Bröchin *et al.*
Energy, Sustainability and Society (2024) 14:27
https://doi.org/10.1186/s13705-024-00458-z

Energy, Sustainability and Society

RESEARCH

Open Access

Harder, better, faster, stronger: understanding and improving the tractability of large energy system models

Manuel Bröchin[1], Bryn Pickering[2], Tim Tröndle[1] and Stefan Pfenninger[3*]

Abstract

Background Energy system models based on linear programming have been growing in size with the increasing need to model renewables with high spatial and temporal detail. Larger models lead to high computational requirements. Furthermore, seemingly small changes in a model can lead to drastic differences in runtime. Here, we investigate measures to address this issue.

Results We review the mathematical structure of a typical energy system model, and discuss issues of sparsity, degeneracy and large numerical range. We introduce and test a method to automatically scale models to improve numerical range. We test this method as well as tweaks to model formulation and solver preferences, finding that adjustments can have a substantial impact on runtime. In particular, the barrier method without crossover can be very fast, but affects the structure of the resulting optimal solution.

Conclusions We conclude with a range of recommendations for energy system modellers: first, on large and difficult models, manually select the barrier method or barrier+crossover method. Second, use appropriate units that minimize the model's numerical range or apply an automatic scaling procedure like the one we introduce here to derive them automatically. Third, be wary of model formulations with cost-free technologies and dummy costs, as those can dramatically worsen the numerical properties of the model. Finally, as a last resort, know the basic solver tolerance settings for your chosen solver and adjust them if necessary.

Keywords Energy system models, Scaling, Linear programming, Numerical issues, Interior-point, Simplex, Benchmark

Background

Introduction

Mathematical optimisation, in particular linear programming (LP), has become one of the key methods in established and emerging energy system modelling tools used for planning the energy transition and assessing energy and climate policy options around the world [5, 6, 31, 34]. Renewable energy technologies need to be represented with a high spatio-temporal resolution and scope, to capture and account for the effect of their intermittency on system stability, to ensure that weather variability can be captured across years [37], and to exploit the balancing effect of geographically distant weather systems [12]. This has led to the development of ever larger models [15, 42, 45]. Model size increases further when using Monte-Carlo or scenario-based methods to deal with structural and parametric uncertainty [7, 10, 20, 29]. The extent to which ever more complex models give us additional insight is a discussion on its own, but the reality is that

*Correspondence:
Stefan Pfenninger
s.pfenninger@tudelft.nl
[1] Department of Environmental Systems Science, Institute for Environmental Decisions, ETH Zürich, Switzerland
[2] Girton College, University of Cambridge, Cambridge, United Kingdom
[3] Faculty of Technology, Policy and Management (TPM), Delft University of Technology, Delft, The Netherlands

many large and complex optimisation models are used in practice, hence improving their tractability is of immediate concern.

There is generally a limit to how much we can scale-up a model before combinatorial explosion makes the model intractable. We say a computational model, such as an energy system model, is tractable if it can be simulated or optimized within an appropriate amount of time. How much time is appropriate for a given problem is of course a subjective judgment. For the energy system models considered in this work, we consider runtimes of around 5 to 10 h to be appropriate. We refer to the property of how much a model can be scaled-up before it becomes intractable as the tractability of the model. We say some measure improves tractability of a model if it allows us to solve larger instances of the model in an appropriate amount of time.

Reference [28] identified tractability as one key challenge for energy system optimisation models. Most often, to improve tractability of energy system models, temporal resolution is reduced [4, 23, 42] or time periods with similar features are clustered and represented only by a subset of 'typical days' [2, 24, 29]. Advancements in the field of time-series aggregation include pre-selection of 'critical' days prior to clustering [26], identification of k-mediods as the most reliable aggregation algorithm [18, 36], and separation of storage into multiple decision variables, to allow information to be linked within and between clustered periods [9, 19]. [14] summarised these methods, finding that existing feature aggregation using k-means, k-mediods, or hierarchical clustering is still the 'state-of-the-art'. However, they reiterated warnings made in several previous studies that time-series aggregation alters model results, both qualitatively and quantitatively, and should therefore be used with caution [18, 26]. Recent work [35] tries to improve tractability of energy system models on the algorithmic layer: they develop a parallel algorithm that exploits the inherent structure of energy system models to break up the problem into weakly related subproblems that can then be solved individually.

Similar to [35], we take a step back from application-specific methods to reduce model complexity. Instead, we examine how the structure of the underlying optimisation problem affects its tractability, and empirically examine a range of options to improve tractability for typical energy system models. We also develop and test a method to facilitate tractability, in which we automatically scale the parameters of an energy system model. To do so, we draw on literature from the broader field of operations research [8, 16, 40]. Recently, Göke introduced AnyMOD.jl [11], a Julia package for creating energy system models. AnyMOD.jl includes a scaling procedure similar to ours. Configurable factors are used for scaling each "type" of variable. These factors have default values but can also be set by the user. The major difference to our work is that we automatically derive optimal scaling factors for variables depending on the actual model data, i.e. the types of variables occurring in the model and the numerical ranges of these types.

The paper proceeds as follows. First, we discuss the mathematical properties of a typical high-resolution energy system model and compare the characteristics of the two main solution methods: simplex and barrier (interior point). We identify scaling of the optimisation problem as a particular area of concern for the performance of these methods and introduce a method to automatically scale an energy system model. We then proceed with a series of systematic experiments to answer the following questions: Which algorithm solves our models in the least amount of time? In what ways differ the solutions provided by the different algorithms? How does our automatic scaling method impact solution time? We then discuss what general guidelines for energy system modelling we can draw from these experiments. We use the open-source Calliope modelling tool [27] for our experiments.

Overview of the mathematical problem

A high-level view of the problem we want to solve is the following: As input we are given a set of locations together with their demand and supply of certain goods and their capacity limits for certain technologies. The supply of a location often consists of timeseries data of certain natural resources such as sunlight or wind. The demand is often for carriers such as electricity or gas and is also given as timeseries data. The capacity limits of a location constrain the maximum capacity of a certain technology, such as solar power, that can be allocated or built at this location. The locations are connected by a network of transmission lines. The modeller is free to fix the capacity of certain technologies or transmission lines in advance or not. The degrees-of-freedom of the model are then the capacity expansion and the operation of technologies. More concretely, as output to our model we get the optimal allocation of capacity per technology and location. Moreover, we get for each timestep the actual production of carriers per location and technology plus their transmission between locations. Optimality of capacity expansion and operation is determined in terms of user-defined operation and allocation costs and is subject to a variety of additional constraints. In particular, the modeller can impose policies on carrier consumption and production such as minimum shares of renewable energy.

A standard way of modelling this type of problem is as a linear program (LP). Many energy system models additionally feature discrete decisions, e.g., should a new power plant be built or not. Modelling discrete decisions requires forcing certain decision variables to take integer values in the solution. Ordinary LPs cannot express such constraints and a mixed-integer LP (MILP) formulation must be chosen instead. The main focus of this work is on understanding and improving the tractability of LPs. However, since MILP solvers need to solve the "relaxed" LP-version of their input problem repeatedly as a sub-problem, our findings also apply to energy system models that are framed as MILPs.

In the following, we provide some details on the structure of LPs. Moreover, we give an overview over the two main classes of algorithms used to solve LPs. The goal is to understand their major differences in order to be able to trade-off their respective benefits. Moreover, we will understand the factors that influence the performance of these algorithms which allows us to better interpret the ensuing experiments and which provides motivation for our scaling approach.

LP problems and solution methods

LPs are a very general framework of mathematical optimization [43]. The general form of an LP is

$$\begin{aligned} &\text{minimize } c^\intercal x \\ &\quad \text{s.t. } Ax = b \\ &\qquad 0 \leq x \end{aligned} \tag{1}$$

The linear objective function $x \longmapsto c^\intercal x$ is to be minimized by choosing suitable variable values $x \in \mathbb{R}^n_+$. The constraint matrix $A \in \mathbb{R}^{m \times n}$ and right-hand side vector $b \in \mathbb{R}^m$ together constrain the choice of variables x with m linear constraints. A positive vector $x \in \mathbb{R}^n_+$ is called feasible if it satisfies all m constraints. A set of m independent columns of A induces a basis A_B of the column space of A. A feasible point x can be obtained by solving $Ax = b$ while setting all variables not corresponding to basis columns to 0. We call such a solution x a basic feasible solution of Eq. 1. Geometrically, a basic feasible solution is an extreme point (a vertex) of the feasible region.

Formulating an energy system in analogy to cost-minimal flow problems as an LP is straightforward: x_i is the variable holding the amount of flow through the ith edge. Row j of A contains a non-zero entry at index i which is 1 or -1 if the i-th edge is an incoming or outgoing edge of node j, respectively. b_j is the flow-demand of node j. c_i is the cost incurred by one unit of flow at edge i. In addition, edge capacities and many constraints that we find in energy system models, such as efficiencies of carrier conversion or complex policies governing the operation of certain plants, can either be expressed directly or approximated by linear constraints.

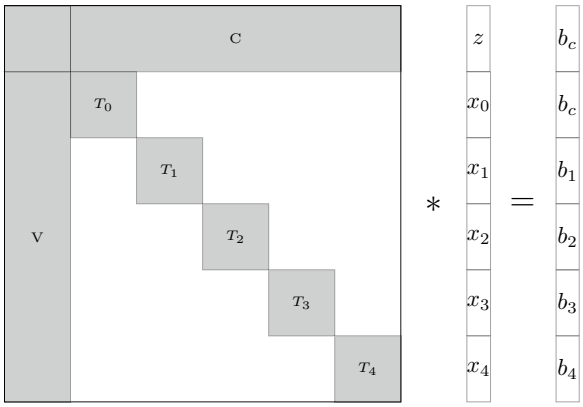

Fig. 1 $A \cdot x = b$ in block-matrix notation showing the typical sparsity pattern of the constraint matrix A of an energy system model. Variables x_i are specific to timestep i and are constrained using parameters τ_i and b_i. $cx = b_c$ are constraints containing dependencies across time steps. Parameters v and variables z model time-independent quantities

As discussed above, energy system models consider the allocation of resources in time and space. Modelling time requires its discretisation into timesteps, which leads to a block-diagonal matrix structure. Moreover, there are often variables and constraints spanning many or all time steps. For instance, modelling the decision of capacity allocation to plants require on variable per capacity that spans all time steps. Similarly, enforcing global policies, such as a minimum share of renewable energy in the system, requires constraints that span all time steps. As a consequence, the constraint matrix A has a very particular structure, depicted in Fig. 1. This structure is often referred to as arrowhead structure. Specialized LP solution methods which try to exploit this structure to speed up the solution process are in development [32].

