Energy Syst (2015) 6:417–438
DOI 10.1007/s12667-015-0148-6

ORIGINAL PAPER

Toward scalable stochastic unit commitment

Part 2: solver configuration and performance assessment

Kwok Cheung[1] · Dinakar Gade[2] · César Silva-Monroy[3] · Sarah M. Ryan[4] · Jean-Paul Watson[5] · Roger J.-B. Wets[6] · David L. Woodruff[7]

Received: 3 May 2014 / Accepted: 26 March 2015 / Published online: 29 April 2015

Abstract In this second portion of a two-part analysis of a scalable computational approach to stochastic unit commitment (SUC), we focus on solving stochastic mixed-integer programs in tractable run-times. Our solution technique is based on Rockafellar and Wets' progressive hedging algorithm, a scenario-based decomposition strategy for

✉ Jean-Paul Watson
jwatson@sandia.gov

Kwok Cheung
kwok.cheung@alstom.com

Dinakar Gade
dinakar.gade@gmail.com

César Silva-Monroy
casilv@sandia.gov

Sarah M. Ryan
smryan@iastate.edu

Roger J.-B. Wets
rjbwets@ucdavis.edu

David L. Woodruff
dlwoodruff@ucdavis.edu

1 Alstom Grid, Redmond, WA, USA

2 Sabre Holdings, Southlake, TX, USA

3 Electric Power Systems Research Department, Sandia National Laboratories, Albuquerque, NM, USA

4 Department of Industrial and Manufacturing Systems Engineering, Iowa State University, Ames, IA, USA

5 Analytics Department, Sandia National Laboratories, Albuquerque, NM, USA

6 Department of Mathematics, University of California Davis, Davis, CA, USA

7 Graduate School of Management, University of California Davis, Davis, CA, USA

solving stochastic programs. To achieve high-quality solutions in tractable run-times, we describe critical, novel customizations of the progressive hedging algorithm for SUC. Using a variant of the WECC-240 test case with 85 thermal generation units, we demonstrate the ability of our approach to solve realistic, moderate-scale SUC problems with reasonable numbers of scenarios in no more than 15 min of wall clock time on commodity compute platforms. Further, we demonstrate that the resulting solutions are high-quality, with costs typically within 1–2.5 % of optimal. For larger test cases with 170 and 340 thermal generators, we are able to obtain solutions of similar quality in no more than 25 min of wall clock time. A major component of our contribution is the public release of the optimization model, associated test cases, and algorithm results, in order to establish a rigorous baseline for both solution quality and run times of SUC solvers.

1 Introduction

While there is a significant body of research on stochastic unit commitment (SUC) in the power systems literature (see [15,18,21,23,26] for a representative sample), these efforts have not yet been successfully transferred to real-world industrial contexts. This is due in large part to the well-known computational difficulty of SUC, where even small cases with a handful of scenarios can take hours to solve [19]. A survey of prior algorithmic approaches to SUC is provided in [14]. An analysis of this survey indicates that the current state-of-the-art for SUC can tackle approximately 50 scenarios on instances with approximately a hundred thermal generation units, achieving solutions in several hours of run time. Further, many of those studies consider short (24-h) time horizons (there are exceptions; e.g., [3] considers an entire week for a multi-stage model to be used for bidding), or use simplified models of industrial unit commitment problems.

The primary purpose of this paper is to detail a research effort dedicated to developing a SUC solver ultimately capable of achieving solutions in tractable (e.g., $<$30 min) run-times, given realistic numbers of time periods (e.g., 48) on realistic and industrial-scale power systems. In this context, a "solution" refers to a feasible non-anticipative commitment schedule, with associated expected cost *and* optimality bound. We demonstrate significant progress toward this goal, considering a variant of the well-known deterministic WECC-240 test case [16]. We leverage modest-scale parallelism to achieve the goal run-times, employing commodity computing capabilities that an ISO/utility either presently possesses or is likely to acquire in the near future. Fundamentally, our primary goal is to demonstrate the viability of SUC in a real-world applications context.

Our computational experiments proceed in the context of load scenarios generated via the approximation procedure described in the companion paper [6]. Accurate assessment of SUC solvers requires accurate load scenarios, as the latter can potentially impact algorithm performance. We limit the investigations of solver performance to given, pre-specified sets of load scenarios. Issues relating to scenario reduction and sampling, out-of-sample solution validation, and quantification of cost savings are beyond the scope of the present contribution. Rather, we focus on the goal of demon-

strating that operational run-times can be achieved for SUC instances with reasonable numbers of realistic load scenarios—a key step toward ultimate commercial adoption of both the underlying uncertainty model and solver.

Secondary goals of this paper are (1) to construct a realistic, validated, industrial-scale, and publicly available SUC model along with sets of corresponding instances and (2) to establish performance baselines for these instances, in terms of both solution quality (i.e., lower and upper bounds) and run time. The availability of such instances and corresponding performance baselines—particularly with high-accuracy scenario sets—is critical to driving SUC solver research, in order to quickly identify and focus on the most promising algorithmic alternatives. To date, SUC research has not been performed in such a context, i.e., direct and accurate comparisons of solver run-times are absent in the literature.

The remainder of this paper is organized as follows. We describe the core deterministic and stochastic optimization models used in our analysis in Sect. 2. Our solution approach is detailed in Sect. 3. Our case study data, based on the deterministic WECC-240 test case, is described in Sect. 4. We describe computational experiments and associated results in Sect. 5. We then conclude with a summary in Sect. 6.

2 Unit commitment models

We now introduce the core optimization models used in the analysis. We begin by introducing the deterministic unit commitment (UC) optimization model in Sect. 2.1. We then discuss the extension of this core model to a two-stage stochastic programming context in Sect. 2.2.

2.1 Core deterministic UC model

Various deterministic models for UC have been proposed in the literature, e.g., see [10,13]. As a basis for our SUC model, we adopt the deterministic UC model introduced by Carrion and Arroyo (CA) [2]. We defer to [2] for details of the core UC decision variables and constraints. We have extensively validated our UC model against Alstom's **e-terra***market* [5] commercial UC model, to ensure correctness, inclusion of industrially relevant constraints, and validity for practice. We note that specific variable and constraint encodings used by Alstom were not available to us. Rather, we ensured that the solutions resulting from the two models were identical, within numerical tolerances. We have implemented our version of the CA deterministic UC model in the open-source Pyomo algebraic modeling language [9].

Our computational experiments focus on reliability unit commitment (RUC). Integration of uncertainty into the day-ahead market (DAM) UC requires modification of core market structure, which is beyond the scope of the present paper. However, in the interest of verifying that SUC could be solved in a DAM context, we treat *all* generators as available for commitment. No modification of market structures would be needed to use stochastic UC in the RUC, i.e., the process executed by an ISO following the close of the DAM and prior to next day operations. In practice, a large proportion of the UC decision variables are fixed as a result of the DAM UC, such

that the number of "free" commitment variables in the RUC is significantly reduced (e.g., typically half or more generators are committed in the DAM UC). As a result, our computational time results should be treated as very conservative for purposes of the RUC.

The choice of the CA deterministic UC model over alternatives reported in the literature was based on preliminary computational experiments on an internal set of test cases. However, our specific choice is not critical to the scalability results reported subsequently in this paper. Other researchers have reported that alternatives to the CA model can yield faster solution times [10,13]. Ultimately, if those reductions hold for our UC test cases, then the run times reported subsequently for our PH solution algorithm would be further improved. In other words, the run times reported in Sect. 5 should be treated as upper bounds relative to the full range of possibilities that exist in the deterministic UC literature.

