Beyond Hosting Capacity: Using Shortest Path Methods to Minimize Upgrade Cost Pathways

Preprint

Nicolas Gensollen, Kelsey Horowitz,
Brian Palmintier, Fei Ding, and Barry Mather
National Renewable Energy Laboratory

To be presented at the 2018 World Conference on Photovoltaic Energy Conversion (WCPEC-7)
Waikoloa, Hawaii
June 10–15, 2018

NREL is a national laboratory of the U.S. Department of Energy
Office of Energy Efficiency & Renewable Energy
Operated by the Alliance for Sustainable Energy, LLC

This report is available at no cost from the National Renewable Energy Laboratory (NREL) at www.nrel.gov/publications.

Conference Paper
NREL/CP-5D00-71565
May 2018

Contract No. DE-AC36-08GO28308

NOTICE

This report is available at no cost from the National Renewable Energy Laboratory (NREL) at www.nrel.gov/publications.

Available electronically at SciTech Connect http:/www.osti.gov/scitech

Available for a processing fee to U.S. Department of Energy and its contractors, in paper, from:

U.S. Department of Energy
Office of Scientific and Technical Information
P.O. Box 62
Oak Ridge, TN 37831-0062
OSTI http://www.osti.gov
Phone: 865.576.8401
Fax: 865.576.5728
Email: reports@osti.gov

Available for sale to the public, in paper, from:

U.S. Department of Commerce
National Technical Information Service
5301 Shawnee Road
Alexandria, VA 22312
NTIS http://www.ntis.gov
Phone: 800.553.6847 or 703.605.6000
Fax: 703.605.6900
Email: orders@ntis.gov

Cover Photos by Dennis Schroeder: (left to right) NREL 26173, NREL 18302, NREL 19758, NREL 29642, NREL 19795.

NREL prints on paper that contains recycled content.

Beyond Hosting Capacity: Using Shortest Path Methods to Minimize Upgrade Cost Pathways

Nicolas Gensollen, Kelsey Horowitz, Bryan Palmintier, Fei Ding, and Barry Mather

National Renewable Energy Laboratory, Golden, CO, 80401, USA

***Abstract*—This paper presents a graph based forwardlooking algorithm applied to distribution planning in the context of distributed PV penetration. We study the target hosting capacity (THC) problem where the objective is to find the cheapest sequence of system upgrades to reach a predefined hosting capacity target value. We show that commonly used short-term cost minimization approaches often lead to suboptimal long-term solutions. By comparing our method against such myopic techniques on real distribution systems, we show that our algorithm is able to reduce the overall integration costs by looking at future decisions. Because hosting capacity is hard to compute, this problem requires efficient methods to search the space. We demonstrate that heuristics using domain specific knowledge can be efficiently used to improve the algorithm performance such that real distribution systems can be studied.**

***Index Terms*—hosting capacity, optimization, decisions, shortest path.**

I. Introduction

The deployment of distributed photovoltaic systems (DPV) in the United States has been rapidly increasing over the last decade, and this trend is likely to continue. This growth raises some challenges since distribution systems were not initially conceived to host large amounts of distributed generation. Various studies have reported problematic consequences of high penetration on certain distribution systems, including over-voltages, line thermal limit violations, or reverse power flows [1] [2]. When these issues are anticipated to arise in response to the interconnection of a specific DPV system, utilities mitigate them using a variety of possible solutions, ranging from modifying voltage regulating device set points to the physical addition of new components (e.g. transformers, voltage regulators, substation load tap changer (LTC), capacitor banks, new distribution lines) [3] [4].

In the US, this problem is typically tackled with a short-term cost minimization approach. When the penetration on a circuit reaches the point where grid operating violations start to occur, the cheapest system upgrade that resolves the issue is implemented. Being more proactive in choosing the system upgrades may substantially reduce the overall integration costs, although this practice remains rare in the US. We propose in this paper a graph based forward-looking method to address this problem in the context of hosting capacity (HC) [5], which provides an estimate of how much PV can be added to a given system without violating operational limits.