At the moment, the two families of algorithms most widely used to solve energy system models are the simplex algorithms and barrier or interior-point methods, of which many variants exist. These two approaches are available both in open-source software packages such as Coin-OR [21] or commercial ones such as Gurobi [13]. In the following we sketch some important characteristics of both simplex and barrier methods based on the treatment in [16]. Table 1 lists the main points for both methods side-by-side.[1]

Although the two algorithms solve the same problem, they operate in entirely different ways. (Primal) Simplex

[1] Note that primal affine scaling is not a state-of-the-art implementation of the barrier method. Like Klotz in [16], we describe this method because it is simple and allows to illustrate the major ideas of barrier methods.

Table 1 Comparison of major characteristics of simplex and barrier methods following [16]

	Simplex methods (primal simplex)	Barrier methods (primal affine scaling)
State	Maintains a basis of A and a corresponding basic feasible solution	Maintains a non-basic (interior) point x satisfying $Ax = b$
Update	Performs row operations on A to update the basis in each step (pivot operation)	Computes a vector along which the objective is improved, projects it onto the nullspace of A and updates x along this vector while maintaining $x > 0$
Main computation	Main computational cost is solving $m \times m$ linear systems of equations for the pivot step	Main computational cost is projecting the update vector onto the nullspace of A. This entails inverting the $m \times m$ matrix $\hat{A}\hat{A}^{\intercal}$ in every step where $\hat{A}$ is a scaled version of A.
Termination condition	Terminates if no basis update can improve the objective value over the current basic feasible solution	Terminates if the last update step was shorter than some threshold
Return value	Returns a basic feasible solution	Returns an interior point that is close to the boundary of the feasible region (we will denote this an interior solution). A crossover method can be added to retrieve an optimal basic feasible solution from an interior solution

maintains and updates a basic feasible solution x_i in every step i of the algorithm. To find x_{i+1}, simplex chooses one of the neighboring vertices of x_i that improves the objective value. This jump from x_i to x_{i+1} is performed algebraically by exchanging a column in the current basis of A. Convexity of the feasible region and linearity of the objective function imply that an x_i which cannot be improved in this way is optimal. An important variant of the simplex method is the dual simplex method. The dual simplex method starts out with an optimal basic solution (which need not be feasible). It then performs updates to the basic solution that reduce infeasibilities while maintaining optimality.

Barrier maintains an interior point x of the feasible region. In every step of the algorithm, barrier finds a vector along which x can be improved with respect to c, and takes a step along this direction without leaving the feasible region. With x becoming closer to optimal, x will also move towards the boundary of the feasible region until some implementation-dependent termination criterion is reached.

Simplex always returns a basic feasible solution while barrier in general returns an interior point of the feasible region. This has several implications:

First, an interior point is, strictly speaking, not an optimal solution to an LP. However, the "gap" can in theory be made arbitrarily small. Moreover, optimality of a basic feasible solution can also only be determined with a certain limited precision. Thus in practice, solvers typically allow controlling the precision of both methods. Neither "optimal" solutions returned by simplex nor by barrier are generally better, but the precision of the solution depends on the solver configuration.

Second, there are cases where a basic feasible solution is needed: (1) when solving a MILP problem, and (2) if the obtained solution is to be used to warm-start subsequent solver runs on the same (or a slightly different) model.

Third, an interior solution has potentially many more variables away from their bounds than a basic solution. Since variables representing real-life quantities often have a lower bound of 0, this means that solutions returned by barrier usually have many more non-zeros than basic feasible solutions; potentially these non-zeros are unrealistically small for the considered problem. A solution with many non-zeros may be harder to interpret and may thus be undesirable.

To rectify these downsides of interior solutions, a dedicated crossover method can be used to find an optimal basic feasible solution from an optimal interior solution. To do this, a vertex 'close' to the interior point solution is found by pushing certain variables to their bounds. Starting from this vertex, simplex steps are applied until an optimal basic solution is found [3].

Neither of the two methods is generally faster than the other. Instead, the most suitable algorithm depends on the structure of the LP at hand [16, 41]. Two LPs of the same size can lead to vastly different solution times for both methods, simplex and barrier. This is mostly due to two reasons: numerical solvers can recognize and exploit problems with special structures and they are susceptible to numerical problems. Three aspects are particularly influential: sparsity (structure), degeneracy (structure), and the numerical range.

The first aspect impacting the solution time is sparsity. Energy system models typically lead to LPs that are highly sparse. This follows directly from the arrowhead structure depicted in Fig. 1: the number of zero entries scales quadratically in the number of time steps, whereas the number of nonzero entries is linear in the number of time steps. A typical model might consider a full year at a granularity of 1 h time steps, causing most entries of the

matrix to be zero. In general, sparsity is a desirable property of a linear program because both simplex and barrier solvers have ways of profiting from sparsity in the constraint matrix A.

The second aspect that influences solution time is degeneracy. Degeneracy in the context of linear programming means that multiple bases of the constraint matrix A lead to the same basic feasible solution. Standard minimum-cost flow problems are often inherently degenerate [1, 41]. Degeneracy can pose significant problems to the simplex method: because multiple bases lead to the same solution, simplex updates may fail to improve the objective and stall progress. Computational studies have shown that up to 90% of all pivot operations of the simplex method on min-cost flow problems can be degenerate [1]. Barrier methods maintain an interior point of the feasible region and never actually encounter a basic feasible solution. Degeneracy thus matters less to the operation of barrier methods [16].

The third and final aspect is a large numerical range of problems, given by the absolute values of coefficients in the constraint matrix, the right-hand side and the objective function. Energy system models often have a very large numerical range. This is a consequence of two facts: first, the input data of energy system models correspond to different physical quantities (e.g., energy, area, cost). Choosing inappropriate combinations of units to represent these quantities will lead to large numerical ranges. Second, even within quantities of the same unit we might encounter large numerical ranges, for instance, between operating costs and investment costs for new generation infrastructure, or even just between the operating costs of very different technologies like photovoltaics and gas-fired power generation. A large numerical range can result in increased solution times and even lead to non-convergence in extreme cases. It can furthermore lead to loss of precision due to round-off errors. To address these issues, it is possible to scale the linear problem before solving it.

Scaling

Choosing positive scaling factors $r_1, \dots, r_m, s_1, \dots, s_n \in \mathbb{R}_+$ we can scale each row i of A by r_i and each column j of A by s_j by multiplying A from left and right with the diagonal matrices $R := \operatorname{diag}(r_1, \dots, r_m)$ and $S := \operatorname{diag}(s_1, \dots, s_n)$.

$$\min\{c^\intercal x \mid x \geq 0, Ax = b\} \tag{2}$$

$$\min\{c^\intercal Sx \mid x \geq 0, RASx = Rb\} \tag{3}$$

Perhaps, unintuitively, the original problem 2 is equivalent to problem 3 which is scaled with R and S. A short proof for this is in the Appendix A. While being equivalent, problem 2 may have better numerical properties than the original LP. In the following we consider the numerical range and the condition number of A as two factors that decide whether an LP has good or bad numerical properties.

The numerical range $\frac{\max_{x \in A,b,c} |x|}{\min_{x \in A,b,c} |x|}$ of an LP impacts the performance of the solution methods. The reason for this is that most solvers use absolute tolerances to compare numbers. For instance, a central parameter of the Gurobi solver is the feasibility tolerance τ_f, set by default to $\tau_f = 10^{-6}$. To check if a point x satisfies some constraint $A_i x \leq b_i$, Gurobi checks whether $A_i x - b_i \leq \tau_f$. If such computations are at the order of 10^{10}, then the relative error of 10^{-16} induced by floating point arithmetic is at the order of the tolerance τ_f. In other words, feasibility can no longer be reliably computed. Similarly, if some computation is at the order of 10^{-6}, the result will likely be meaningless to Gurobi. If our model now includes both very large and very small numbers, it is likely that both types of problems appear during the solution process. It is thus desirable to limit the numerical range in our model formulation. Note that this applies equally to both simplex and barrier methods.

A different notion is the condition number $\kappa(M) := \|M\|\|M^{-1}\|$ of a matrix M. Intuitively, the condition number of a matrix measures the effect of small rounding errors when solving the linear system of equations associated with the matrix: for a matrix with small condition number, the rounding errors will barely impact the solution. For a matrix with large condition number, the effect of the rounding error can be significant. Note that both simplex and barrier rely heavily on solving linear systems of equations. In the case of simplex, these matrices are the square submatrices of A corresponding to the visited bases of A. In the case of barrier, the important matrices are $\hat{A}\hat{A}^\intercal$ that are inverted at each step (see Table 1).

Both factors, the numerical range and the condition number thus impact the accuracy of the computations performed during both simplex and barrier and both are affected by scaling. The relationship between numerical range and condition number, however, is complicated: improving one may make the other one worse. We elaborate more on this in Appendix B. The accuracy of computations impacts the performance of these algorithms in multiple ways. First, these algorithms make decisions based on numerical computations. For instance, simplex chooses the next basic feasible solution based on some metric which is computed numerically. Large errors in the computation of this metric may cause simplex to make a suboptimal choice and increase the number of steps necessary

to reach optimality. Second, to avoid the previous issue, solvers have ways to detect and deal with loss of precision. One way is switching to a numeric data type with higher precision. These types incur a higher computational cost for each operation and thus cause overall slowdown of the algorithm (a secondary impact would additionally be the higher memory cost of higher-precision data types, which in itself can be significant for very large models).

An automated scaling method

Both numerical range and condition number of a matrix are affected by scaling rows and columns of the matrix. While reducing numerical range with scaling is straightforward, reducing the condition number of all relevant matrices that are encountered during simplex and barrier is not. In the case of simplex [8], compares many different scaling methods and their effect on the condition number. For certain problems, scaling can lead to an increase of the condition number averaged over all basis matrices encountered during a simplex run.