2.2 Two-stage stochastic UC model

Following [23] and others, we extend a basic deterministic UC model into a two-stage stochastic UC model. To construct the stochastic UC model, the variables in the core deterministic UC model are partitioned into two stages, mirroring the structure of the corresponding real-world decision process. The first-stage variables consist of the thermal generator on/off state indicator variables, one for each hour in the planning horizon (in the CA UC formulation, startup and shutdown variables are implicit—this is not the case in other deterministic UC models). All remaining variables, including thermal generator dispatch levels, reserve allocations, and cost computations, are classified as second-stage variables. First-stage variables are required to be *non-anticipative* in a two-stage stochastic program, such that their value does not depend on the scenario that is ultimately realized. Given a set S of scenarios and their attached probabilities, in our case obtained using the procedures described in the companion paper [6], a two-stage stochastic UC model can be constructed by creating an instance of a deterministic UC model for each scenario $s \in S$. To enforce non-anticipativity across the first-stage variables, we then impose equality constraints among the instances of the corresponding variables in all scenarios. The resulting model is known as an explicit *extensive form* of the corresponding two-stage stochastic program, in which the first-stage decision variables (for reasons that are discussed in Sect. 3) are replicated for each scenario instance. We take minimization of the sum of first stage cost (e.g., startup, no-load, and shutdown costs) plus expected second stage cost (e.g., production cost) as the optimization objective. However, we note that our SUC model and solver extends trivially to risk-oriented optimization metrics such as conditional value-at-risk. For further details regarding the structure and properties of two-stage stochastic programs, we refer to [20].

To present our solution algorithm for stochastic UC described in Sect. 3, we make use of an abstract formulation of a two-stage stochastic program. In this abstract formulation, x and y represent vectors of first stage (e.g., unit commitment) and second stage decisions, respectively. The functions $f(x)$ and $g_s(x, y)$ respectively compute the first stage (e.g., startup and shutdown) and second stage costs for a scenario s. If

the future were known with certainty, there would be only one scenario, s, and the resulting optimization problem could be written as:

$$\min f(x) + g_s(x, y) | (x, y) \in Q_s. \tag{1}$$

The notation $(x, y) \in Q_s$ abstractly captures the requirement that any combination of feasible first stage and second stage decision vectors must satisfy all constraints imposed by the laws of physics and system operating policies under scenario s. In the case of SUC, each scenario corresponds to the deterministic RUC that is presently solved in ISO and utility daily operations.

Of course, the future cannot be known with certainty, so one must determine a non-anticipative x and corresponding scenario-specific y_s such that (1) the sum of first stage costs plus the expected second stage costs is minimized and (2) $(x, y_s) \in Q_s$ for all $s \in \mathcal{S}$. This formulation is given as follows:

$$\min \sum_{s \in \mathcal{S}} p_s \left[f(x_s) + g_s(x_s, y_s) \right] \quad s.t. \tag{2}$$

$$(x_s, y_s) \in Q_s, \quad \forall s \in \mathcal{S} \tag{3}$$

$$x_s = \sum_{i \in \mathcal{S}} p_i x_i, \quad \forall s \in \mathcal{S} \tag{4}$$

The objective function given by (2) represents the expected cost and the constraints (3) summarize logical and physical system requirements. Constraints (4) enforce the requirement that all scenarios have an identical, non-anticipative x vector. There are other ways to implement the non-anticipativity requirement, but this method is appropriate in the context of the PH solution algorithm, as discussed in Sect. 3.

Building on the Pyomo implementation of the core deterministic UC model, we have implemented our two-stage stochastic UC model in the open-source PySP package for stochastic programming [25]. Both Pyomo and PySP are distributed as part of the Pyomo optimization software package (https://software.sandia.gov/trac/pyomo).

3 Solution approach based on progressive hedging

Following a brief survey of prior efforts involving the solution of stochastic UC models in Sect. 3.1, we describe the basic progressive hedging solution algorithm in Sect. 3.2. We then discuss specializations of the core progressive hedging algorithm to two-stage stochastic UC in Sect. 3.3. Issues related to the deployment and parallelization of the algorithm are detailed in Sect. 3.4. Computation of lower bounds on solution quality is briefly discussed in Sect. 3.5.

3.1 Background

The extensive form of the two-stage SUC is practically insoluble via direct methods such as commercially available MIP solvers, as we report in Sect. 5.3. Similar find-

ings are reported throughout the SUC literature, e.g., see [14]. To achieve tractable run-times for two-stage SUC, decomposition techniques must be leveraged. Two dominant classes of decomposition techniques for general two-stage stochastic programs are time stage-based and scenario-based. The exemplar stage-based technique is the L-shaped method (Benders decomposition) [22]. Exemplars of scenario-based decomposition include progressive hedging (PH) [17] and dual decomposition [1].

One advantage of scenario-based decomposition techniques over their stage-based counterparts is a more uniform distribution of work load in parallel computing environments. In particular, the computational difficulty of the master problem in the L-shaped method can grow significantly as the number of iterations increases, while the sub-problems are typically comparatively easy. Another advantage is that they are easily implemented in situations where software for solving the deterministic version of the problem already exists, and may have been highly customized for efficient solution—as is the case for unit commitment.

As discussed in [14, p. 17], most prior analyses of SUC consider direct solution of the extensive form. Even those studies that use decomposition schemes are limited in the sizes of the test cases considered—typically no more than 100 generators, with 24 time periods, and fewer than 50 scenarios. Further, the reported run-times for the largest of these cases exceeds 30 min, and more typically an hour. Scalability to larger numbers of scenarios and time periods is thus a major and open concern, prior to serious consideration by industry of stochastic UC models and algorithms. Recent algorithmic advances in SUC are reported in [11,12].

3.2 Progressive hedging

PH decomposes the extensive form of a stochastic program by scenario, initially relaxing the non-anticipativity constraints. Non-anticipativity is then restored via an iterative multiplier update scheme. The basic PH algorithm operates as follows:

1: Initialization: $\nu \leftarrow 0$ and $w_s^\nu \leftarrow 0, \forall s \in \mathcal{S}$
2: Iteration 0: $\forall s \in \mathcal{S}$,
$x_s^\nu = argmin_{x,y} f(x) + g_s(x, y) \mid (x, y) \in Q_s$
3: Aggregation: $\overline{x}^\nu = \sum_{s \in \mathcal{S}} p_s x_s^\nu$
4: Iteration Update: $\nu \leftarrow \nu + 1$
5: Multiplier Update: $w_s^\nu \leftarrow w_s^{\nu-1} + \rho(x_s^{\nu-1} - \overline{x}^{\nu-1}), \forall s \in \mathcal{S}$
6: Iteration ν: $\forall s \in \mathcal{S}$,
$x_s^\nu = argmin_{x,y} \left[f(x) + g_s(x, y) + w_s^\nu x + \frac{\rho}{2} ||x - \overline{x}^{\nu-1}||^2 \mid (x, y) \in Q_s \right]$
7: Convergence Check: If all solutions x_s^ν are identical, halt. Otherwise, go to Step 3.

