – are challenging if we try to solve at

https://doi.org/10.1017/S1471068426100490 Published online by Cambridge University Press

Table 1. *Number of solved instances by graph size and method. Cell content x; y denotes that we solve x instances with sequential planning, and y instances with parallel planning, respectively*

$\|V\|$	Exp. search	$\frac{1}{2}g$	g	$2g$	Total
8	25;25	0;25	25;25	25;25	75;100
12	21;25	0;25	25;25	25;25	71;100
18	11;25	0;25	19;21	16;16	46;87
22	7;25	0;16	13;11	9;11	29;63
30	5;25	0;15	5;10	5;5	15;55
40	5;25	0;10	5;5	0;0	10;40
50	0;25	0;5	0;0	0;0	0;30
Total	74;175	0;121	92;97	80;82	246;475

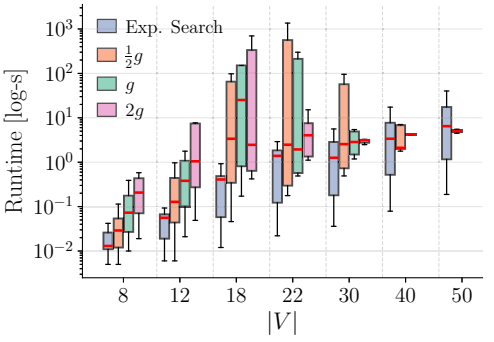


Fig. 4. Parallel plans runtime comparison across graph sizes. Red bar is the median runtime.

a much higher bound: we observe several instances that can be solved within seconds by the exponential search approach but timeout with larger bounds. Table 1 shows that all techniques solve *fewer* instances as $|V|$ increases. Parallel exponential search scales better as $|V|$ increases. This is confirmed by runtimes distributions in Figure 4. The largest instances are comparable in size to those handled by human experts, both for training and for the actual long-term planning of power grids. Larger grids are generally decomposed into networks of between 20-25 nodes. In that sense, our approach can readily support the work of experts. Overall, these results on synthetic instances motivate our focusing on parallel plans and exponential search approaches to solve real-world scenarios.

Real instances. Table 2 reports runtime of our exponential search approach, with and without optimisation statements (as discussed in Section 4). We are able to solve instances of up to 42 nodes, proving the approach is useful for operation support in realistic scenarios, and can solve instances of size larger than those experts deal in daily operations. Runtimes are comparable with the ones we obtained over synthetic instances of the same size. On some solved instances, we are also able to prove optimality of parallel plans. In real-world scenarios, this translate in lower costs and simpler-to-put-in-place operations. These solutions were validated by an automated tool and visually reviewed by

https://doi.org/10.1017/S1471068426100490 Published online by Cambridge University Press

Table 2. *Runtime (r) on real instances of a given size (# Nodes), with total number of actions ($\Sigma|A_i|$) and maximum number of concurrent actions ($\max|A_i|$). Dashes (–) denote timeout. The "Opt?" column marks instances where we are able to prove optimality of the parallel plan*

		Parallel			Parallel optimisation			
Instance	# Nodes	r	$\Sigma\|A_i\|$	$\max\|A_i\|$	r	$\Sigma\|A_i\|$	$\max\|A_i\|$	Opt?
P06	8	0.0	12	5	0.0	7	4	✓
P10	12	0.0	20	6	0.1	20	6	✓
P13	12	0.0	19	5	0.1	16	7	✓
P16	18	1.0	30	7	0.9	30	7	–
P22	24	0.4	40	12	1.8	29	10	✓
P30	30	41.2	67	10	62.2	65	11	–
P40	42	–	–	–	1485.7	103	14	–

domain experts, who confirmed their consistency with DSO planning rules and their comparability, at the topological level, to solutions typically produced by experts.

6 Conclusion

The problem of long-term power grid planning requires computing sequences of operations on the power grid infrastructure to *transform* it into a target configuration, while guaranteeing that each intermediate state satisfies certain properties of interest. Enacting these plans may span many years of costly operations. Currently, such plans are devised manually by domain experts. This paper proposes an ASP-based approach to tackle the problem, providing an automated solution inspired by ASP planning and ASP reconfiguration. Our experiments on synthetic and real-world data show that the approach can solve complex instances comparable to, or larger than, those currently handled by human operators. As future work, we plan to further improve scalability by applying, for example domain-specific heuristics and decomposition methods and adapt to infrastructure-specific requirements that may be present in other installations, such as immutability of certain components or budget limitations.

References

Ahmadi, H. and Martí, J. R. 2015. Mathematical representation of radiality constraint in distribution system reconfiguration problem. *International Journal of Electrical Power & Energy Systems* 64, 293–299.

Ajaja, A. and Galiana, F. D. 2012. Distribution network reconfiguration for loss reduction using MILP. In *IEEE PES Innovative Smart Grid Technologies Conference, ISGT 2012, Washington, DC, USA, January 16-20, 2012*, IEEE, Washington, DC, USA, 1–6.

Arabali, A., Ghofrani, M., Etezadi-Amoli, M., Fadali, M. S. and Baghzouz, Y. 2012. Genetic-algorithm-based optimization approach for energy management. *IEEE Transactions on Power Delivery* 28, 1, 162–170.