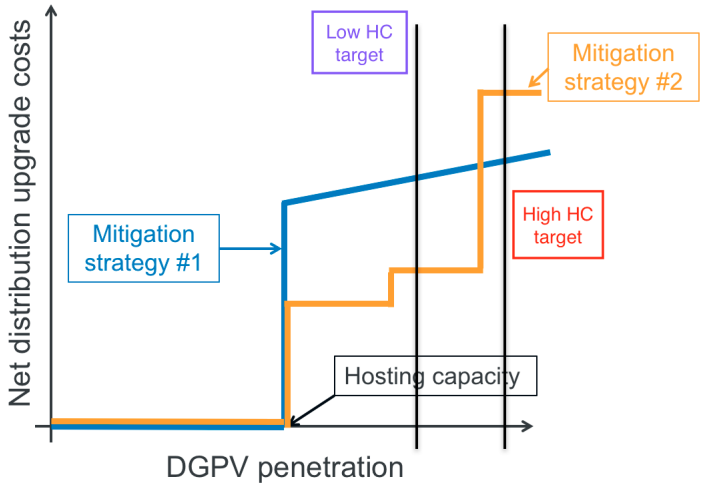

Fig. 1. Target Hosting Capacity (THC) Problem. The two vertical lines represent two possible HC targets, and the two colored lines display two possible sequences of actions. Applying the cheapest action first (orange line) may be suboptimal depending on the target. Here, for the higher target THC, applying the blue sequence provides a better long-term option, despite a larger up-front investment.

More precisely, we study the target hosting capacity (THC) problem where a system planner has to decide which sequence of system upgrades to implement in order to reach a target value of hosting capacity (Fig. 1). This problem arises when utilities want to achieve a certain penetration of DPV on a feeder such as to defer transmission upgrades by offsetting load, or to achieve a renewable energy target. We focus on the costs of these upgrades such that we wish to find the cheapest sequence of upgrades that would increase the HC value to a given predefined target.

We show that commonly used myopic greedy approaches can lead to suboptimal long-term solutions, sometimes very far from the global optimum. Taking into account future decisions can greatly lower the overall integration cost. A key challenge of this work is that the penetration of DPV is unknown beforehand, both temporally and spatially, such

This work was authored by Alliance for Sustainable Energy, LLC, the Manager and Operator of the National Renewable Energy Laboratory for the U.S. Department of Energy (DOE) under Contract No. DE-AC36-08GO28308. Funding provided by U.S. Department of Energy Office of Energy Efficiency and Renewable Energy Solar Energy Technologies Office. The views expressed in the article do not necessarily represent the views of the DOE or the U.S. Government. The U.S. Government retains and the publisher, by accepting the article for publication, acknowledges that the U.S. Government retains a nonexclusive, paid-up, irrevocable, worldwide license to publish or reproduce the published form of this work, or allow others to do so, for U.S. Government purposes.

that computing the HC of a given system is often difficult. Depending on the spatial distribution and size of the DPV systems, and the type of violations considered, different results can be obtained [6]. In addition, the computation of HC traditionaly relies on Monte-Carlo simulations which are time consuming by nature [6] [7]. By looking at diverse sequences of actions and evaluating them, the THC problem relies heavily on many HC computations which can make the problem quickly intractable. To overcome this challenge, we demonstrate the effectiveness of domain-informed heuristic methods for speeding the search by reducing the number of states visited.

This paper is organized as follows. Section II introduces notation and details the problem formulation. Section III briefly introduces a software implementation. Section IV presents the main ideas and features of the proposed method. Finally, section V describes results obtained for the EPRI J1 feeder.

II. Problem Overview

A. Notation

This work considers distribution systems on a feeder-by-feeder basis [7]. At the start, a feeder has an initial hosting capacity, denoted by H_0, that can be reached without any modifications to the system. Out goal is to reach a given HC target $H^\star > H_0$ by implementing system upgrades sequentially. Choosing one of these upgrades and implementing it will be called an *action* (or a *decision*) and will be denoted by d. We use the variable $k \in \mathbb{N}$ to denote the time-step. The state of the system depends on k and will be denoted by s_k. We denote by $\mathcal{S}_k$ the state space at step k, and $\mathcal{S} = \cup \mathcal{S}_k$ the full state space.

Depending on the system state, we have a set of possible actions; for example, in a state with voltage violation, possible actions could include: *adding a new regulator*, *changing the settings on an existing regulator*, or *adjusting the PF of the DPV inverters*." We denote by $\mathcal{D}_{s_k}$ the set of available actions, at step k, for the system being in state s_k. As for the state space, $\mathcal{D} = \cup \mathcal{D}_{s_k}$ represents the full set of actions.