We have developed an automatic scaling method that focuses on minimizing the numerical range of the input data, which ignores the issue of the condition number (we refer to this method as "autoscaling" from here one). The basic idea is to group input values by type (such as area, cost, energy, etc.) and scale all values in one group with the same factor. The assumption is that values of the same type will often be contained within an acceptable range. This approach corresponds to choosing suitable units (such as m^2, dollars, kWh, etc.) for each type of quantity. Next, we discuss in more detail how we implemented this intuitive approach.

Most variables in Calliope are associated with some physical quantity such as energy or costs. We call this the type of the quantity. Values of different types can be of vastly different orders of magnitude. This effect is often a side effect representing certain types of quantities (cost, energy, etc.) with standard units (dollars, kWh, etc.), regardless of the range of their values. Note that changing the unit of a type of quantity is a special case of the scaling considered in Sect. "Scaling" with $R = I$ and $S = \mathbf{diag}(f_{u_i})$ where u_i for $i \in [n]$ is the unit of the i-th decision variable.

In the context of Calliope, it is necessary to distinguish between base units and derived units. The sets U of base units and V of derived units are related as follows:

$$u \in U \implies u \in V, \quad u \in U \implies \frac{1}{u} \in V, \quad u_1, u_2 \in U \implies \frac{u_1}{u_2} \in V \tag{4}$$

This distinction is necessary because Calliope models often feature combinations of base units such as cost per energy. Scaling energy by a factor s_e and cost by a factor s_c we necessarily need to scale cost per energy by the factor $\frac{s_c}{s_e}$ to avoid changing the semantics of the model. Associating a special base unit $\mathbf{1} \in U$ with all quantities that do not correspond to a physical quantity allows us to represent all derived units $v \in V$ of Calliope models as a fraction of base units $v = \frac{u_i}{u_j}$. Let A_u the set of values in the model that have unit $u \in V$. Our goal is to find scaling factors f_u for each unit $u \in V$ that minimize

$$\kappa' := \frac{\max_{u \in U} \max_{a \in A_u} f_u \cdot a}{\min_{v \in U} \min_{b \in A_v} f_v \cdot b} \tag{5}$$

There are, however, two additional subtleties we need to consider. First, scaling a floating-point number with a power of two will be exact whereas scaling it with a different number will often lead to a loss of precision. Such a loss of precision means that the semantics of the scaled model is not equivalent to the semantics of the unscaled model and should thus be avoided (see Appendix C). Second, minimizing the above quantity may lead to A containing values of prohibitively large or small values. As discussed above, absolute values matter to the solver and they should be kept within sensible ranges (see Appendix C). One way to avoid very large or small values in A is to additionally perform a scaling of the rows after having scaled the columns. However, we choose to instead limit the allowable range of values of A after column scaling and thus to constrain the choice of scaling factors.

We now formulate an auxiliary optimization problem for finding scaling factors f_u taking into account the two constraints from above: We wish to find scaling factors $S = \{f_u \mid u \in U\} \subseteq \{2^x | x \in \mathbb{Z}\}$ that are powers of two and that minimize the numeric range between different types of variables while avoiding to scale any variable below some threshold L. Note that all variables of some column A_i of constraint matrix A have the unit of the i-th decision variable. Thus denote by $u_i, v_i \in U$ for each $i \in [n]$ the base units satisfying that all variables in column i of A have unit $\frac{u_i}{v_i}$. We thus wish to solve

$$\min_{S \subseteq \{2^x | x \in \mathbb{Z}\}} \max_{\substack{0 \le i,j \le m \\ 0 \le k,l \le n}} \frac{\frac{f_{u_k}}{f_{v_k}} A_{i,k}}{\frac{f_{u_l}}{f_{v_l}} A_{j,l}} \tag{6}$$

$$\text{s.t.} \ \frac{f_{u_k}}{f_{v_k}} \cdot A_{i,k} \ge L \qquad \forall k \in \{0, \dots, n\}, \forall i \in \{0, \dots, m\} \tag{7}$$

We can rephrase this into an integer LP by taking logs of the entries of A and of the scaling factors $f \in S$. That is, we define $f_u := 2^{g_u}$ for each $u \in U$, and constrain $g_u \in \mathbb{Z}$ to ensure that all scaling factors are powers of two.

$$\begin{aligned} \min \quad & r \\ \text{s.t. } & g_{u_k} - g_{v_k} + \log(A_{i,k}) - g_{u_l} + g_{v_l} - \log(A_{j,l}) \leq r \qquad \forall\, i,j,k,l \end{aligned} \tag{8}$$

$$g_{u_k} - g_{v_k} \geq \log\left(\frac{L}{A_{i,k}}\right) \qquad \forall k \in \{0,\ldots,n\}, \forall i \in \{0,\ldots,m\} \tag{9}$$

$$g_u \in \mathbb{Z} \qquad \forall u \in U \tag{10}$$

Note that we don't actually need to consider all entries of A. It suffices to include the minimum and the maximum of value of each unit. This simplification considerably reduces the amount of constraints from n^2m^2 to $|U|^2$ which is usually very small (below 100). Moreover, instead of solving the actual integer LP, we can relax the integrality constraint of g_u and round the resulting variables to the closest integer. This will give a 4-approximation of the optimal scaling factors. In practice we find that the integer LP is solved quite rapidly, thus we keep the integrality constraint in place.

Methods

We consider models generated by the Calliope modelling framework [27] and want to investigate (1) which algorithm solves these problems in the least amount of time, (2) what the trade-offs when choosing between a basic feasible solution and an interior solution are, and (3) what the impact of our scaling approach on the solution time is. The models used for the experiments are specified in Table 2.

All models are implemented with the Calliope modelling framework. We will often add a postfix to model names to indicate the time range, e.g., Euro_15d and Euro_5m refer to the Euro-Calliope model run over a 15-day and a 5-month time range, respectively, starting from January 1, always at the time step resolution of 1 h. Note that the Bangalore model is set in a leap year while Euro and UK are not. Because we set the time ranges by date, the selected time range in the Bangalore model is often one day longer than in the other two models.

Table 2 Calliope models used in experiments

Name	Model	Citation
Euro	34-zone LP Euro-Calliope model[a]	[42]
UK	30-zone LP UK-Calliope[b]	[26]
Bangalore	10-zone MILP Bangalore-Calliope but without integrality constraints[c]	[30]

Note that the Euro model by default includes manual scaling to improve performance; for the experiments here we undo this manual scaling. Also, the Bangalore model by default contains MILP constraints in the form of purchasable technologies. We have removed these constraints, making these technologies available without up-front purchase costs

[a] https://github.com/calliope-project/euro-calliope

[b] https://github.com/calliope-project/uk-calliope

[c] https://github.com/brynpickering/bangalore-calliope

We run all benchmarks on the Euler cluster of ETH Zurich. All experiments where solution times are measured are performed on the same setup: compute nodes with two 18-core Intel Xeon Gold 6150 processors (2.7–3.7 GHz) with 192 GB of DDR4 memory clocked at 2666 MHz. Each experiment was run alone on a full node, i.e. taking up all 36 cores, to prevent competing processes from other cluster users from affecting the solution time. Note, however, that the number of cores does not correspond to the number of software threads. Except where explicitly noted we set the number of software threads for the solvers to 4. We run Gurobi Optimizer version 9.0.0 build v9.0.0rc2 (linux64). Unless otherwise noted, we use the default configurations of Gurobi except for setting barConvTol to 10^{-6}, feasibilityTol to 10^{-5} and optimalityTol to 10^{-5}.

The authors of [17] observe that the performance of MILP solvers is non-deterministic and is often subject to large variation due to seemingly unimportant changes: different machines, compilers, and libraries can all lead to vastly different performance. We observe the same behaviour for Gurobi's LP solvers, mostly for the crossover phase (including final simplex cleanup). In order to account for this variability in our benchmarks, we perform each measurement several times only changing the random seed value of Gurobi. For each benchmark we indicate the number of repetitions, the average and min/max solution times.

Solver choice We compare the suitability of the commercial Gurobi solver and the open-source Coin-OR Clp solver for our problems. We compare both simplex and barrier methods of these two solvers. Comparing primal simplex, dual simplex and the barrier method covers the most common and most successful algorithms for solving linear programs. The scientific consensus is that no algorithm clearly dominates the others, instead, which algorithm works best depends on the structure of the problem to be solved. With this comparison we thus want to answer the question of which algorithms works best for energy system models created by the Calliope framework.

On the other hand, there are many software packages other than Gurobi and Coin-OR that implement these algorithms. Instead of comparing all software packages we chose two examples that are commonly used, one commercial and one open-source. Other authors have performed more thorough solver comparison on a wide class of problems.[2] We note that both Gurobi and Clp

[2] http://plato.asu.edu/guide.html.

score high in these comparisons but that there is generally a difference in solver capabilities between commercial and open-source implementations. Thus, the second question we want to answer with this comparison is whether we benefit from using a commercial solver or if a free implementation suffices for our problems.

We compare the runtime of each of these solvers on a range of models. In particular, we create instances of different sizes of the models in Table 2 varying the time range between two days and six months. The shortest time frames are of little practical relevance but are included here to give a more complete picture of solver capabilities. We aim at configuring each solver and algorithm equivalently despite the inherent differences in configurability between Coin-OR and Gurobi's solvers: in particular, we instruct each solver to use 4 threads and we set optimality and feasibility tolerances to 10^{-5} (changing from the 10^{-6} default in Gurobi).

Interior vs. basic solution As discussed in Sect. "LP problems and solution methods", the barrier method run by itself returns an optimal interior point solution, whereas the simplex method returns an optimal basic solution. When using the barrier method, it is possible to run the crossover method to find a basic feasible solution, taking the interior solution as the initial value. We consider up- and downsides of using an interior solution as compared to obtaining a basic solution.

We consider two questions: (1) how quickly we can obtain both interior and basic feasible solutions, and (2) how "good" is either solution. To answer these questions we create a sequence of different models, solve them with both barrier and barrier+crossover and then compare the interior solution returned by barrier with the basic feasible solution returned by barrier+crossover. The models considered are all slight variations of the models listed in Table 2. The variations were obtained by varying costs, variables and constraints in several ways. The primary goal of the variations was not necessarily to create faithful models of reality. Instead, we wanted to create a larger set of linear programs that structurally still correspond to energy system models defined by Calliope but which differ in their solution space (the shape of the polytope) and their cost vector from the original models. The concrete modifications performed are described in Appendix E. To answer questions (1) and (2) from above, we compare the interior and the basic feasible solutions with respect to the objective value obtained, the time it took to find the solution and the fraction of non-zero values in the solution.