In the PH pseudocode above, we superscript the multipliers w, the first-stage scenario solutions x, and the first-stage variable averages $\overline{x}$ by the iteration counter ν; the w and x are additionally subscripted by the scenario $s \in \mathcal{S}$. Following initialization, PH solves the scenario sub-problems, in order to form an initial "best guess" at a solution that is non-anticipative. PH then updates the estimates of the multipliers w_s^ν

required to enforce non-anticipativity, using a penalty parameter ρ. We observe that while ρ is a scalar in the pseudocode shown above, in general it can have a different value for each variable. Following the multiplier update, PH solves variants of the scenario sub-problems that are augmented with a linear term in x proportional to the multiplier w_s^ν and a quadratic proximal term penalizing deviation of x_s^ν from $\overline{x}^{\nu-1}$. These additional terms, in conjunction with the multiplier updates, are designed to gradually reduce any differences in x_s as PH progresses, eventually yielding a non-anticipative solution x. PH can be trivially accelerated by executing the independent sub-problem solves in Steps 2 and 6 in parallel, with a barrier synchronization immediately following each step. While the pseudocode provided above is specific to two-stage stochastic programs, the algorithm generalizes to multi-stage contexts. Finally, we note that in PH iterations $\nu \geq 1$, individual scenario sub-problem solves can be warm-started using the solution for the corresponding scenario from the previous PH iteration. Since only the objective function is altered between PH iterations, feasibility is guaranteed. Hence, the solution from the previous iteration can be given to the sub-problem solver to establish a good upper bound and to provide a starting point for branching. Such warm-starting has significant practical impact when solving mixed-integer stochastic programs.

The progressive hedging algorithm [17] relies, at least implicitly, on a version of a (projected) augmented Lagrangian method and, consequently, convergence is guaranteed when the given problem is convex but also when the augmentation can be selected to induce local convexity in the modified problem; this has been possible in a significant number of applications. However, this later possibility is out of the question when the "non-convexity" arises from constraints requiring that some of the non-anticipative variables be integer or discrete. In particular, the presence of integer decision variables can induce cycling behavior. However, effective techniques for detecting and breaking cycles have been recently introduced [24]. Further, accelerators are typically necessary to improve PH convergence, in order to achieve practical run-times. These include variable fixing (freezing the values of variables that have converged for the past k PH iterations) and slamming (forcing early convergence of specific variables that have minimal impact on the objective). Both of these techniques are described fully in [24].

3.3 Specialization to stochastic UC

PH was first applied to the SUC problem by Takriti et al. [23], who examined a variant of PH designed to address the possibility of non-convergence in the mixed-integer case. Goez et al. [8] also examine PH for SUC, and compare it with alternative heuristics. While both research groups report promising results, the test cases used are not fully described nor publicly available. Further, in the case of Takriti et al., the experiments were performed on now-dated hardware. Lacking the ability to examine test cases in detail, it is difficult to assess the specific set of features employed in the core deterministic UC models (e.g., the presence of ramping constraints that are binding and the use of complex, realistic representations of startup costs), which in turn have a major impact on computational difficulty. Takriti et al. report experiments on

instances with approximately 100 thermal units and 20 scenarios, with a 168 h planning horizon; they report typical runs took 6 h of CPU time and 100 PH iterations. Goez et al. consider a test case with 32 thermal units, 72 time periods, and approximately 30 scenarios; no run time statistics are reported. In both studies, no configuration or tuning experiments are reported.

PH performance is known to be critically dependent upon the value of the ρ parameter. Poor choices can lead to non-convergence, or extremely slow convergence. In our PH configurations, we use variable-specific ρ values for generation unit commitment on/off variables. For a given thermal generator g, we compute the production cost $\overline{p}_g$ associated with the average power output level. We then introduce a global scaling factor α, and compute generator-specific values $\rho_g = \alpha \overline{p}_g$. This strategy for setting ρ falls in a broad class known as "cost-proportional" ρ, shown to be an effective technique for PH parameterization [24,25].

Another technique we use to improve the performance of PH for stochastic UC is the use of approximate sub-problem solutions in early algorithm iterations. The cost of obtaining optimal solutions to scenario sub-problems is prohibitive in early iterations, and is further not needed—precise estimates of the penalty terms w_s^ν are not necessary. Consequently, for PH iterations 0 and 1, we set the optimality tolerance (i.e., the "mipgap") for scenario sub-problem solves to a value γ_{01}. For PH iterations ≥ 2, we then linearly scale the mipgap as a function of the current value of PH convergence metric δ between γ_{01} and a final value equal to the default mipgap tolerance of the sub-problem solver employed, e.g., 1e$-$5 in the case of CPLEX. This technique is fully documented in [24].

We observe that a lower mipgap is less costly in later PH iterations due to the use of variable fixing strategies, which we now describe. Specifically, our PH implementation for SUC fixes generator commitment variables that have converged to a consistent value for the past μ PH iterations. We use the word "fix" here to mean that we constrain the variable to take the fixed value. The intuition here is that if a particular variable has converged for a number of PH iterations, it is likely to remain fixed in subsequent iterations. The technique, while heuristic, has the benefit of reducing the size of the scenario sub-problems, in turn reducing solve times. Finally, we note that non-anticipativity is enforced only for the generator commitment variables, and not for auxiliary first stage variables (e.g., startup costs) that are fully determined by the commitments. Since the only variables subject to non-anticipativity constraints are binary, expansion of the square in the proximal term results in a quadratic that is simply the product of two binaries, which is linear. Hence, the sub-problem solves are linear MIPs.

A basic implementation of PH is provided by the PySP stochastic programming library [25].

3.4 Parallelization and deployment

As indicated above, parallelization of PH is conceptually straightforward—the sub-problem solves at Steps 2 and 6 are independent, and can execute on distinct processing elements. In a parallel PH environment, a client process is responsible for initiating the request for sub-problem solves, computing the solution averages $\overline{x}^\nu$, and updating

the multipliers w_s^ν. Relative to the (mixed-integer) sub-problem solves, these actions consume a small fraction of the overall run time. Rather, parallel efficiency is limited by the difference between the average and maximum sub-problem solve time, which in practice can be significant. Because the primary performance metric is wall clock time, as opposed to sustained usage of all processors, we ignore issues relating to parallel efficiency in our experiments. Ideally, asynchronous extensions could be employed to further reduce PH run times. Finally, we use the parallel PH execution capabilities available in the PySP stochastic programming library [25], which are in turn built on the Python Remote Objects library (http://pypi.python.org/pypi/Pyro). Pyro provides for parallel execution on distributed memory clusters and multi-core workstations.

3.5 Computation of lower bounds

Mirroring the case for deterministic unit commitment, a goal of SUC solvers is to provide both an implementable solution and some quantification of its quality. Recently, we showed in [7] that a valid lower bound in the stochastic mixed-integer case can be obtained in any iteration ν of PH, simply by solving the optimization problems of the form

$$\min f(x) + g_s(x, y) + w^\nu x_s, \quad \forall s \in \mathcal{S} \tag{5}$$

and computing the probability-weighted average of the resulting costs. We report these bounds in our computational experiments, considering only the bound associated with the final PH iteration. Although not analyzed in this paper, there exists a strong relationship between ρ and the quality of both lower and upper PH bounds, as documented in [7].