Any action $d \in \mathcal{D}_{s_k}$, i.e. any upgrade or sequence of upgrades on system s_k, comes with a strictly positive cost $C_k(s_k, d)$. This cost can be a function of time if we expect the price of technologies to evolve for instance. It can also represent diverse notions such as equipment costs, installation costs, or maintenance costs... In this work, costs represent the cost of the equipments (if any) plus the cost of the installation/modification and come from real utility costs [8]. Because of the available data for this work [8], costs do not depend on time nor on the state of the system, but only on the action itself. Therefore, we simplify the notation to $C(d)$. In other words, changing the setpoints of a regulator, for instance, will always incur the same cost at any point in time, and no matter what decisions we took before.

B. Decision graph

It is convenient to picture the problem as a decision graph where a node represents a state s_k and an edge represents an action d. In this representation, an edge points from node s_k to node s_{k+1} if the action led the system from state s_k to state s_{k+1}. In addition, each edge has a weight associated to it representing the cost of applying this action. We call Going from s_k to s_{k+1} by implementing d a "*transition*," denoted by $s_{k+1} = M(s_k, d)$, where function $M : (\mathcal{S}, \mathcal{D}) \longrightarrow \mathcal{S}$ is called the *transition model*. Depending on the problem, M can be a simple analytic expression, or a more complex model. Likewise, M can be a determistic model, that is applying action d to system state s_k always leads to state s_{k+1}, or a stochastic model. The transition model used for the THC problem is relatively complex and will be detailed in section IV-C.

If a node s_k holds the information for the state it represents, the path from the root s_0 to this node contains the sequence of upgrades required. This includes the sequence of decisions that can be used to compute the total cost to get to the node. For most problems, the decision graph is actually a tree such that this path is unique. For some problems, it might be possible to reach the same node using different sequences of actions such that the decision path is the shortest, i.e. cheapest, path from the root to the node. With a slight abuse of notations, we denote by $C(s_k)$ the cost of the shortest/cheapest decision path going from the root to s_k.

A node is called a solution node if the HC of the system is larger or equal than the target: $H_{s_k} \geq H^\star$. One challenge is that the graph is unknown beforehand, we only know the root, i.e. the initial state. Moreover, we often have no prior information on the solution nodes. It is also possible for the set of solution nodes to be empty, i.e. the target is unreachable for the feeder given the available actions. This makes the problem similar to a graph search problem. Indeed, using a basic breadth-first-search algorithm would reveal the graph until we find solutions or run out of actions. However, we face two main issues: First, the number of nodes in the decision tree grows exponentially with the number of decisions available. Second, and maybe more problematically, the only way to check whether a node is a solution or not is to compute the HC of the system it represents. This computation typically relies on Monte-Carlo simulations, and is computationally expensive. Minimizing the number of nodes we expand is therefore a key consideration and one that is not provided by pure graph search techniques.

III. Shortest Path Tool

This section describes the software implemention of our Shortest Path THC tool to introduce more of the problem structure and code architecture used. The tool is developed in Python and uses OpenDSS for simulation. It was first developed to solve generic sequential decision problems with THC being one of the key use cases. As a result, most of the tool's components are generic and can be used to solve

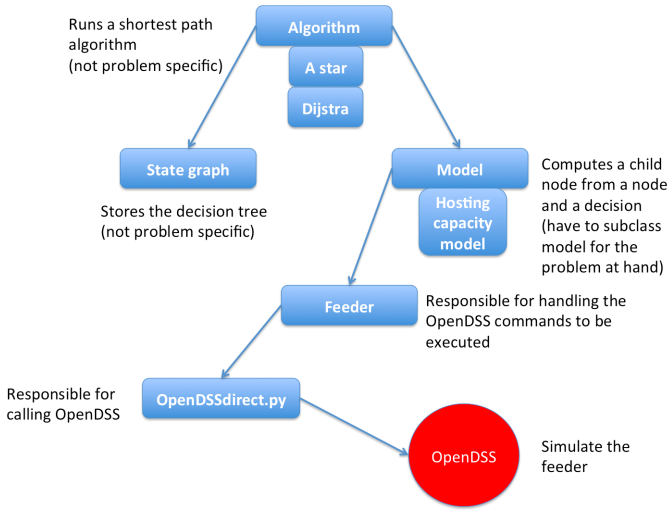

Fig. 2. Code structure of the shortest path target hosting capacity tool.