When interpreting the solution to a model we are mostly interested in the following per:

- `capacity`: indicates for each location and technology the installed capacity (from now on denoted *cap*).
- `carrier_production`: indicates for each time step, location and technology, how much of a given carrier is produced (from now on denoted *prod*).
- `systemwide_levelised_cost`: indicates the total per-unit cost paid for each carrier across the entire energy system (from now on denoted *lcoe*).

Scaling As discussed in Sects. "LP problems and solution methods" and "Scaling", energy system models often have large numerical ranges and it is desirable to apply scaling to avoid numerical issues. To diagnose numerical issues in our models we closely analyze the logs produced by the solver. This methodology is showcased for instance in [16]. The models we examine are often numerically problematic, and we investigate whether our autoscaling method as developed in Sect. C can effectively mitigate these problems. To do so, we benchmark the solution time of various models using Gurobi with and without autoscaling and quantify the effects of scaling. Note that Gurobi internally scales the input problem by default (parameter ScaleFlag = −1). According to the Gurobi online forum,[3] the default scaling option chooses between equilibration and geometric scaling. We did not change this setting, meaning that we compare our autoscaling to the scaling that Gurobi performs internally. Apart from scaling, Gurobi features some parameters for tuning its algorithms for numerically difficult problems. In particular, the NumericFocus parameter will cause Gurobi to spend more time detecting and reacting to numerical issues while the Quad parameter will use a larger numeric type for representing all numbers in the problem. We left these parameters at their defaults in all experiments.

Results

Solver choice

In this comparison, we use the primal simplex, dual simplex, and barrier method with and without crossover of Gurobi and Coin-OR solvers to solve energy system models of varying sizes. Table 3 lists the model instances and the runtime of each of the solution methods in seconds. We group our observations in the following paragraphs roughly by solver.

Coin-OR barrier As an important caveat we must note that the barrier implementation of Coin-OR is meant as a baseline implementation which the user is supposed to extend with problem specific implementations. In particular, it is stated on the Clp homepage[4]

[3] https://support.gurobi.com/hc/en-us/community/posts/15330309039633-How-does-the-ScaleFlag-option-work-.

[4] https://www.coin-or.org/Clp/faq.html.

that "the sparse factorization [of barrier] requires a good ordering algorithm, which the user is expected to provide (perhaps a better factorization code as well)." We nevertheless use this baseline implementation for our comparison, and perhaps unsurprisingly, Coin-OR's default barrier algorithm struggles with all but the smallest problems (Table 3). After a certain size Coin-OR fails to perform a single barrier iteration and either times out (error condition (2)) or chooses to run the simplex method instead of barrier (error condition (1)). Also for small model sizes, barrier of Coin-OR is not competitive with any of the other algorithms.

Primal and dual simplex Primal and dual simplex implementations of Gurobi and Coin-OR each seem to be comparable in their capabilities but Gurobi is generally slightly better (Table 3). Both dual simplex implementations solve almost the same set of problem instances, each with comparable solution times. Similarly, both primal simplex implementations solve almost the same set of problems with comparable solution times. In general, dual simplex seems to be at least as good as primal simplex (with some exceptions for Gurobi, for instance UK_181d).

Gurobi Barrier methods Gurobi's barrier method with and without crossover solves all problem instances and more instances than all other solvers (Table 3). Except for the smallest problem instances, barrier methods have the shortest solution times. Turning on crossover has an unpredictable effect on runtims: in some cases solution time is barely affected (e.g., UK_181d), in other cases the solution time triples (e.g., Bangalore_182d and Euro_181d).

Summarizing the above, we find that the difference in performance between commercial and free software package are quite small for the primal and dual simplex method but they are great for the barrier method. The barrier implementation of Gurobi turns out to be the most effective at solving our problem instances, while the barrier implementation of Coin-OR is the least effective. In the following, we will focus on the Gurobi software package.

Table 3 Solution times in seconds of different solvers on different model instances

Model	Coin-OR BC				Gurobi			
	Bar	Crossover	Dual	Primal	Bar	Crossover	Dual	Primal
Bangalore_2d	52	54	10	9	9	10	8	9
Bangalore_3d	579	570	14	12	11	13	11	14
Bangalore_5d	6017	6019	16	19	15	18	15	18
Bangalore_11d	–(2)	–(2)	31	64	30	31	28	49
Bangalore_31d	–(1)	–(1)	115	211	79	89	80	219
Bangalore_62d	–(1)	–(1)	328	300	164	234	225	784
Bangalore_182d	–(1)	–(1)	2727	2348	588	1509	1395	–(2)
Euro_2d	70	73	68	97	63	62	67	90
Euro_3d	128	131	80	153	73	72	79	158
Euro_5d	834	837	128	360	93	96	121	401
Euro_11d	–(2)	–(2)	493	2668	161	175	441	1547
Euro_31d	–(2)	–(2)	–(2)	–(2)	394	485	5819	–(2)
Euro_61d	–(1)	–(1)	–(2)	–(2)	771	1526	–(2)	–(2)
Euro_181d	–(1)	–(1)	–(2)	–(2)	2564	8026	–(2)	–(2)
UK_2d	121	121	19	20	22	21	22	21
UK_3d	475	469	25	25	28	28	28	29
UK_5d	4054	4056	39	39	42	42	43	43
UK_11d	–(2)	–(2)	86	89	83	85	91	87
UK_31d	–(2)	–(2)	342	445	229	230	273	245
UK_61d	–(3)	–(3)	1026	1622	447	451	708	615
UK_181d	–(3)	–(3)	–(2)	–(2)	1400	1414	4257	2773

Bar is barrier method, **Crossover** is barrier+crossover, **Primal** and **Dual** are primal and dual simplex, respectively. Each solution time is the average of two runs. Runs that did not terminate successfully within 4 h and with 72GB of RAM are indicated with –. In those cases, we indicate the returned error: (1) wrong algorithm, (2) timeout, (3) solver error. See the text for more detailed information on what these errors mean

Table 4 Comparison of objective value, solution time and fraction of non-zeros of basic feasible (basic) and interior (inter) solutions of various Calliope models using barrier+crossover (basic) and barrier only (inter), respectively

Model	Objective			Time		Non-zeros	
	Basic	Inter	ε	Basic	Inter	Basic	Inter
Euro[1]_180d	7.34e+10	7.34e+10	5e−07	4008	1869	0.27	0.62
Euro[2]_180d	6.06e+10	6.06e+10	1e−08	3860	2313	0.29	0.58
Euro[3]_180d	2.20e+10	2.20e+10	2e−10	65	63	0.42	0.6
Euro[4]_180d	2.19e+10	2.19e+10	5e−16	56	54	0.43	0.56
Euro[5]_180d	2.17e+10	2.17e+10	9e−11	57	52	0.43	0.6
Euro[6]_180d	2.20e+10	2.20e+10	4e−12	52	50	0.42	0.58
UK[1]_180d	1.24e+10	1.24e+10	3e−11	2038	2274	0.25	0.48
UK[2]_180d	1.70e+10	1.70e+10	6e−08	2788	2215	0.26	0.52
UK[3]_180d	1.27e+10	1.27e+10	2e−09	5081	4105	0.24	0.5
UK[4]_180d	1.29e+10	1.29e+10	3e−09	3399	2239	0.25	0.5
UK[5]_180d	1.98e+10	1.98e+10	1e−07	3519	2381	0.25	0.52
UK[6]_180d	3.11e+10	3.11e+10	2e−06	5347	3888	0.21	0.41
Bangalore[1]_181d	2.78e+08	2.78e+08	2e−08	3818	1360	0.33	0.84
Bangalore[2]_181d	1.64e+08	–	–	5108	–	0.24	–

ε is the relative error of the interior solution compared with the basic feasible solution. Solution time is in seconds. Non-zeros denotes the fraction of decision variables with absolute value $> 10^{-10}$

To crossover or not

In the previous section, we observed that the barrier method of Gurobi often solves models much faster than simplex methods do, but that the crossover step can have a significant impact on the total solution time. It is tempting to say that disabling crossover is therefore an easy way to dramatically improve solution times. However, time to solution alone is not sufficient to choose a method because the solutions returned by barrier and those returned by simplex or barrier+crossover are inherently different. In this section, we thus compare the interior solutions obtained by running barrier without crossover with the basic feasible solution obtained by running either the simplex method or barrier+crossover.

Table 4 compares solutions to various Calliope models that were obtained using barrier and barrier+crossover. Appendix E describes the differences between the models. For each model, we compare an interior with a basic feasible solution in terms of objective value, the time it took to get the solutions and the fraction on non-zero variables in the solutions. For comparing the objective value of interior and basic feasible solution, we additionally list their signed relative error ε:

$$\varepsilon := \frac{\text{interior} - \text{basic}}{\text{basic}} \tag{11}$$

In this section, we consider all numbers with absolute value $\leq 10^{-10}$ to be zero.

First, we note that Bangalore[2]_181d was not successfully solved by the barrier method alone. Closer investigation shows that this happens consistently for this model instance while it never happens for the closely related model Bangalore[1]_181d. The two models are identical apart from slightly different costs associated with technologies. As described in Appendix E, Bangalore[2]_181d scales the cost contribution of carbon by 0.365 compared to the original model and by a factor of 26 compared to Bangalore[1]_181d. Moreover, the Bangalore model associates dummy carbon costs with all technologies, whose sole purpose it is to prevent the solver from allocating unused capacities. The use of these technologies is otherwise unbounded. The combined effect of these modelling decisions is that Bangalore[2]_181d contains a set of unbounded variables whose cost contribution is almost zero (and lower than in other instances of this model). This formulation geometrically leads to unbounded faces that are almost "flat", i.e. even distant points on the face have almost the same cost. It is known that unbounded optimal faces may lead to numerical issues for barrier methods [44]. Indeed, associating slightly higher costs to electricity_transmission in Bangalore[2]_181d resolves the issue entirely: barrier consistently solves the modified problem to optimality. This case illustrates that barrier can be more susceptible to numerical issues than barrier+crossover. We will return to this insight in Sect.