Mirroring the case of the basic PH algorithm, the lower bound computation is straightforward to parallelize. The PH bound can be computed even when (as is the case for the experiments reported below) the scenario sub-problems are not solved to optimality, specifically by subtracting the absolute gaps from the resulting costs. As is the case with standard PH primal iterations, we warm-start lower bound scenario sub-problem solves with the corresponding primal scenario solution from the associated PH iteration—although the effectiveness of the warm-start is diminished in this context. Finally, we note that all fixed variables must be (temporarily) freed when solving lower bound scenario sub-problems.

4 WECC-240 case study

As a basis for a test case, we choose the WECC-240 instance introduced in [16], which provides a simplified description of the western US interconnection grid. This instance consists of 85 thermal generators. Because it was originally introduced to assess market design alternatives, we have modified this instance to capture characteristics more relevant to RUC, which were absent or incomplete in the original data. These include startup, shutdown, and nominal ramping limits, startup cost curve data, and minimum generator up and down times. A full description of the modifications, and the case itself, can be obtained by contacting the authors or visiting https://prescient.sandia.gov. The choice of WECC-240 as a baseline test case was driven by the desire to develop a

publicly releasable test case. At present, our ISO New England (ISO-NE) test case (corresponding to the load scenario generation process described in the companion paper [6]) contains proprietary data.

We consider three primary SUC test instances in our experiments, which we construct by scaling ISO-NE load scenarios to match WECC-240 system load characteristics. Additional cases, one for each day in 2011, are available at https://prescient.sandia.gov. Scenarios in the base case, denoted *WECC-240-r1*, are generated by randomly perturbing load from the original WECC-240 case (for a chosen day) by $\pm 10\,\%$. The base case is used for tuning PH parameters. Out-of-sample testing is then performed on two cases based on scenarios constructed via the process described in the companion paper [6]. These cases are denoted *WECC-240-r2* and *WECC-240-r3*, and respectively represent low-variance and high-variance load scenario sets for ISO-NE (corresponding to Figures 6 and 7 in [6]). For each case, we consider instances with 3, 5, 10, 25, 50, and 100 scenarios.

In terms of problem scale, the resulting instances consist of $85 \times 48 = 4080$ non-anticipative binary first-stage commitment variables. The extensive form of the 100-scenario instance contains 2,894,281 variables, 6,944,701 constraints, and 24,072,401 non-zeros, and clearly represents a significant computational challenge.

Finally, we have created larger versions of the base WECC-240 test case, using the following process. First, we generate n copies of each thermal generator in the fleet, and scale the load for each time period in each scenario by n. Second, for each of the resulting thermal generators, we randomly sample a scaling factor $\beta \in [1.0, 1.05]$. We then scale all cost information associated with the generator, relative to the original WECC-240 case, by β; this specifically includes the startup cost parameters and all parameters associated with the production cost curves. While the intent of this perturbation is to make the resulting cases more realistic, a side effect is mitigation of solution symmetry induced by the presence of substitutable generators. We specifically focus on variants of the *WECC-240-r2* case in our experiments, although additional instances are available. We denote the instances corresponding to $n = 2$ and $n = 4$ by *WECC-240-r2-x2* and *WECC-240-r2-x4*, respectively. These instances respectively possess 170 and 340 thermal generators, corresponding to fleet sizes present in smaller ISO-scale systems (e.g., ISO-NE).

5 Empirical results

We now analyze the performance of the PH algorithm for two-stage SUC. We initially consider the *WECC-240-r1* case, for purposes of parameter tuning and analysis. We then fix the PH configuration and examine performance on the out-of-sample and more realistic *WECC-240-r2* and *WECC-240-r3* cases. We analyze scalability to the larger *WECC-240-r2-x2* and *WECC-240-r2-x4* cases. Finally, we consider the impact of tighter deterministic UC models on the performance in PH in Sect. 5.8.

5.1 Computational platforms

Our experiments are executed on two distinct compute platforms. The first represents a commodity higher-end workstation, and consists of eight 8-core AMD Opteron 6278

2.4 GHz processors with 512 GB of RAM. Such a workstation is representative of the type of resource that is likely to be currently available or available in the near term to typical utilities and ISOs, and can be purchased for <$20 K USD. This platform allows for modest-scale parallelism. The second platform, used exclusively for the larger 50 and 100-scenario instances, is Sandia National Laboratories' Red Sky cluster, whose individual blades consist of two quad-core 2.3 GHz Intel X5570 processors and 12 GB of RAM. We observe that the dependency of the results on this particular HPC architecture is negligible, in that a small-scale cluster with similar processors could achieve identical performance. Finally, we note that when using CPLEX to solve scenario sub-problems, results on Intel processors complete in roughly 60 % of the time observed on AMD processors with similar clock speeds.

All parallel PH jobs are allocated a number of processes equal to the number of scenarios, plus additional processes for executing the PH master algorithm and coordinating communication among the sub-problem "solver server" processes. We use CPLEX 12.5 as the extensive form and scenario sub-problem solver, with default parameter settings unless otherwise noted.

5.2 Individual scenario sub-problem difficulty

The overall run-time of PH is largely a function of the difficulty of individual scenario sub-problems, both with and without the augmented objective terms. Thus, we begin the empirical analysis of PH performance considering CPLEX run-times on scenario sub-problems. Specifically, we consider our 100-scenario *WECC-240-r1* instance, executing PH in serial for one iteration using a ρ scaling factor equal to 1 and no variable fixing. Warm-starting between iterations 0 and 1 is enabled, to reflect the actual PH configuration used in subsequent experiments. We vary the mipgap termination threshold γ over $\{0.02, 0.025, 0.03\}$. As the results discussed below indicate, solution times with smaller γ are prohibitive in the context of PH. For each invocation of CPLEX, we allocate two threads; larger thread counts are not realistic in operational environments, where the number of scenarios is likely to exceed the number of available compute cores. In our experiments, PH is executed in serial, to prevent core contention.

In preliminary experimentation, we observed that CPLEX 12.5 performance is significantly improved on a range of deterministic UC instances when the following two options are employed. First, we enable the relaxation induced neighborhood search (RINS) heuristic [4], to be applied every 100 nodes in the branch-and-cut tree. Second, we set the search emphasis to "moving best bound" (option 3). Lacking either of these options, the run-times reported below are significantly inflated. Run-time is impacted to a significantly lesser degree by two additional CPLEX options that we employ: limiting the number of cut passes at the root node to 1, and disabling presolve repetition.

Statistics for the scenario sub-problem solve times (in s) are reported in Table 1. We report average and maximum statistics, particularly as the latter is a key driver in parallel synchronous PH performance. The results immediately highlight the absolute difficulty of the deterministic single-scenario instances associated with the *WECC-240-r1* case, which is consistent with results reported for similarly sized instances [2,13]. Given target run-times on the order of 30 min for SUC solvers to be considered viable in deployment contexts, it is clear that $\gamma \leq 0.02$ is impractical; individual

Table 1 Solve time (in s) and solution quality statistics for the scenario sub-problems associated with the 100-scenario *WECC-240-r1* instance

MIP gap (%)	Solve time (avg./max.)	
	PH iteration 0	PH iteration 1
0.03	34.44/53.81	5.12/7.70
0.025	61.99/123.53	6.11/9.87
0.02	205.75/604.86	9.42/25.74

scenario solve times necessarily bound the time of individual PH iterations. In auxiliary experimentation, we observe that the allocation of additional threads does not drop the solve times appreciably, such that the additional cores can more effectively be allocated to disparate scenario sub-problem solves.