different problems. Figure 2 shows the code architecture and the relations between the main classes of the tool:

- **State graph:** Data structure that keeps track of visited nodes as well as the decisions linking them together.
- **Model:** Implements the details of the problem. Describes the decision space, the costs, and the transition model.
- **Algorithm:** Uses a state graph and a model to find the solution nodes.
- **Feeder:** Low level class that translates decisions to OpenDSS commands.
- **OpenDSSdirect:** Direct DLL linking Python and OpenDSS

The main idea is that an *algorithm* class needs a *state graph* class to store what has been explored so far, as well as a *Model* class to expand nodes. In other words, the *Model* class implements the transition functions that basically compute the next state from a given state and a decision. At this level, decisions are stored as Python objects and the *Feeder* class is responsible for the translation between them and OpenDSS text commands. It is also responsible for implementing the PV deployments and supporting sudden changes of states at the DSS level, which happen when the *algorithm* decides to move to another node in the *state graph*. *OpenDSSdirect* is a Python library that interfaces the OpenDSS engine enabling cross-platform support.

IV. Proposed Method

A. Informed Heuristics

In general, shortest path algorithms maintain a stack of nodes to expand, initialized with the root, or initial, state. Expanding a node consists of applying all the available decisions at this node and creating one child node per decision. In other words, a child node $s_{k+1} = M(s_k, d)$ represents the state of the system if we apply decision $d \in \mathcal{D}_{s_k}$ on node s_k. The child nodes are then inserted in the stack of nodes to be expanded.

Informed heuristic search techniques [9] focus mainly on how nodes should be inserted in the stack. The main idea is that nodes are inserted according to a ranking function $F : \mathcal{S} \longrightarrow \Re$. For example, $C(s_k) = \sum_{i \leq k} C_i(s_i, a_i)$ will expand in priority the nodes at the end of the cheapest branches. This is effectively the Dijkstra algorithm [10]. Ranking functions used by informed heuristic searches can be written in the general form $F(s_k) = C(s_k) + U(s_k)$ where $C(s_k)$ is the cost from the root to the current node, and U is a heuristic function estimating the cost to go from the current node to the destination. One of the most famous algorithms of this sort is called A* [9] and aims at finding shortest-paths in graphs. In a road graph for example, the heuristic used by A* can be the straight-line distance from the current node to the destination. The challenge here is that, contrary to a shortest path problem where we know the destination, we do not have much information on solution nodes. Our only piece of information is that they should satisfy $H_{s_k} \geq H^\star$. The heuristic function we use computes the cost of $H^\star - H_{s_k}$ by using a per-unit estimation cost based on previous actions:

$$U(s_k) = (H^\star - H_{s_k}) \frac{C(s_k)}{H_{s_k} - H_0} \tag{1}$$

Another key aspect revolves around using the costs of previously found solutions as bounds for the search. More precisely, if the cost from the root to the current node plus the cost of the action considered is larger than the best known solution, then evaluating this action is pointless since it will lead necessarily to more expensive solutions. This approach, known as branch and bound [9] [11], is a classic and very useful way of pruning the graph as long as one can provide good bound values. A bound value can either be provided by the user as the cost of an already known solution, or as the maximum budget available. If no information is known beforehand, the initial bound is set to infinity such that nothing will be pruned until a complete solution is found. In the present work, we do not assume any prior information such that bounds have to be discovered.

Because of this, finding a good solution quickly, even if not the optimum, provides a key benefit. This means that the search algorithm should be able to quickly identify promising regions of the space, but keep the ability to change its focus quickly if needed. In this work, we use a simulated annealing heuristic [12] to address this exploration vs. exploitation problem. The main idea is that, non-promising actions might get picked with a time-decreasing probability. In its early stage, the algorithm will have the possibility to explore actions that might look poor at first but might turn out to be good when looking further into the future. While in the late stages, the algorithm will focus on promising regions discovered so far. An important point is that these heuristics only target the ways nodes from the queue are evaluated, they do not impact the optimality of the solution returned. As long as the queue is emptied, either by evaluation or by pruning, the solution returned will be the global optimum.