Table 5 Fraction of non-zero elements in the decision variables of the interior and basic feasible solutions of the Bangalore[1]_181d

Variable	Length	Non-zeros		Explanation
		Basic	Inter	
energy_cap	166	0.53	0.93	Allocated energy capacity of each technology and location
carrier_prod	628992	0.31	0.89	Production of a carrier per location, technology and timestep
carrier_con	567840	0.37	1.0	Consumption of a carrier per location, technology and timestep
cost	288	0.46	0.92	Cost of each cost class per location and technology
resource_area	11	1.0	1.0	The area allocated for each technology per location
storage_cap	13	0.92	1.0	The storage capacity allocated of each storage technology per location
storage	56784	0.62	1.0	The energy stored per storage technology, location and timestep
resource_con	48048	0.51	0.51	The resource consumption per technology, location and timestep
resource_cap	11	1.0	1.0	The resource consumption capacity allocated per location and technology
carrier_export	56784	0.0	0.58	The exported amount of a carrier per location, technology and timestep
cost_var	314496	0.33	0.47	The variable cost per cost class, technology, location and timestep
cost_invest	288	0.42	0.88	The investment cost per cost class, location and technology

Length is the number of entries of the decision variable, the fraction of non-zero elements for the basic feasible solution and the interior solution are indicated as basic and inter, respectively

"Discussion" where we develop guidelines for model formulation.

Next, we observe that the interior solution is generally slightly worse than the basic feasible solution. However, the differences are negligible in all cases. Moreover, we recall that a better approximation can be obtained by tightening the barrier convergence tolerance parameter.

Barrier+crossover always performs additional work compared to just barrier, thus obtaining a basic feasible solution takes longer than obtaining an interior solution. A counter-example to this intuitive rule is the UK[1]_180d where obtaining an interior solution takes longer than obtaining a basic feasible solution. This is most likely due to solution time variability as discussed in Sect. "Methods" .

As expected, we can observe that the fraction of non-zeros is significantly higher in interior solutions: In many cases the interior solution contains twice as many non-zero decision variables as the basic feasible solution and almost all interior solutions have more than half of their decision variables away from their bounds. There are multiple reasons for this: first, there may be multiple optimal vertices in the polytope (due to, for instance, the inherent degeneracy of network-flow problems discussed above, or due to cost-free technologies in the model). By linearity of the problem, then, every linear combination of these optimal vertices is itself an optimal solution. While simplex will always return one of the optimal vertices, an interior point method will most likely converge to some other point in the optimal linear subspace. Second, interior point methods always return an interior point of the polytope that, though close to the boundary, still has most variables away from their bounds. Since zero-entries in the solution correspond to variables at their bounds, an interior solution, even when close to a vertex, still has many more non-zero entries than a basic solution at this vertex. The absolute magnitude of this second type of non-zeros is usually small and depends on the parametrization of the algorithm. Heuristically, we can remove such unwanted non-zeros from the solution by rounding but note that even after rounding, our examples exhibit a much larger fraction of non-zeroes in the interior point solution. Table 5 breaks down the non-zeros in the Bangalore[1]_181d model by decision variable. Decision variables in Calliope can have multiple dimensions, e.g., the decision variable energy_cap is a vector of installed technology capacities with one entry for each technology and location combination. Table 5 highlights why a basic feasible solution might be easier to interpret than an interior solution: the interior solution allocates many more technologies that produce and store electricity (energy_cap, storage_cap > 0). In addition, it schedules “facilities” to consume and produce energy on more different timesteps (carrier_prod, carrier_con > 0). Clearly, both interior and basic feasible solutions achieve the necessary allocation of carriers at (almost) identical cost, but we find the sparse solution easier to interpret, because it provides a more minimal set of necessary technologies. The risk of an equivalent but less sparse solution is that we might conclude from it, that each of the non-zero values is required for optimality, rather than just being an artefact of the algorithm. On the other hand, the fact that an interior point solution will return a linear combination of basic feasible solutions might make this solution

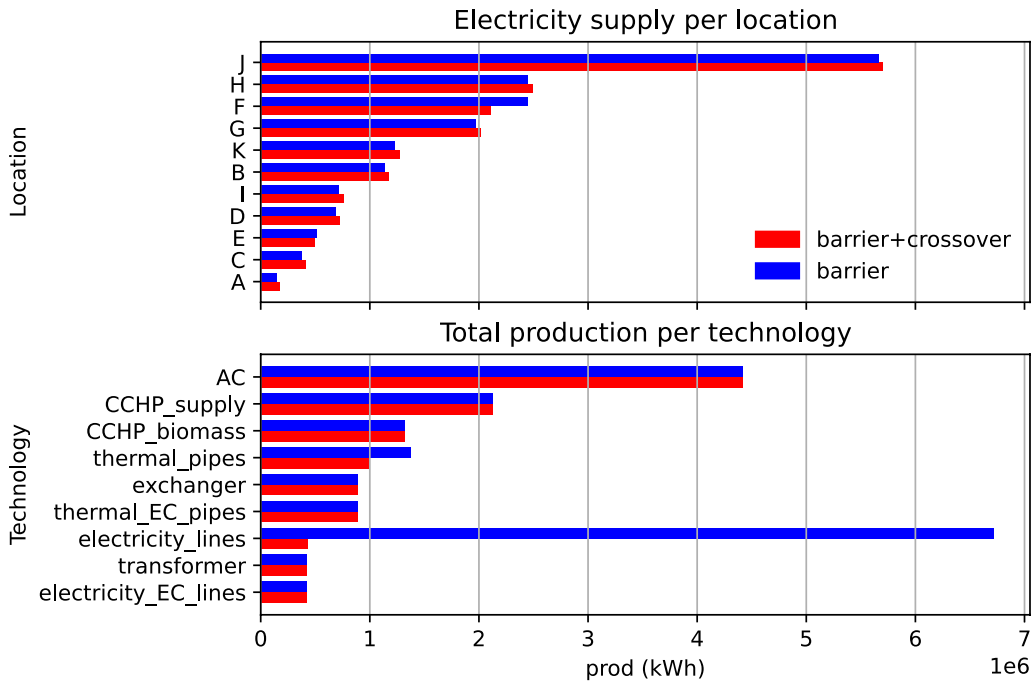

Fig. 2 Carrier production *prod* in the Bangalore[1]_181d model as computed by barrier+crossover and barrier alone, respectively. Top: Electricity supply from the national grid per location, aggregated over time. Bottom: Carrier production of technologies inside the system, aggregated over time and locations. Terminology: A-K are the demand nodes in the network (office buildings with demand for electricity and cooling). Apart from the demand nodes, there is one energy center (EC) as centralized energy and cooling distributor. Technologies in the lower plot are: Air conditioning (AC), step down transformer (transformer), cold water exchanger (exchanger), Combined Cooling, Heat & Power (CCHP), as well as transmission technologies between buildings and to and from the energy center: thermal energy pipelines (thermal_pipes and thermal_EC_pipes) for cooling, electricity lines (electricity_lines and electricity_EC_lines) for electricity

more realistic: instead of a minimal set of technologies the solution might suggest a more diverse setup.

As discussed in Sect. "Methods", the most central variables of Calliope energy system models are the installed technology capacities (*cap*) and the energy "production" amounts (*prod*), in addition to the cost parameters which directly and indirectly control these two variables. Figure 2 shows *prod* of the interior and basic feasible solution of the Bangalore[1]_181d model side-by-side. More precisely, it shows the production at "facilities" summed up over all time steps. Figure 2 suggests an intuitive interpretation for the differences between the interior and basic feasible solution: In the interior solution, more electricity is produced centrally at location F and then distributed to many other locations using electricity lines. These other locations, in turn, produce less energy themselves.

Automatically improving scaling

The major indicators of numerical problems we find across all three examined models are:

1. Crossover makes very slow progress and sometimes needs to be restarted.
2. Simplex makes only very slow progress on the objective value or jumps wildly in the solution space.
3. Barrier returns a sub-optimal objective.
4. The solver detects numerical issues and tries to counteract them by: switching to higher precision, dropping variables from the current basis, tightening Markowitz tolerance.

(1) and (2) often occur for the Euro model. (3) is common for the Bangalore model. We conclude that these models exhibit numerical issues that need to be addressed. The UK model, on the other hand, seldom shows any of these issues.

Figure 3 shows the average solution time of the Euro model for different time frames. Runs that did not successfully converge were not included in the average solution time but instead, their number is indicated in brackets. We find that our autoscaling approach reduces the average solution time of barrier+crossover in all cases we considered. For model sizes of at least $0.75 \cdot 10^7$, the average runtime with autoscaling is always at least 2x faster than without autoscaling and in many cases 3x to 4x faster. We notice that autoscaling not only reduces average solution time, but also leads to more regular

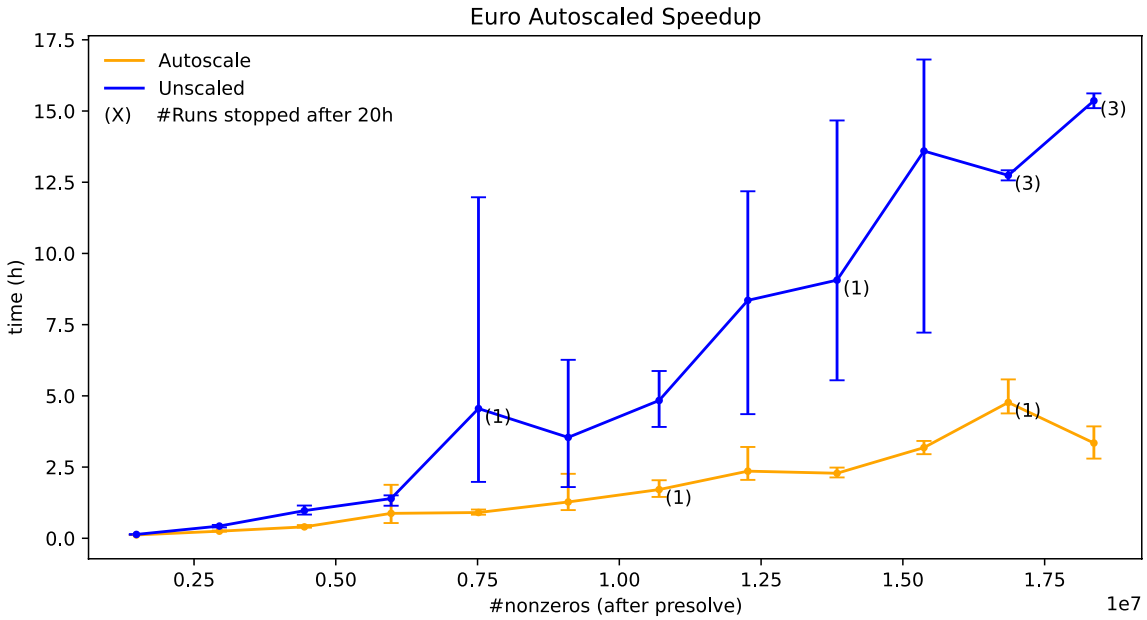

Fig. 3 Average absolute solution time for the Euro model for time frames between 1 and 12 months both with and without autoscaling and using barrier+crossover. Average was taken over 5 runs, the vertical bars show the max and min solution time. Runs taking more than 20 h were aborted and not considered in either average or errorbars. The number of aborted runs per instance is indicated in brackets next to the average solution time

solution time behaviour and reduces the number of times the algorithm does not converge. However, there are two large outliers in the scaled solution times that did not converge in time. This behaviour seems surprising because in both cases the remaining four runs terminated quickly and with similar solution time. We recall that only the random seed changes within a set of 5 runs of the same model instance. We will discuss this issue of extreme outlier behaviour of solution times below.