Based on the results presented above, we limit scenario sub-problem solve times in any PH iteration to 2 min in the PH tuning experiments described below in Sect. 5.4, and focus on cases when γ equals either 0.025 or 0.03. Values of γ significantly larger than 0.03—even in early PH iterations—yield very poor PH behavior, for the following reason. In the *WECC-240-r1* instance, feasible solutions are quickly found—leveraging the characteristic that the majority of generators can be committed for all time periods, while maintaining a low power output level. Taken across all scenarios, this can result in premature convergence of PH, to this trivial and highly sub-optimal solution.

Finally, we note that the use of the CPLEX RINS and moving-best-bound options empirically result in rapid reductions in the upper bound, relative to the default settings that more rapidly increase the lower bound. This difference is key, in that it moves PH away from the trivial solutions described above. There are two non-exclusive strategies for achieving a target γ, and we have focused on those that reduce the optimality gap through identification of high-quality incumbent solutions.

5.3 Solving the extensive form

Next, we analyze the computational difficulty of the extensive form of the *WECC-240-r1* instance, as a function of the number of scenarios considered. The results serve as a performance baseline for the PH algorithm, and additionally provide an indication of absolute instance difficulty.

We execute all experiments associated with extensive form solves on a 64-core AMD workstation, allocating the maximum possible number of threads (64) to each CPLEX run. Mirroring the case for individual scenario solves, enabling the RINS heuristic and moving best-bound emphasis yields significant improvements in solution quality relative to the default parameter settings. Consequently, we employ identical CPLEX parameter settings to those described in Sect. 5.2. For each instance, we perform runs with a limit of 2 and 4 h of wall clock time. For each run, we record the optimality gap reported at termination, the incumbent objective function value, the final lower bound, and the observed wall clock time. The latter can differ from the allocated time limit due to the granularity with which CPLEX checks the overall run time against the allocated limit. The results for the 2 and 4 h run limits are reported in Tables 2 and 3, respectively.

Table 2 Solution quality statistics for the extensive form of the *WECC-240-r1* instance, given 2 h of run time

# Scenarios	Objective value	MIP lower bound	Gap %	Run time (s)
3	64,279.18	63,708.67	0.89	7291
5	62,857.52	62,052.75	1.26	7309
10	61,873.01	60,769.78	1.77	7444
25	61,496.24	59,900.40	2.59	7739
50	61,911.74	59,432.08	4.01	8279
100	62,388.85	3500.70	94.39	9379

Table 3 Solution quality statistics for the extensive form of the *WECC-240-r1* instance, given 4 h of run time

# Scenarios	Objective value	MIP lower bound	Gap %	Run time (s)
3	64,278.20	63,797.72	0.75	14,491
5	62,740.67	62,180.86	0.89	14,723
10	61,563.10	60,835.45	1.18	14,630
25	61,455.55	59,963.78	2.36	14,960
50	61,911.74	59,540.87	3.83	15,480
100	62,388.85	59,548.23	4.51	16,562

Examining the results, we immediately observe the absolute difficulty of the *WECC-240-r1* extensive forms. In no case was an optimality gap <0.75 % observed, and for the larger instances—despite the overall run-time and number of cores available to CPLEX—the gaps are significant. For the 50 and 100 scenario instances, processing had not progressed beyond the root node of the branch-and-cut tree, and the root relaxation (LP solve) time consumed a significant proportion of the run-time (e.g., 6084 s in the 100-scenario case). We highlight such behavior to illustrate that parallelism opportunities for a direct solve of the stochastic UC extensive form are limited, given current mixed-integer solver technology. As reported for individual scenario instances, identification of a feasible solution is relatively straightforward in the case of *WECC-240-r1*, specifically a trivial solution in which the majority of generators are committed at low output levels for all time periods. Improvement of this trivial solution often does not occur until beyond an hour of wall clock time, particularly for instances with 25 or more scenarios. Clearly, the difficulty of the root linear programming relaxation solve alone precludes direct solution of the stochastic UC in an operational context. However, the results do provide a key performance baseline.

5.4 Parameter tuning for PH

As discussed in Sect. 3, there are three key parameters underlying our PH algorithm for stochastic UC: the scale factor α used to compute the ρ values for individual generator commitment variables, the choice of initial sub-problem mipgap γ, and the discrete

variable fix lag μ. We now analyze the performance of the PH algorithm for various choices of these parameters, to illustrate their influence on performance in terms of both solution quality and run-time. For brevity, we do not perform a fully crossed experiment, but rather explore a subset of parameter settings based on experience with tuning PH in other domains and the sub-problem solve time statistics reported in Sect. 5.2. In all experiments, we enable the cycle detection logic available in PySP's PH implementation. If cycles are detected, they are heuristically broken simply by selecting the generator commitment variable involved in the cycling (representing a single time period), and fixing the commitment value to maximum observed (0 or 1) across all scenarios. For details of cycle detection logic, we refer to [24].

The initial experiments consist of the following PH configuration: $\alpha = 1.0$, $\mu = 3$ for iteration counter $\nu \geq 1$, immediate fixing at PH iteration 0 of all variables with converged value equal to 0, and an initial mipgap $\gamma = 0.03$. We limit the total number of PH iterations to 100, and record the terminating value of the convergence metric, the final objective function value (which is the expected cost), the total number of variables fixed, and the overall wall clock time. We execute the full set of scenario instances on a 64-core workstation, and only 50 and 100 scenario instances on the Red Sky cluster. Note that if full convergence is achieved, the objective function values correspond to non-anticipative solution. The objective in this experiment is to examine the overall nature of PH convergence on stochastic UC, for a relatively basic configuration. Convergence acceleration mechanisms are discussed subsequently in Sect. 5.5.

The results of this first experiment are reported in Table 4. We observe that in all cases, PH converges to a non-anticipative solution in at most 33 iterations and in no more than 20 min of wall clock time. The number of variables fixed at convergence is typically large, which is suggestive of rapid and largely natural convergence of the scenario sub-problems to a non-anticipative solution. Cycles are infrequently detected and broken, and are more common in cases with fewer numbers of scenarios. With the exception of the smaller 3 and 5-scenario instances, the PH solutions are significantly

Table 4 Solve time (in s) and solution quality statistics for PH executing on the *WECC-240-r1* instance, with $\alpha = 1.0$, $\mu = 3$, and $\gamma = 0.03$

# Scenarios	Convergence metric	Obj. value	PH L.B.	# Vars Fx.	Time
64-core workstation results					
3	0.0 (in 23 iters)	64,727.714	63,188.709	4080	155
5	0.0 (in 26 iters)	62,911.104	61,609.576	4080	163
10	0.0 (in 26 iters)	61,493.375	60,347.220	4080	227
25	0.0 (in 27 iters)	60,990.111	59,875.661	4080	364
50	0.0 (in 17 iters)	60,721.319	59,527.252	4076	584
100	0.0 (in 23 iters)	61,156.832	59,880.559	4080	1218
Red sky results					
50	0.0 (in 29 iters)	60,676.383	59,670.142	4062	514
100	0.0 (in 33 iters)	61,122.781	60,148.285	4073	672

better than the extensive form solutions reported in Tables 2 and 3, even considering those allocated 4 h of wall clock time. All run times are within the range required for operational deployment. Further, we observe that the 50 and 100-scenario results obtained on a 64-core workstation are obtained under conditions in which there is significant core contention among the PH processes. For example, in the 100-scenario run, 200 threads associated with CPLEX are created at each PH iteration, to be allocated across 64 cores. In contention-free contexts, the run times are no more than 11 min.