B. Algorithm

Initialize empty stack $S = \emptyset$;
Provide transition model $M : (\mathcal{S}, \mathcal{D}) \longrightarrow \mathcal{S}$;
Provide heuristic function $F : \mathcal{S} \longrightarrow \Re$.;
Compute H_0;
Set the target $H^\star > H_0$;
Initialize cost of best solution $c^\star = \infty$;
Create root node: $root = (H_0, C = 0, k = 0)$;
Add root node to the stack: $S \leftarrow root$;
while $|S| > 0$ **do**
  With probability p, pick a random node s_k from the stack;
  With probability 1-p, pick top node s_k from the stack;
  for *decision* $d \in \mathcal{D}_k$ **do**
    Restore state s_k;
    if $C(s_k) + C_k(d) \leq c^\star$ **then**
      Compute child $s_{k+1} = M(s_k, d)$;
      if s_{k+1} *is a solution node* **then**
        $c^\star = MIN\{c^\star, C(s_{k+1})\}$;
      else
        Compute heuristic value: $F(s_{k+1})$;
        Insert child in the stack according to its value: $S \leftarrow s_{k+1}$;
      end
    end
  end
end

Algorithm 1: Shortest path Target Hosting Capacity

Algorithm 1 explains the main steps of the search in pseudo-code. The main inputs are the transition model $M : (\mathcal{S}, \mathcal{D}) \longrightarrow \mathcal{S}$ that computes the state s_{k+1} resulting from applying an action $d \in \mathcal{D}$ to state s_k, and the heuristic function $F : \mathcal{S} \longrightarrow \Re$ that attributes a real value to a state.

An empty stack is initialized and the root node, i.e. the initial state, is pushed to the stack. In its initial state, the system can host a given number of PVs "for free", i.e. without having to pay for any upgrades, such that we have an initial HC $H_0 \geq 0$ with an associated cost $C_0 = 0$. The cost of the best known solution is set to infinity, and a target $H^\star > H_0$ is provided. Obviously, if the target is too small, any decision will reach it, and if it is too high, the problem might not have a feasible solution.

The stack holds the states which still need to be expanded, but ranks them according to the heuristic F such that the node on top of the stack is the best node according to F. As discussed in section IV-A, we introduce a probability p, which might be time-dependant, to select a random node rather than the top one. When selecting a node from the stack, we first need to restore the state of the system. In other words, all the decisions leading from the root to this node are re-applied on a cleaned system.

Expanding a node s_k is the operation that consists in computing all the children nodes $s_{k+1} = M(s_k, d),\ \forall d \in \mathcal{D}_{s_k}$. Since this operation is time consuming, we compute a transition only if the current cost $C(s_k)$ plus the cost of the decision $C_k(d)$ is smaller than the cost of the best solution found so far $c^\star$. It is possible to add some filtering on the decision set $\mathcal{D}_{s_k}$ to lower the number of decisions to evaluate. For example, we might remove actions that does the exact opposite of what was done in the previous time step since this is equivalent to spending money to do nothing. We might also remove actions applying to the equipment we changed in the previous time step.

If a child node is a solution node with a better cost than the best solution found, then the best solution is updated. Otherwise, the child node is inserted in the stack at the ranking position given by the heuristic F. Another implementation to speed the algorithm up is to insert only child nodes that have a better HC than their parents.

C. Transition Model

Unlike traditional shortest path problems where the transition has a closed form set of equations, the transition model $M : (\mathcal{S}, \mathcal{D}) \longrightarrow \mathcal{S}$ used for the target hosting capacity problem relies heavily on Monte-Carlo simulations to estimate the hosting capacity. If we wish to evaluate the impact of a decision d_1 on the HC, we have to consider multiple PV deployments as well as multiple load scenarios. In this work, we sample the locations for the PVs from a uniform distribution, meaning that all buses that can physically host a PV are equally likely to be selected (a bus can also host more than one PV if it gets selected at multiple times). Sampling from this distribution, we add PVs until a voltage violation occurs (over voltage or under voltage). Other type of violations such as line thermal violations are being implemented but are not considered in the present work.