Figure 4 shows the effect of autoscaling on Euro_6m, UK_12m and Bangalore_6m. We solve each model using Gurobi's barrier+crossover method both with, and without autoscaling enabled. The average solution time per model with and without autoscaling is indicated by the height of each bar. In the Euro_6m model the solution time improves by about 3x when applying autoscaling. In Bangalore there is still a noticeable improvement of solution time after scaling. In the UK model the solution time is hardly affected. This supports our hypothesis that the UK model is generally well formulated already and does not cause numerical issues. Looking at the numerical range before and after scaling of the three model instances in Table 6 shows that the original Euro_6m model has a numerical range close to 16 orders of magnitude. As discussed in Sect. "Scaling" this may very well be the cause of grave numerical issues. Moreover, scaling achieves a significant reduction in the range of the Euro_6m model. The other two model instances, however, have a more moderate numerical range and scaling achieves a much smaller improvement of the numerical range which explains the solution time for solving these problems is less affected. Gurobi recommends numerical ranges of at most 10^9, further supporting these conclusions.

In Fig. 4, we report the fraction of the solution time spent in each phase of the solution algorithm (to be precise, the different sections of each bar represent the fraction between the average of the corresponding phase and the average of the total solution time). The Checkscale phase is not part of the solving but it corresponds to the fraction of code where we check and report the numerical range of the model. In the case where autoscaling is enabled, it also includes the time to compute the optimal scaling factors and to perform the scaling of the model. We note that the contribution of autoscaling to the solution time of the algorithm is negligible. The Preprocess phase corresponds to the

Table 6 Improvement of κ' by scaling

Model	κ' Before scaling	κ' After scaling
Euro_6m	$3.7 \cdot 10^{15}$	$3.3 \cdot 10^{8}$
UK_12m	$3.4 \cdot 10^{11}$	$7.7 \cdot 10^{7}$
Bangalore_6m	$5.1 \cdot 10^{10}$	$3.1 \cdot 10^{9}$

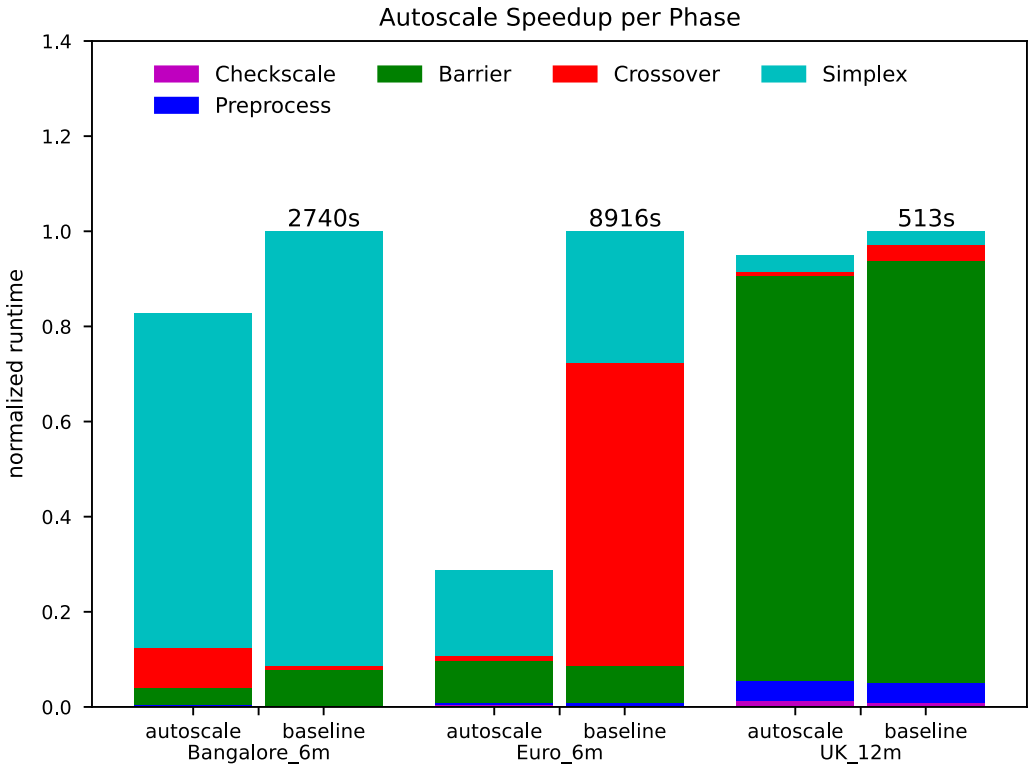

Fig. 4 Normalized solution time of barrier+crossover for three models with autoscaling (left bar) and without autoscaling (right bar). The solution time is broken down into the Checkscale phase, where we inspect the numerical range of the model and apply autoscaling if enabled, and the four main phases of the algorithm. Each bar is the average of the solution times of 10 experiments. Absolute solution time in seconds is indicated above the bars of the unscaled models. Error bars were omitted to improve readability

solver-internal preprocessing. Barrier corresponds to the execution of the barrier algorithm. We split the crossover phase into two parts and plot them separately: crossover, where a basic point (a vertex) close to the interior solution is found, and simplex, where this basic point is re-optimized using simplex steps. Interestingly, for the Euro and Bangalore models different phases of the algorithm are affected by scaling: Euro_6m gains most in the crossover phase (red) whereas for Bangalore_6m the crossover phase with scaling takes even longer than without scaling. Bangalore_6m instead gains most in the simplex phase (cyan). The results also suggest that the barrier method is generally least sensitive to scaling: whereas crossover and simplex phases often speed up noticeably, autoscaling usually shows only very minor improvement.

An important insight is that every phase of the algorithm can encounter numerical difficulties and that the exact nature of the problems leading to deteriorating solution time is subtle. In particular, a measure that improves the solution time of some phase may badly affect the solution time of another phase. Finding a scaling that always works and never deteriorates solution time proved to be a major difficulty in designing a suitable autoscaling approach.

Performance variability caused by scaling

While scaling often reduces average time to solve a specific model using the barrier+crossover method, we observed in the previous section that it does not eliminate the risk of non-convergence or extremely long solution times for solving certain models (consider for instance the outlier behaviour in Fig. 3). In particular, we regularly observe that the final simplex phase does not converge. As described above, there are two different flavours of this: either simplex just stops making progress or it jumps wildly in the solution space. The latter seems to be a strategy Gurobi applies to deal with stalling progress in the simplex solver.

An intuitive explanation for why scaling might induce this behaviour is given by [8] and briefly discussed in Sect. "Scaling": while scaling can improve the average condition number, it can increase the condition number of certain bases in the matrix. [8] gives an illustrative example where scaling increases the condition number of all relevant bases in an LP by an arbitrarily large

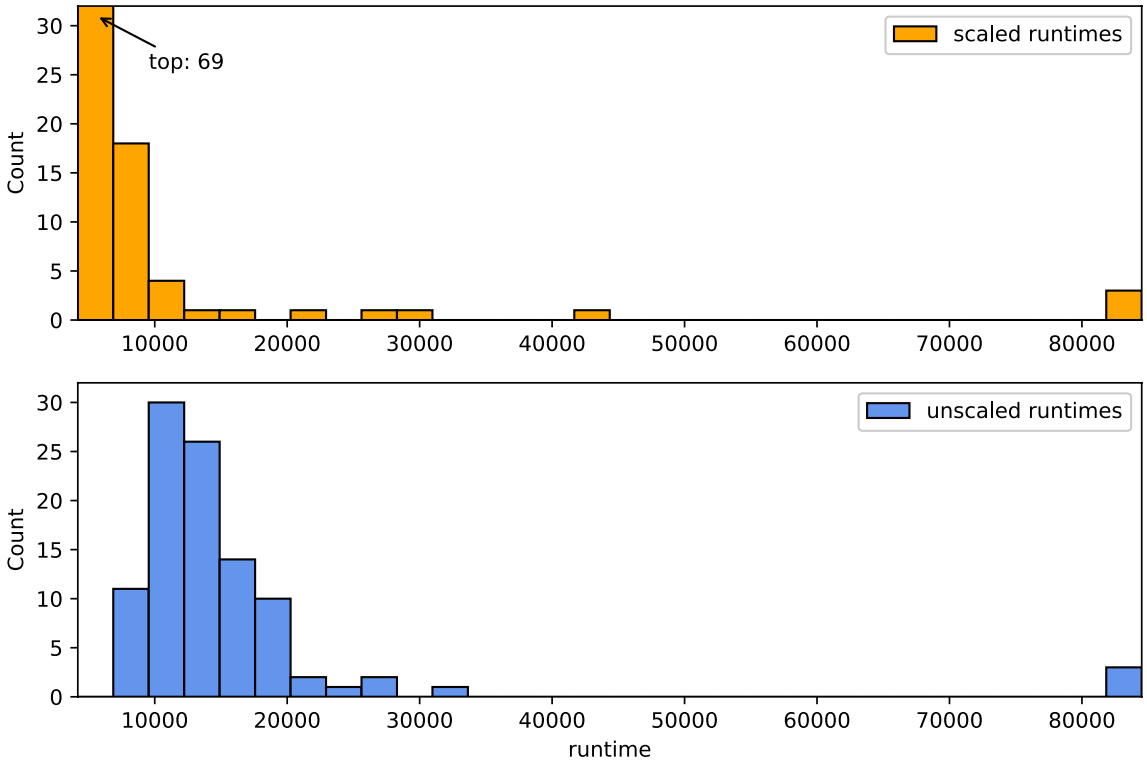

Fig. 5 Histogram with 30 bins showing runtimes of 100 runs of each, the scaled and unscaled Euro_6m model

amount. As the bases encountered during simplex are subject to randomness, this may explain why scaling in some cases leads to very long solution times or even non-convergence.