To assess solution quality in absolute terms, we additionally compute lower bounds using the procedure described in Sect. 3.5. Specifically, we compute the lower bound using PH following only the final iteration, as opposed to during each iteration. For this computation, we allocate each sub-problem solve a limit of 300 s of wall clock time. Relative to the extensive form lower bounds reported in Tables 2 and 3, the PH bounds are of lower quality for the smaller (3, 5, and 10 scenario) instances. However, the trend reverses for the larger instances, such that the PH bounds dominate on the larger 50 and 100 scenario instances. Further, we note that the relative computational effort required to compute the PH bounds is very modest (5 min of wall clock time), and significantly larger run-times will lead to improved lower bounds. In all but one case, the absolute optimality gaps for the PH solutions are <1500 (and more typically 1000) relative to objective function values of approximately 60,000—representing optimality gaps of between 1.5 and 2.5 %.

Next, in an effort to improve solution quality, we replicate the prior experiment with one exception: we decrease the ρ scale factor from $\alpha = 1.0$ to 0.5. As reported in [24], lower values of ρ can improve solution quality, albeit possibly at the expense of increased run-times. The results are shown in Table 5. Relative to the results obtained using $\alpha = 1.0$, we achieve fully non-anticipative solutions in approximately the same run-times. While the solutions are of similar quality (with each configuration achieving the better solution approximately half of the time), the lower bounds obtained with $\alpha = 0.5$ are uniformly improved relative to the base $\alpha = 1.0$ configuration. This is consistent with the empirical observation reported in [7]: smaller ρ values generally yield improved lower bounds in PH. The relative consistency of wall clock times despite the increased number of PH iterations is due to the fact that lower α values yield smaller iteration-to-iteration perturbations to the scenario sub-problems, which in turn increases the effectiveness of warm-start solutions and consequently reduces the sub-problem solve times. In contrast, larger values of α can induce large sub-problem perturbations, such that sub-problem solve times can often significantly exceed (if a limit were not imposed) the values reported in Table 1.

In the next experiment, we reduce γ from 0.03 to 0.025. Lower γ results in improved sub-problem solutions, albeit at increased run-time costs. Intuitively, we expect increases in PH run-times, but with generally improved solutions. The results, shown in Table 6, indicate improved solutions, similar lower bound quality, but no significant increase in run time. The absolute run-times do not exceed 22 min in cases where there is significant multi-core contention; in contention-free situations, the run-times do not exceed 11 min.

Finally, we replicate the previous experiment, with the exception that we increase the PH variable iteration fix lag μ from 3 to 6. Increased fix lags should lead to better solutions, as aggressive fixing runs the risk of premature convergence of particular

Table 5 Solve time (in s) and solution quality statistics for PH executing on the *WECC-240-r1* instance, with $\alpha = 0.5$, $\mu = 3$, and $\gamma = 0.03$

# Scenarios	Convergence metric	Obj. value	PH L.B.	# Vars Fx.	Time
64-core workstation results					
3	0.0 (in 26 iters)	64,518.666	63,123.404	4079	137
5	0.0 (in 24 iters)	62,941.233	61,751.435	4075	144
10	0.0 (in 35 iters)	61,451.731	60,418.577	4078	220
25	0.0 (in 48 iters)	60,988.949	59,930.202	4079	400
50	0.0 (in 40 iters)	60,676.135	59,591.983	4062	694
100	0.0 (in 24 iters)	61,209.183	60,024.991	4080	1342
Red sky results					
50	0.0 (in 49 iters)	60,623.569	59,732.214	4078	472
100	0.0 (in 53 iters)	61,172.185	60,217.081	4060	667

Table 6 Solve time (in s) and solution quality statistics for PH executing on the *WECC-240-r1* instance, with $\alpha = 0.5$, $\mu = 3$, and $\gamma = 0.025$

# Scenarios	Convergence metric	Obj. value	PH L.B.	# Vars Fx.	Time
64-core workstation results					
3	0.0 (in 37 iters)	64,244.647	63,219.026	4080	213
5	0.0 (in 26 iters)	62,739.635	61,800.037	4077	319
10	0.0 (in 27 iters)	61,450.749	60,436.535	4080	261
25	0.0 (in 38 iters)	60,990.129	59,912.846	4071	418
50	0.0 (in 39 iters)	60,675.517	59,602.647	4080	652
100	0.0 (in 18 iters)	61,184.361	60,014.741	4069	1374
Red sky results					
50	0.0 (in 40 iters)	60,623.297	59,749.630	4079	453
100	0.0 (in 57 iters)	61,126.760	60,198.469	4074	667

generator commitments. The results, reported in Table 7, confirm our intuition: $\mu = 6$ results in consistently improved solutions and lower bounds at the expense of increased computation time.

We note that lower values of α generally yield improved lower bounds, as illustrated by our empirical results; this behavior is discussed further in [7]. However, lower values of α also delay primal convergence, due to more gradual changes in the PH weight vectors. In our experiments, for example, $\alpha = 0.1$ yields significantly longer run-times than $\alpha = 0.5$, holding all other parameters constant. The primal solution is often of higher quality, but the configurations can drive the run-times to beyond those required for operational environments—particularly for larger problem instances and large numbers of scenarios.

Table 7 Solve time (in s) and solution quality statistics for PH executing on the *WECC-240-r1* instance, with $\alpha = 0.5$, $\mu = 6$, and $\gamma = 0.025$

# Scenarios	Convergence metric	Obj. value	PH L.B.	# Vars Fx.	Time
64-core workstation results					
3	0.0 (in 20 iters)	64,213.397	63,235.381	4080	508
5	0.0 (in 18 iters)	62,642.531	61,767.253	4079	674
10	0.0 (in 35 iters)	61,396.553	60,476.604	4066	648
25	0.0 (in 22 iters)	60,935.040	59,992.622	4066	761
50	0.0 (in 15 iters)	60,625.149	59,631.839	4034	1076
100	0.0 (in 25 iters)	61,155.387	60,014.571	4080	1735
Red sky results					
50	0.0 (in 16 iters)	60,623.343	59,779.813	4007	404
100	0.0 (in 25 iters)	61,120.943	60,275.744	4080	549

5.5 Convergence accelerators for PH

Our results demonstrate that careful tuning of the PH configuration can yield "natural" convergence to a fully non-anticipative solution. However, in general, additional mechanisms may be employed in cases where this does not occur. One example, variable slamming—discussed in Sect. 3.2, forces non-anticipativity for non-converged variables if the convergence metric associated with PH stops decreasing or decreases at an insufficiently fast rate. Another option is to terminate PH once a sufficient number of first-stage variables have been fixed, and solve a restricted extensive form with the remaining variables free. These accelerators can be used to reduce computation times further than those reported in the experiments above, but perhaps at the expense of solution quality.