Bell, K., Coles, A., Coles, A., Fox, M. and Long, D. 2009. The role of AI planning as a decision support tool in power substation management. *AI Communications* 22, 1, 37–57.

https://doi.org/10.1017/S1471068426100490 Published online by Cambridge University Press

Bertoli, P., Cimatti, A., Slaney, J. K. and Thiébaux, S. 2002. Solving power supply restoration problems with planning via symbolic model checking. In *Proceedings of the 15th European Conference on Artificial Intelligence, ECAI'2002, Lyon, France, July 2002*, F. van Harmelen Ed., 576–580.

Brewka, G., Eiter, T. and Truszczynski, M. 2011. Answer set programming at a glance. *Communications of the ACM* 54, 12, 92–103.

Castellanos-Paez, S., Alvarez-Herault, M.-C. and Lalanda, P. 2022. Automated planning to evolve smart grids with renewable energies. In *Artificial Intelligence for Knowledge Management, Energy, and Sustainability*, Springer, 141–155.

Castellanos-Paez, S., Alvarez-Herault, M. C. and Lalanda, P. 2023. Decision support tool for the development of power distribution networks based on AI planning. In CIRED 12–15 June, 2023, 1425–1429. IET Conference, Rome, Italy. https://ieeexplore.ieee.org/document/10267542

Conejo, A. J., Carrion, M. and Morales, J. M. 2010. *Decision Making Under Uncertainty in Electricity Markets*. Springer.

Erdem, E., Gelfond, M. and Leone, N. 2016. Applications of answer set programming. *AI Magazine* 37, 3, 53–68.

ERDF. 2008. Description physique du réseau public. Accessed: 2026-01-30.

Falkner, A. A., Friedrich, G., Schekotihin, K., Taupe, R. and Teppan, E. C. 2018. Industrial applications of answer set programming. *KI - Künstliche Intelligenz* 32, 165–176.

Fang, X., Misra, S., Xue, G. and Yang, D. 2012. Smart grid — the new and improved power grid: A survey. *IEEE Communications Surveys & Tutorials* 14, 4, 944–980.

Gebser, M., Kaminski, R., Kaufmann, B. and Schaub, T. 2012. *Answer Set Solving in Practice*. Synthesis Lectures on Artificial Intelligence and Machine Learning. Morgan & Claypool Publishers.

Gelfond, M. and Lifschitz, V. 1988. The stable model semantics for logic programming. In *Logic Programming, Proceedings of the Fifth International Conference and Symposium Seattle, Washington, USA, August 15–19, 1988*, R. A. Kowalski and K. A. Bowen Eds., MIT Press, 1070–1080. https://dblp.org/rec/conf/iclp/GelfondL88.bib.

Glavic, M., Fonteneau, R. and Ernst, D. 2017. Reinforcement learning for electric power system decision and control: Past considerations and perspectives. *IFAC-PapersOnLine* 50, 1, 6918–6927.

Jabr, R. A., Singh, R. and Pal, B. C. 2012. Minimum loss network reconfiguration using mixed-integer convex programming. *IEEE Transactions on Power Systems* 27, 2, 1106–1115.

Kato, M., Banbara, M., Schaub, T., Soh, T. and Tamura, N. 2024. Dominating set reconfiguration with answer set programming. *Theory and Practice of Logic Programming* 24, 4, 755–771.

Khator, S. K. and Leung, L. C. 1997. Power distribution planning: A review of models and issues. *IEEE Transactions on Power Systems* 12, 3, 1151–1159.

Lifschitz, V. 2019. *Answer Set Programming*. Springer.

Rintanen, J., Heljanko, K. and Niemelä, I. 2006. Planning as satisfiability: parallel plans and algorithms for plan search. *Artificial Intelligence* 170, 1031–1080.

Son, T. C., Pontelli, E., Balduccini, M. and Schaub, T. 2023. Answer set planning: A survey. *Theory and Practice of Logic Programming* 23, 1, 226–298.

Thiébaux, S., Coffrin, C., Hijazi, H. and Slaney, J. 2013. Planning with MIP for supply restoration in power distribution systems. In *Proceedings of the 23rd International Joint Conference on Artificial Intelligence, Beijing, China, August 3–9, 2013*, F. Rossi Ed., IJCAI/AAAI, 2900–2907. http://www.aaai.org/ocs/index.php/IJCAI/IJCAI13/paper/view/6863.

https://doi.org/10.1017/S1471068426100490 Published online by Cambridge University Press

Vázquez-Canteli, J. R. and Nagy, Z. 2019. Reinforcement learning for demand response: A review of algorithms and modeling techniques. *Applied Energy* 235, 1072–1089.

Wollenberg, B. F. and Sakaguchi, T. 1987. Artificial intelligence in power system operations. *Proceedings of the IEEE* 75, 12, 1678–1685.

Yamada, Y., Banbara, M., Inoue, K. and Schaub, T. R. 2023. Bounded combinatorial reconfiguration with answer set programming. In *Proceedings of Logics in Artificial Intelligence - 18th European Conference, (JELIA) 2023, Dresden, Germany, September 20–22*. S. A. Gaggl and M. V. Martinez and M. Ortiz, Eds., 278–286. https://doi.org/10.1007/978-3-031-43619-2_20.

https://doi.org/10.1017/S1471068426100490 Published online by Cambridge University Press