For each sample PV deployment, we consider two scenarios: peak load and minimum load and stop as soon as a violation occurred for at least one of these scenarios. The tool uses random seeds to make sure that all decisions are evaluated on the same PV deployments, which allows a fair comparison. In this sense, a sample can be thought of as a PV deployment scenario that will be played every time we evaluate a decision. Let $N_{samples}$ be the number of samples used to compute a transition. This means that we compute the HC of the feeder $2N_{samples}$ times every time we evaluate a decision (peak load and min load).

Obviously, the parameter $N_{samples}$ has a strong impact on the algorithm's speed. The tool lets the user specify the number of samples (which might be a function of time if more precision is needed for some specific periods) as well as the load scenarios to consider. In this way, it is possible to run pre-analysis, with a poor precision, on a laptop to make sure that the problem settings are correct before increasing the number of samples and running it in parallel.

V. Main Results

We tested our approach using the EPRI J1 test feeder which is a real-world, full-scale distribution feeder. Table I presents

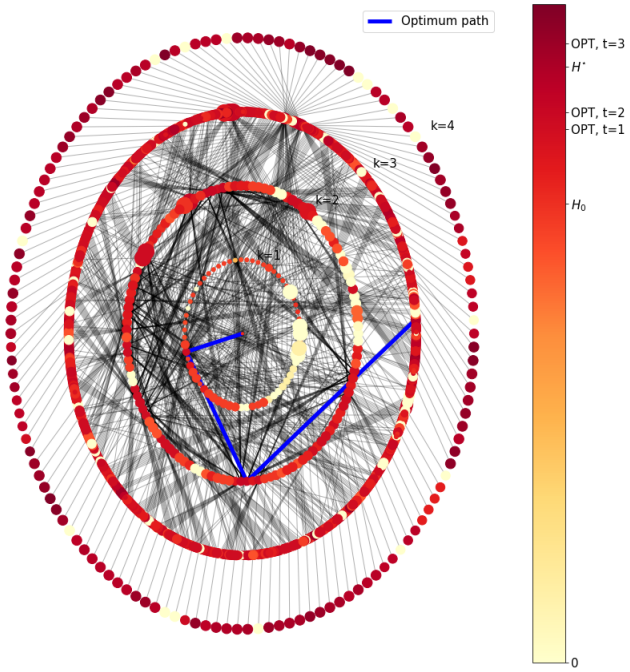

Fig. 3. Decision tree obtained on the EPRI J1 feeder. The size of a node is proportional to its cost (the cost of the decision path leading to it), and its color indicates the hosting capacity reached.

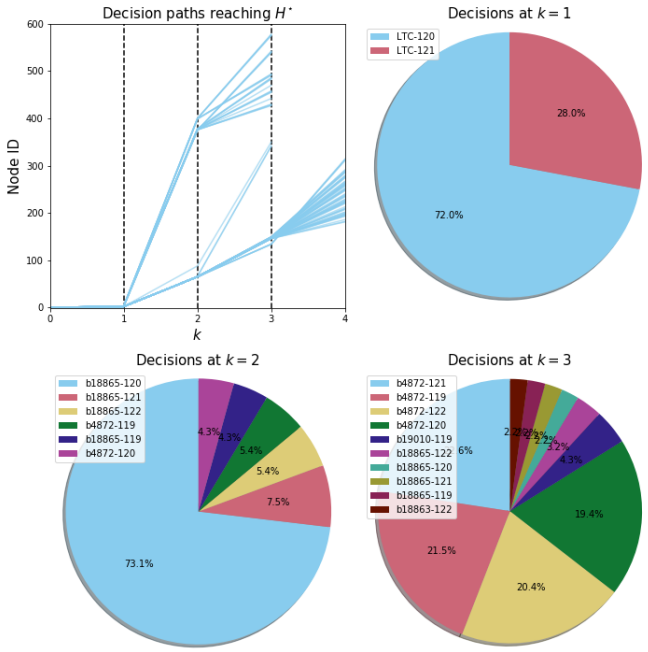

Fig. 4. Target reaching paths and the distributions of decisions for $k \in [1, 3]$

the system upgrade options (decisions) and corresponding costs obtained from [8]. It also includes selection parameters. For example, location, phase, and control settings are required when adding a new capacitor bank. This makes the decision space grow extremely fast requiring restricting the possibilities for some parameters. It is intractable to consider all buses as potential locations since some locations do not make domain sense or are so close that the results would be the same. One way to deal with this kind of issues is to rely on rules of thumbs used by power system engineers on the field. Concerning capacitor banks additions, one rule of thumb used in practice is to place the new capacitor $2/3$ of the way down the length of the feeder or of the line section from the last capacitor. In the present work we use this as a way to filter relevant locations. Another filter could also be the physical space available at each bus if available.