5.6 Out-of-sample testing

To demonstrate that the performance associated with our PH configurations is not due to specialized tuning on the specific *WECC-240-r1* case, we now fix the configuration with $\alpha = 0.5$ and $\nu = 3$. Instead of a fixed γ, we employ an innovative adaptive γ strategy, in which we set the initial mipgap, γ_{01} equal to the average gap associated with solutions obtained after 2 min of wall clock time following the iteration 0 solves. This makes sense for problems such as unit commitment where there is a goal to find a solution in a limited amount of wall clock time. In practice, the initial γ is sensitive to the scenario set and test case under consideration making it difficult to set in such a way that the time available is used effectively. Further, this strategy empirically works well for a broad range of instances, including the larger test cases we consider next in Sect. 5.7.

We test this PH configuration on 50-scenario versions of our *WECC-240-r2* and *WECC-240-r3* test cases; the results of these runs are summarized in Table 8. Performance statistics are qualitatively similar across a range of out-of-sample 50 and 100 scenario cases associated with our WECC-240 test case. The results indicate that

Table 8 Solve time (in s) and solution quality statistics for PH executing on 50-scenario *WECC-240-r2* and *WECC-240-r3* instances, with $\alpha = 0.5$, $\mu = 3$, and an adaptive γ strategy

Instance	Convergence metric	Obj. value	PH L.B.	# Vars Fx.	Time
64-core workstation results					
WECC-240-r2	0.0 (in 61 iters)	59,154.305	58,129.192	4076	843
WECC-240-r3	0.0 (in 29 iters)	87,611.436	86,464.088	4080	684
Red sky results					
WECC-240-r2	0.0 (in 27 iters)	58,951.140	58,268.300	3991	814
WECC-240-r3	0.0 (in 53 iters)	87,644.537	86,533.268	4080	626

performance of PH extends to these out-of-sample instances. In particular, run-times do not exceed 15 min (and are more typically on the order of 10 min), and lower bound quality is maintained at approximately 1–2.5 %. Perhaps counterintuitively, the specifics of the load scenarios (e.g., low versus high variability) do not appear to impact PH convergence or solution quality in a systematic manner.

With some exceptions, the wall clock run time tends to increase with the number of scenarios, even taking into account differences in the number of PH iterations. There are two primary causes of this behavior. First, for larger numbers of scenarios in the case of runs on the 64 core workstation, there is the issue of contention for processors. Second, the times required to solve scenario sub-problems are not homogeneous, and the PH algorithm waits for all sub-problems to report a solution before proceeding to the next iteration. Consequently, although the wait time process is noisy, more scenarios generally lead to larger maximum sub-problem solve times given a particular target mipgap.

5.7 Scalability tests

All of the PH configuration tuning and testing of enhancements for SUC described thus far has been conducted on an realistic generator fleet from the WECC, using realistic and high-accuracy load scenarios constructed from historical data associated with ISO-NE. We now consider PH performance on our larger *x2* and *x4* variants the base WECC case, to assess the potential to scalability to larger systems. Recall that these two test case variants respectively possess 170 and 340 thermal generators, with the latter approximating the fleet size of ISO-NE. We only consider runs on the Red Sky cluster, due to the significant increase in scenario sub-problem difficulty (taking advantage of the faster Intel processors). We limit the PH iteration 0 sub-problem solve times to 180 and 240 s for the *x2* and *x4* cases, respectively. Finally, we note that the number of non-anticipative (first stage) binary commitment variables for these cases is 8160 and 16,320, respectively.

The results of the scalability tests, for *x2* and *x4* variants of the base *WECC-240-r2* case, are reported in Table 9; performance is qualitatively similar for a range of other cases based on different load scenario sets. The number of PH iterations required for convergence does not differ from the results observed for the base WECC-240 experiments. Run times for *x2* instances are typically $<$13 min, and the

Table 9 Solve time (in s) and solution quality statistics for PH executing on 50-scenario *WECC-240-r2-x2* and *WECC-240-r2-x4* instances, with $\alpha = 0.5$, $\mu = 3$, and an adaptive γ strategy

Instance	Convergence metric	Obj. value	PH L.B.	# Vars Fx.	Time
Red sky results					
WECC-240-r2-x2	0.0 (in 22 iters)	117,794.429	116,538.868	8159	741
WECC-240-r2-x4	0.0 (in 19 iters)	232,189.338	228,992.984	16, 311	1421

resulting solutions are typically within 1.5 % of optimal. Run-times for the larger *x4* instances increase slightly, to typically <25 min, yielding solutions within 2 % of optimal.

Overall, the results demonstrate the ability of a tuned PH configuration to obtain provably high-quality solutions (within 2 % of optimal), on industrial-scale SUC test cases with a moderate number of realistic load scenarios, in <25 min of wall clock time—leveraging only modest commodity (cluster or shared-memory) parallel computing resources. Fundamentally, this level of performance indicates that both the run-time and scenario generation barriers (the latter is addressed in the companion [6]) to commercial adoption of SUC are mitigable in practice. Finally, we recall that the day-ahead UC process generally results in a significant (between 50 and 75 %) number of fixed commitments, which must be honored in the reliability UC. Thus, the run-time results reported in this paper represent worst-case performance in the case of reliability SUC. In practice, the number of binary commitment decision variables is significantly smaller, yielding faster scenario sub-problem solve times and consequently accelerated PH solve times.

5.8 The impact of improved deterministic formulations on PH

A key advantage of PH over many alternative decomposition schemes relates to the direct impact of improvements to solving the deterministic (single scenario) problem variant on both solution quality and overall run-time. Specifically, advances in both the deterministic problem formulation (e.g., via polyhedral strengthening) and algorithm enhancements for solving the deterministic formulation (e.g., via either parameter tuning or more fundamental changes) can be immediately leveraged—because the decomposition is scenario-based—in all PH scenario sub-problem solves.

This feature is particularly relevant in the case of unit commitment, where researchers have been able to achieve advances in particular in the realm of tighter problem formulations. We now examine a particular advanced deterministic UC formulation, very recently introduced by [10]. This formulation, which we denote MTR (for the authors), focuses on tighter formulations for the constraints associated with computing thermal generator startup costs. Relative to the CA deterministic model, the MTR model yields significant performance improvements due to the introduction of the alternative representation of startup cost computation constraints.

Table 10 Solve time (in s) and solution quality statistics for PH executing on the *WECC-240-r1* instance, with $\alpha = 0.5$, $\mu = 3$, and the MTR deterministic UC model

# Scenarios	Convergence metric	Obj. value	PH L.B.	# Vars Fx.	Time
64-core workstation results					
3	0.0 (in 36 iters)	64,141.771	64,109.021	4080	237
5	0.0 (in 23 iters)	62,628.532	62,499.212	4080	161
10	0.0 (in 26 iters)	61,384.016	61,327.734	4080	215
25	0.0 (in 41 iters)	60,927.903	60,850.717	4080	366
50	0.0 (in 11 iters)	60,617.311	60,470.956	4044	318

On our WECC-240 problem instances, the MTR is able to solve deterministic scenario sub-problems to optimality in reasonable (at most a minute) run times, eliminating the need for consideration of the mipgap parameter γ. In Table 10, we show the results for PH when running on the *WECC-240-r1* instance, with $\alpha = 0.5$, $\mu = 3$, and using the MTR model. We immediately observe that while the run times are similar to those observed in runs with the CA model, both the objective function values obtained and the corresponding lower bounds are significantly improved. In particular, the achieved optimality gaps are now expressed in fractions (e.g., <0.1) of a percent. Similar results are obtained on the out-of-sample test cases, and on a wide range of other WECC-240 and larger problem instances.