In the following, we want to quantify this effect. Pragmatically, to judge if autoscaling is still useful we want to answer the following questions

1. What is the average time it takes to solve a given model?
2. How often will we encounter outlier cases with very long solution time?

We have seen above that scaling can effectively improve average solution time of a model in a previous paragraph. In order to address the second question we perform the following experiment: We consider the Euro_6m model, which exhibits the outlier behaviour discussed above, and run it 100 times with scaling and 100 times without scaling. We then evaluate the empirical distribution function of these solution times.

Figure 5 shows the histogram of runtimes of both the scaled and the unscaled data. The runtimes at the right end correspond to timeout runs. While the runtimes of the scaled model have a 40% lower mean than the ones from the unscaled model, their variance is 25% higher. While the number of runs that don't converge within the set time is the same with and without autoscaling, a distribution fitted to the scaled solution times would have a heavier tail than a distribution fitted to the unscaled solution times.

Discussion

We find that barrier is generally superior to simplex for the kinds of models investigated here, and barrier alone is faster than barrier+crossover, as the latter performs strictly additional work. The difference in solution time, however, is unpredictable and can be either large or barely noticeable, depending on the model instance. Barrier also sometimes fails to converge. In particular, while degeneracy per se is no problem for barrier, the specific case of unbounded (large) optimal surfaces may lead to non-convergence of barrier. Such cases can potentially be remedied if crossover is run after barrier. However, we find it difficult to answer the question of whether or not running crossover is worth the additional effort in finding a basic feasible solution. Interior solutions inherently have a larger number of non-zero values in their decision variables as compared to a basic feasible solution. This may negatively affect the interpretability of results. For instance, the interior solution of one Bangalore model produced more electricity in a centralized location and

distributed it via transmission links compared to the basic feasible solution. Investing in the additional effort of crossover must likely be decided on a case-by-case and model-by-model basis, and underscores once again the general problem with relying on a single, "optimal" result [7, 20, 25].

To enable the use of crossover without excessive solution times, we find that model scaling can be helpful; our autoscaling approach often reduces the average solution time of barrier+crossover significantly. This reduction generally happens in the crossover and simplex phases. From this, we extrapolate that autoscaling has a negligible effect on the solution time of barrier alone but that it improves the average solution time of the simplex method. We find that autoscaling helps to prevent numerical problems in the solver. In particular, solution times for the numerically challenging Euro and Bangalore models were improved drastically while the numerically well-behaved UK model did not benefit much from autoscaling. We also find that autoscaling increases the probability of solution time outliers: We see more exceptionally long solution times when using autoscaling. This problem requires further investigation in future work. However, as the autoscaling yields almost always shorter solution times and higher convergence rates than our unscaled base cases, autoscaling seems to be a no-regret solution in terms of solution time for barrier+crossover.

A separate issue which is not easily addressed with scaling is the problem of cost-free technologies. The model Bangalore^2_181d consistently failed to converge on a solution using the barrier method (Table 4). For this model, we found that two small modifications to the original model formulation each lead to successful optimization by barrier: increasing costs for the installed capacity of electricity transmission technologies by 2 orders of magnitude, and upper-bounding the energy capacity of electricity transmission. Both measures counteract the numerical problems caused by unbounded optimal faces, each in a different way, supporting our hypothesis that these problems are due to unbounded optimal faces. It is thus advisable to avoid model formulations with free and unbounded technologies. There are, however, cases in the real world where some technologies are virtually free to operate or are free to install given the system scope. Electricity transmission is one example: their operation cost can be considered negligible compared to the cost of installation, and sizing them might be outside the scope of the problem altogether. In these cases it seems unavoidable to resort to dummy costs, but the modeller should be aware that it is non-trivial to set these dummy costs in a way that represents the real world whilst also being high enough to avoid unbounded optimal faces.

If updates to the algorithm, model scaling, and cost-free technologies do not enable model tractability, adjusting solver tolerances may still help. Since solver authors made certain assumptions about the input models they will need to solve when setting tolerances, they may not be optimally set for the numerical range or required accuracy of energy system models. We have found that choosing the right value for tolerances is hard and usually the underlying problem is not solved by modifying tolerances, thus it is only advisable to do so on rare occasions. Three key tolerances are Feasibility, Optimality and Barrier-Convergence[5]. They all control how tightly some inequality must be fulfilled. Tightening Optimality and Barrier-Convergence tolerances will lead to a better objective value in the solution returned by simplex/barrier+crossover methods and barrier methods, respectively. However, tightening these tolerances too much may lead to longer solution times and (in the case of barrier) to non-convergence. The Feasibility tolerance controls how strictly constraints need to be satisfied for a solution to be feasible. Loosening these tolerances may lead to faster convergence of simplex and barrier+crossover methods. However, this usually does not magically resolve all difficulties with numerically challenging problems. It also requires careful experimentation to ensure results do not violate physical properties of the system being described (e.g., negative stored energy). Solvers often feature parameters to tune theirs algorithms for numerically challenging problems. Examples are the NumericFocus and Quad parameter of Gurobi. NumericFocus will cause Gurobi to spend more time detecting and reacting to numerical issues while the Quad parameter will use a larger numeric type for representing all numbers in the problem. Both options trade performance for numerical stability.

Conclusions

In the context of the results discussed above, we can formulate the following guidelines to improve the computational performance of typical energy system optimisation models:

- On large and difficult models, manually select the barrier method or barrier+crossover method. Prefer the latter if you suspect your model formulation to be numerically unstable or if you want a minimal solution in terms of technology allocations.

[5] When using crossover, there is also the *Markowitz* tolerance, described in more detail in Appendix D, but we found no evidence that it improved performance in our benchmarks.

- Use appropriate units that minimize the model's numerical range or apply an automatic scaling procedure like the one we introduce here to derive them automatically.
- Be wary of model formulations with cost-free technologies and dummy costs, as those can dramatically worsen the numerical properties of the model and thus increase solution time
- Know the basic solver tolerance settings for your chosen solver and adjust them if necessary; however, this should usually be the very last resort.

Ultimately, more systematic work to understand the properties of energy system models could help them provide better decision support, for example by making it possible to rapidly explore large numbers of scenarios or alternative solutions even in models that depict the system with high spatial and temporal detail. Promising avenues are custom solvers that exploit these model properties [33], or even solvers that exploit the fact that a single optimal solution is not necessarily useful; a range of near-optimal solutions, for example extracted from an interior-point algorithm, could be just as relevant for real-world applications [7, 20, 25]. To complement this, more systematic work is needed on the difference between basic feasible and interior solutions and the implications of these differences for the resulting energy system designs. While black-box solvers like the ones examined here are continuing to be developed and becoming more powerful, the energy modelling community could reap many practical benefits if such work can improve solution times by multiples or even orders of magnitude, as some of our explorations here suggests may be possible. Finally, of course, optimisation is not always the appropriate method. For example, some problems may be better solved with simulation approaches. Yet it is clear that large linear optimisation models will continue to play a role for the foreseeable future, so improving their tractability is a useful effort.

Appendices

Proof of equivalence between scaled and unscaled LP

Let us denote by R and S two diagonal matrices

$$R = \mathbf{diag}(r_1, \ldots, r_n), \qquad S = \mathbf{diag}(s_1, \ldots, s_n) \tag{12}$$

with $r_i, s_i > 0$. Furthermore, let $A \in \mathbb{R}^{m \times n}$, $b \in \mathbb{R}^m$ and $c \in \mathbb{R}^n$. Let's consider the three polyhedra $P := \{x \mid Ax \leq b\}$, $P'' := \{x \mid RAx \leq Rb\}$ and $P' := \{x \mid RASx \leq Rb\}$. then the following two LPs are equivalent

$$\min\{c^\intercal x \mid x \in P\}, \qquad \min\{c^\intercal Sx \mid x \in P'\} \tag{13}$$

To see this note that $P = P''$ because R is invertible and that for each $x \in \mathbb{R}^n$ holds $x \in P' \Leftrightarrow Sx \in P''$. □

Condition number and numerical range

As discussed in Sect. "Scaling", both condition number $\kappa(A)$ and the numerical range of A influence how well simplex and barrier can solve the LP with constraint matrix A. We can affect both notions by row and column scaling. Consider the matrix

$$\begin{bmatrix} 1 & 0 \\ 0 & \varepsilon \end{bmatrix} \tag{14}$$

which has arbitrarily large condition number and numerical range $\frac{1}{\varepsilon}$. Scaling the second row of 14 with $\frac{1}{\varepsilon}$ yields the identity matrix which has both condition number and numerical range of 1.

On first sight, it may seem as though minimizing numerical range will always also reduce the condition number. The next example illustrates that this relationship does not hold in general. consider the matrix

$$\begin{bmatrix} 1 & \varepsilon \\ \varepsilon & 1 \end{bmatrix} \tag{15}$$

Note that if $\varepsilon = 1$ the numerical range of the matrix is 1, as small as it can get. At the same time, the matrix is singular, i.e., its condition number is ∞. Making ε go to 0 will continually increase the numerical range while decreasing the condition number of the matrix. It follows, that we cannot in general optimize both the condition number and the numerical range simultaneously.