Figure 3 shows the decision tree obtained on the EPRI J1 feeder with decisions described in table I. The decision set has 63 decisions and is constant over time. The peak and minimum loads are also assumed to stay constant over time but the framework is readily extendable to evolving load conditions. The graph of figure 3 contains 1195 nodes, meaning that 1195 nodes were visited by the algorithm during the search. The full graph with the same number of levels contains $\sum_{k=0}^{k=4} 63^k = 16007041$ nodes, meaning that the algorithm was able to find the optimum solution by looking at only 0.0075% of the possible nodes. For clarity, the nodes are organized in concentric circles, each circle corresponding to a given decision step.

As is clearly visible, some decisions (nodes in light colors) do not improve HC and might even deteriorate it. Some particularly bad decisions result in violations before any PV deployment and therefore in a null HC. The optimum decision path is represented by the thick blue lines and is, in this example, a sequence of three actions: changing the LTC setpoints first and two voltage regulator setpoints modifications after, for a total cost of $8000 + 2500 + 2500 = 13000USD$. This means that in this example it is better to first spend more than three times the cost of the cheapest decision available ($2500USD$), which is something that a cost-minimizing greeedy approach would not capture.

Although the optimum is reached at $k = 3$, the algorithm still explores some alternate paths further to ensure there is no better solutions. As visible on figure 3, it stops at $k = 4$ because it could not find any node in $k = 5$ with a potential better HC with a smaller cost. Note that this could have been possible because the cheapest possible cost, regardless of HC improvements, at $k = 5$ is $5 \times 2500 = 12500USD$, slightly less than the optimum found.

The blue lines on figure 3 shows the optimum decision path for this example. Nonetheless, when running the algorithm, other decision paths reaching the target are found. Figure 4 shows what these paths are doing in terms of nodes visited in the graph and in terms of decisions taken. The top left subplot shows the paths in terms of node IDs. All path starts at node 0 (i.e. the initial state) at $k = 0$, at $k = 1$ there is a very small diversity of nodes lying on these paths since there are only two nodes visited (2 and 3). As k increases, the diversity of possible nodes grows.

The pie charts of figure 4 shows the distribution of deci-

TABLE I
DECISIONS CONSIDERED, RELATED PARAMETERS, AND COSTS

Upgrade option (decision)	parameters	costs
Change the setpoints of existing regulators	vreg, band	2500 USD/Unit
Change the setpoints of existing capacitor banks	controlON, controlOFF	7200 USD/Unit
Change the LTC control settings	vreg	8000 USD/Unit
Add new capacitor banks	location, phase, var, controlON, controlOFF	• 600 kvar: 10723 USD/Unit • 900 kvar: 13747 USD/Unit • 1200 kvar: 32200 USD/Unit
Add new regulator	location, phase, vreg, band	55000 USD/unit
Remove existing capacitor	-	3000 USD/unit

sions on these paths for $k \in [1, 3]$. At $k = 1$, we can see that about 72% of the paths change the LTC settings to 120 while the rest also change the LTC settings but with 121 as a setpoint. At $k = 2$ and $k = 3$, all decisions are voltage regulator setpoint modifications but the diversity increases as the best decision depends on the history of each path. We can also see that some paths reach the target at $k = 3$ (like the optimum path) and other at $k = 4$.

Figure 4 also shows that using a cost-minimizing greedy approach would necessarily give more expansive solutions (if any) since all the target-reaching paths, that were not pruned because of prohibitive costs, start with the expensive LTC decision rather than cheaper ones like setpoints modification on regulators or capacitors.