Overall, our experiments involving the MTR deterministic UC model strongly highlight the key property that advances in the solution of deterministic UC can immediately and seamlessly transfer to solution of the stochastic UC using PH. Advanced UC features, such as higher resolution modeling of combined cycle units, can be incorporated similarly.

6 Conclusions

Driven by the desire to directly incorporate representations of uncertainty, many researchers have explored development of algorithms for solving the SUC problem. Yet, these advances have not yet impacted practice, primarily due to the computational challenge of the problem. In this paper, we describe a decomposition-based strategy for solving the SUC problem, based on the progressive hedging algorithm of Rockafellar and Wets. Leveraging various advances over the past decade in the configuration, tuning, and lower bounding of progressive hedging for the SUC, we demonstrate tractable ($\leq$15 min) solve times on the WECC-240 test case with a reasonable number of scenarios. Further, the results scale to test cases with 170 and 340 thermal generators, where we obtain solutions in $<$25 min of run time. The resulting solutions are provably within 1–2.5 % of optimal.

These performance levels can be achieved with both small-scale multi-core workstations and commodity distributed memory clusters. Both platforms represent computing capabilities either currently deployed at ISOs and utilities, or likely to be deployed in the near future. Associated with our results are an extensive set of test cases for

medium to large-scale SUC, filling a critical gap in the literature and establishing a performance baseline for SUC solvers.

We are presently engaged in efforts to increase the scalability of our approach, focusing on larger-scale, proprietary, ISO test cases and scenarios concurrently considering uncertainty in load and renewables output. The PH algorithm we described is immediately extensible to the multi-stage case, which may have relevance for day ahead unit commitment, but could also important for intra-day, as well as longer term planning.

Acknowledgments Sandia National Laboratories is a multi-program laboratory managed and operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation, for the US Department of Energy's National Nuclear Security Administration under Contract DE-AC04-94-AL85000. This work was funded by the Department of Energy's Advanced Research Projects Agency-Energy, under the Green Energy Network Integration (GENI) project portfolio, and by Sandia's Laboratory Directed Research and Development program.

References

1. Caroe, C., Schultz, R.: Dual decomposition in stochastic integer programming. Oper. Res. Lett. **24**(1–2), 37–45 (1999)
2. Carrion, M., Arroyo, J.: A computationally efficient mixed-integer linear formulation for the thermal unit commitment problem. IEEE Trans. Power Syst. **21**(3), 1371–1378 (2006)
3. Cerisola, S., Baillo, A., Fernandez-Lopez, J.M., Ramos, A., Gollmer, R.: Stochastic power generation unit commitment in electricity markets: a novel formulation and a comparison of solution methods. Oper. Res. **57**, 32–46 (2009)
4. Danna, E., Rothberg, E., Pape, C.L.: Exploring relaxation induced neighborhoods to improve MIP solutions. Math. Program. **102**(1), 71–90 (2005)
5. eTerraMarket. http://www.alstom.com. Accessed Apr 2015 (2014)
6. Feng, Y., Rios, I., Ryan, S.M., Spurkel, K., Watson, J.-P., Wets, R.J.-B., Woodruff, D.L.: Toward scalable stochastic unit commitment. Part 1: load scenario generation. Energy Syst. (2015). doi:10.1007/s12667-015-0146-8
7. Gade, D., Hackebeil, G., Ryan, S.M., Watson, J.-P., Wets, R.J.-B., Woodruff, D.L.: Obtaining lower bounds from the progressive hedging algorithm for stochastic mixed-integer programs. (2015). http://www.optimization-online.org/DB_HTML/2015/01/4728.html
8. Goez, J., Luedtke, J., Rajan, D., Kalagnanam, J.: Stochastic unit commitment problem. Tech. rep., IBM (2008)
9. Hart, W., Watson, J., Woodruff, D.: Pyomo: Modeling and solving mathematical programs in Python. Math. Program. Comput. **3**(3), 219–260 (2011)
10. Morales-Espana, G., Latorre, J.M., Ramos, A.: Tight and compact MILP formulation for the thermal unit commitment problem. IEEE Trans. Power Syst. **21**, 4897–4908 (2013)
11. Oren, S., Papavasiliou, A., O'Neil, R.: Reserve requirements for wind power integration: a scenario-based stochastic programming framework. IEEE Trans. Power Syst. **26**(4), 2197–2206 (2011)
12. Oren, S., Papavasiliou, A., O'Neil, R.: Multi-area stochasti c unit commitment for high wind penetration in a transmission-constrained network. Oper. Res. **61**(3) (2013)
13. Ostrowski, J., Anjos, M., Vanneli, A.: Tight mixed integer linear programming formulations for the unit commitment problem. IEEE Trans. Power Syst. **27**(1), 39–46 (2012)
14. Papavasiliou, A.: Coupling renewable energy supply with deferrable demand. Ph.D. thesis, University of California Berkeley (2011)
15. Papavasiliou, A., Oren, S.: A stochastic unit commitment model for integrating renewable supply and demand response. In: Proceedings of the 2012 IEEE Power and Energy Society Meeting (2012)
16. Price, J.: Reduced network modeling of WECC as a market design prototype. In: Proceedings of the 2011 IEEE Power and Energy Society General Meeting (2011)
17. Rockafellar, R., Wets, R.J.B.: Scenarios and policy aggregation in optimization under uncertainty. In: Mathematics of Operations Research, pp. 119–147 (1991)

18. Ruiz, P., Philbrick, C., Sauer, P.: Modeling approaches for computational cost reduction in stochastic unit commitment formulations. IEEE Trans. Power Syst. **25**(1), 588–589 (2010)
19. Ruiz, P., Philbrick, R., Zack, E., Cheung, K., Sauer, P.: Uncertainty management in the unit commitment problem. IEEE Trans. Power Syst. **24**(2), 642–651 (2009)
20. Shapiro, A., Dentcheva, D., Ruszczyński, A.: Lectures on stochastic programming: modeling and theory. Society for Industrial and Applied Mathematics (SIAM), Philadelphia (2009)
21. Siface, D., Vespucci, M., Gelmini, A.: Solution of the mixed integer large scale unit commitment problem by means of a continuous stochastic linear programming model. Energy Syst. **5**(2), 269–284 (2014). doi:10.1007/s12667-013-0107-z
22. Slyke, R.V., Wets, R.: L-shaped linear programs with applications to optimal control and stochastic programming. SIAM J. Appl. Math. **17**, 638–663 (1969)
23. Takriti, S., Birge, J., Long, E.: A stochastic model for the unit commitment problem. IEEE Trans. Power Syst. **11**(3), 1497–1508 (1996)
24. Watson, J., Woodruff, D.: Progressive hedging innovations for a class of stochastic mixed-integer resource allocation problems. Comput. Manag. Sci. **8**(4), 355–370 (2011)
25. Watson, J.P., Woodruff, D., Hart, W.: PySp: modeling and solving stochastic programs in Python. Math. Program. Comput. **4**(2), 109–149 (2012)
26. Zheng, Q., Wang, J., Pardalos, P., Guan, Y.: A decomposition approach to the two-stage stochastic unit commitment problem. Ann. Oper. Res. **210**(4), 387–410 (2013)