VI. DISCUSSION

With these limited set of decisions, the best sequence of actions is often a combination of modifications on existing equipments (LTC, regulators, capacitor banks) rather than adding new equipments to the system. Given the cost difference ($2.5kUSD$ to change a regulator setpoint and $55kUSD$ to add a new regulator for example) this is easily understandable. Nonetheless, adding other decisions—e.g. advanced communications and controls, relocating existing equipment—and looking further in the future (by setting more aggressive targets) is expected to result in even more situations where more expensive upfront investments prove the optimal approach to higher hosting capacities. This also means that we could see very different paths leading to the target: long paths implementing many inexpensive decisions and shorter paths implementing a few more expensive upgrades. Looking at all these paths, and not just at the optimum one, could provide some insights into the "preferable" sequence of actions. In the same way Google Maps gives multiple itineraries to reach a destination, leaving the "*highway vs. pretty back roads*" choice to the user could provide different solutions with similar costs, leaving the final decision to the planner.

VII. CONCLUSION

We propose a new method for the THC problem in the context of distribution planning. We show that our approach, by looking into the future, is able to perform better than myopic upgrades commonly applied in U.S. distribution planning. The problem becomes quickly intractable such that innovative techniques to search the space efficiently are needed. We present some promising techniques using adaptations to traditional shortest path algorithms. These approaches could be very useful for utilities to reduce the costs of system upgrades, or by providing a cost reference against which other solutions could be compared. Future work involves adding diversity to the decision space and enabling parallelism such that long simulations can be run in a high performance computing environment.

REFERENCES

[1] B. Palmintier, R. Broderick, B. Mather, M. Coddington, K. Baker, F. Ding, M. Reno, M. Lave, and A. Bharatkumar, "On the path to sunshot. emerging issues and challenges in integrating solar with the distribution system."

[2] F. Ding, A. Pratt, T. Bialek, F. Bell, M. McCarty, K. Atef, A. Nagarajan, and P. Gotseff, "Voltage support study of smart pv inverters on a high-photovoltaic penetration utility distribution feeder," in *2016 IEEE 43rd Photovoltaic Specialists Conference (PVSC)*, June 2016, pp. 1375–1380.

[3] T. Stetz, K. Diwold, M. Kraiczy, D. Geibel, S. Schmidt, and M. Braun, "Techno-economic assessment of voltage control strategies in low voltage grids," *IEEE Transactions on Smart Grid*, vol. 5, no. 4, pp. 2125–2132, July 2014.

[4] F. Ding and B. Mather, "On distributed pv hosting capacity estimation, sensitivity study, and improvement," *IEEE Transactions on Sustainable Energy*, vol. 8, no. 3, pp. 1010–1020, July 2017.

[5] E. P. R. Institute, "Stochastic analysis to determine feeder hosting capacity for distributed solar pv," 2012. [Online]. Available: https://www.epri.com/#/pages/product/1026640/

[6] F. Ebe, B. Idlbi, J. Morris, G. Heilscher, and F. Meier, "Evaluation of pv hosting capacity of distribuion grids considering a solar roof potential analysis: Comparison of different algorithms," in *2017 IEEE Manchester PowerTech*, June 2017, pp. 1–6.

[7] M. Rylander, J. Smith, and W. Sunderman, "Streamlined method for determining distribution system hosting capacity," *IEEE Transactions on Industry Applications*, vol. 52, no. 1, pp. 105–111, Jan 2016.

[8] K. Horowitz, "Distribution system upgrade unit cost database," National Renewable Energy Laboratory, Tech. Rep., 2017. [Online]. Available: https://www.nrel.gov/solar/distribution-grid-integration-unit-cost-database.html

[9] C. Grosan and A. Abraham, *Informed (Heuristic) Search*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 53–81. [Online]. Available: https://doi.org/10.1007/978-3-642-21004-4_3

[10] E. W. Dijkstra, "A note on two problems in connexion with graphs," *Numer. Math.*, vol. 1, no. 1, pp. 269–271, Dec. 1959. [Online]. Available: http://dx.doi.org/10.1007/BF01386390

[11] E. L. Lawler and D. E. Wood, "Branch-and-bound methods: A survey," *Operations Research*, vol. 14, no. 4, pp. 699–719, 1966. [Online]. Available: https://doi.org/10.1287/opre.14.4.699

[12] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, "Optimization by simulated annealing," *Science*, vol. 220, no. 4598, pp. 671–680, 1983. [Online]. Available: http://www.jstor.org/stable/1690046