A second problem with improving the condition number of LPs is that it's not possible to improve the condition number of all bases simultaneously: improving the condition number of one basis may adversely affect the condition number of another basis. Since it is unknown a priori which bases of A will be visited in the course of the simplex algorithm it is possible that a scaling method has a negative effect on the bases actually encountered during simplex. Elble and Sahinidis [8] investigates the effects of many mainstream scaling methods on the condition number in a large set of practical linear programs.

Scaling method

In the following we elaborate on the two main constraints we face when choosing scaling factors.

Scaling factors that tamper with the precision of input values will corrupt the model In order to see the significance of numerical precision, consider the following example: Consider the inequality $0.1x \leq 100$. If we scale this inequality by $\frac{1}{3}$ we get the inequality $\frac{1}{30}x \leq \frac{100}{3}$. In mathematics, the second inequality is equivalent to the first one, but in floating point arithmetic it is not. Let us assume for simplicity that we work in a decimal floating point system with 5 decimal digits precision, then the second inequality becomes $0.03333x \leq 33.33333$. Note that 100 was scaled by 0.3333333 and that 0.1 was scaled only by 0.3333. The inequality is thus no longer equivalent to the original inequality; in fact, the second inequality now reduces to $x \leq 1000.0999$. Note that the relative error $\frac{1000.0999-1000}{1000}$ is exactly $\frac{0.3333333}{0.3333} - 1$. The relative error thus increases with the numerical range of the inequality. This loss of precision can be avoided if all scaling factors are chosen to be powers of 2.

Optimal scaling factors may lead to prohibitively large or small values Gurobi recommends that all values in a model be between 10^{-3} and 10^{6}.[6] The absolute size of values matters to the solver because it internally uses absolute tolerances to compare values. In particular, having input values smaller than the solvers tolerances means that the solver can no longer distinguish between legitimate values and values arising from round-off errors [16]. Ensuring that absolute values stay above a certain threshold sometimes limits how much κ' can be decreased. Consider the following example:

Let u and v be the two base units in our model and let's assume that the derived units in our model all have the form u, v or $\frac{u}{v}$. Assume the values in our model have the ranges as shown in Table 7.

Here, $\kappa' = 10^5$ cannot be decreased any further. Assume that we want to ensure that all values in our model have absolute value at least 0.01. To achieve this, we need to choose scaling factors $f_v \geq 10$ and $f_u \geq 10 \cdot f_v$. Thus after scaling it holds that $\kappa' \geq \frac{10 \cdot f_v \cdot 50}{f_v \cdot 0.001} = 5 \cdot 10^5$ which is larger than the original κ'. There is thus a trade-off between minimizing κ' and ensuring that the values have a sensible absolute size.

Table 7 Example range of values in a model to demonstrate the inability to reduce κ' if absolute values are to stay within a recommended range

Unit	Min	Max
u	1	50
v	0.001	100
$\frac{u}{v}$	0.001	100

[6] https://www.gurobi.com/documentation/9.0/refman/num_advanced_user_scaling.html.

Markowitz tolerance

In computing the LU factorization of a matrix, the *Markowitz tolerance* controls which elements are sufficiently large to be used as pivots. An LU factorization of the constraint matrix A is computed at several different stages of the algorithm: during the simplex algorithm in order to recompute the current basis [22, 38, 39] and during crossover in order to construct a valid basis from scratch [3]. Choosing a large Markowitz tolerance means that many potential pivots are disregarded in order to ensure a numerically stable basis. When solving certain numerically challenging models, the crossover method will often need to restart with an updated value of the Markowitz tolerance. One might expect that setting a more conservative value for the Markowitz tolerance in the first place will prevent this but we could not consistently verify this in our benchmarks.

Model variations in experiments

In this section, we describe the model variations we used in "To crossover or not". Table 8 shows the input parameters which vary in the explored variations of the Euro model, while Tables 9 and 10 show the respective parameter variations for the UK and Bangalore models.

Euro model

Battery cost controls the cost per energy capacity and the cost per storage capacity of batteries.

Zones varies the number of countries in the model. While the baseline model consists of 34 European countries, each making up one zone of the model, the variations consist of 20 countries and 1 country (Germany), respectively.

CO2 caps limits the total amount of co2 produced in each location. The bound ranges from 1.2 Mt in Cyprus to 184 Mt in Germany.

Renewable shares requires a certain share of the total electricity consumption of each country to exceed a minimum value. This value varies per country and ranges from 11% in Luxembourg to 78% in Portugal.

Hydro reservoir controls whether the hydro reservoir technology can be used to store energy (yes) or not (no).

Table 8 Parameter variations in the different explored variants of the Euro model

Model	Battery cost	Zones	CO2 caps	Renewable shares	Hydro reservoir	Data source
Euro_180d	135€/kW, 315€/kWh	34	Yes	Yes	No	Original
Euro[1]_180d	135€/kW, 315€/kWh	20	No	No	Yes	Averaged
Euro[2]_180d	135€/kW, 315€/kWh	20	Yes	Yes	No	Original
Euro[3]_180d	135€/kW, 315€/kWh	1	Yes	Yes	No	Original
Euro[4]_180d	68€/kW, 158€/kWh	1	Yes	Yes	No	Original
Euro[5]_180d	34€/kW, 79€/kWh	1	Yes	Yes	No	Original
Euro[6]_180d	135€/kW, 315€/kWh	1	No	Yes	No	Original

Table 9 Parameter variations in the different explored variants of the UK model

Model	Battery cost	Imports	Renewables share	New Nuclear
UK_180d	140£/kW,109£/kWh	12.75 GW	0%	No
UK[1]_180d	140£/kW,109£/kWh	12.75 GW	0%	No
UK[2]_180d	300£/kW,200£/kWh	0 GW	80%	No
UK[3]_180d	75£/kW,50£/kWh	25.5 GW	40%	Yes
UK[4]_180d	150£/kW,100£/kWh	12.75 GW	50%	No
UK[5]_180d	75£/kW,50£/kWh	0 GW	90%	No
UK[6]_180d	75£/kW,50£/kWh	0 GW	100%	No

Table 10 Parameter variations in the different explored variants of the Bangalore model

Model	Cost of carbon
Bangalore_181d	1
Bangalore[1]_181d	9.49
Bangalore[2]_181d	0.365

Data source describes what source was used in the model to control the resource constraints of renewables. The original model contains csv timeseries for each of wind, pv and hydro. In average, these values were simply averaged over the whole timerange.

UK model

Battery cost controls the cost per energy capacity and the cost per storage capacity of batteries.

Imports sets the high-voltage energy import per zone. The value shown in the table is aggregated over all zones.

A *Renewables share* of X% forces the combined electricity production of wind_onshore, wind_offshore, pv_rooftop, pv_utility_scale and hydro to make up at least X% of the total electricity.

New Nuclear allows for 17.3 GW additional energy produced by nuclear technology.

Bangalore model

Cost of carbon is the factor of the cost of emitted carbon in the objective function.

Acknowledgements
Not applicable

Author contributions
M.B., B.P, T.T and S.P. designed the research and developed the models used in the experiments, M.B. performed the research, analysed the data, and plotted the figures, M.B. and S.P. drafted the manuscript, M.B., B.P., T.T. and S.P. discussed and revised the manuscript.

Funding
The authors acknowledge funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 837089, and the SEEDS project supported by the CHIST-ERA grant CHIST-ERA-19-CES-004, the Swiss National Science Foundation grant number 195537, the Fundação para a Ciência e Tecnologia (FCT) grant number CHIST-ERA/0005/2019, the Spanish Agencia Estatal de Investigación with grant PCI2020-120710-2, and the Estonian Research Council grant number 4-8/20/26.

Availability of data and materials
This manuscript has also been submitted to arXiv as a preprint, with exactly the same content as in the currently submitted revision: arxiv.org/abs/2211.12299
All experiments in this work were performed using the Calliope open-source modelling framework, accessible on Github (https://github.com/calliope-project/calliope). The automated scaling method was implemented on a Calliope fork, accessible on Github (https://github.com/brmanuel/calliope). Further, all Calliope models used in the experiments are available on Github (https://github.com/brmanuel/calliope-models).

Declarations

Ethics approval and consent to participate
Not applicable.

Consent for publication
Not applicable.

Competing interests
The authors have no competing interests to declare that are relevant to the content of this article.

Received: 3 August 2023 Accepted: 12 May 2024
Published online: 03 June 2024

References

1. Ahuja R, Orlin J, Magnanti T (1993) Network flows: theory. algorithms, and applications, Prentice Hall, chap 11:419–467
2. Babonneau F, Caramanis M, Haurie A (2017) ETEM-SG: optimizing regional smart energy system with power distribution constraints and options. Environ Model Assess 22(5):411–430. https://doi.org/10.1007/s10666-016-9544-0
3. Bixby RE, Saltzman MJ (1994) Recovering an optimal lp basis from an interior point solution. Oper Res Lett 15(4):169–178. https://doi.org/10.1016/0167-6377(94)90074-4
4. Burandt T, Xiong B, Löffler K, Oei PY (2019) Decarbonizing China's energy system—modeling the transformation of the electricity, transportation, heat, and industrial sectors. Appl Energ 255(113):820. https://doi.org/10.1016/j.apenergy.2019.113820
5. Chang M, Thellufsen JZ, Zakeri B, Pickering B, Pfenninger S, Lund H, Østergaard PA (2021) Trends in tools and approaches for modelling the energy transition. Appl Energy 290(116):731. https://doi.org/10.1016/j.apenergy.2021.116731
6. Connolly D, Lund H, Mathiesen BV, Leahy M (2010) A review of computer tools for analysing the integration of renewable energy into various energy systems. Appl Energy 87(4):1059–1082. https://doi.org/10.1016/j.apenergy.2009.09.026
7. DeCarolis JF, Babaee S, Li B, Kanungo S (2016) Modelling to generate alternatives with an energy system optimization model. Environ Modell Softw 79:300–310. https://doi.org/10.1016/j.envsoft.2015.11.019
8. Elble J, Sahinidis N (2012) Scaling linear optimization problems prior to application of the simplex method. Comput Optim Appl 52(2):345–371. https://doi.org/10.1007/s10589-011-9420-4
9. Gabrielli P, Gazzani M, Martelli E, Mazzotti M (2018) Optimal design of multi-energy systems with seasonal storage. Appl Energy 219:408–424. https://doi.org/10.1016/j.apenergy.2017.07.142
10. Gabrielli P, Fürer F, Mavromatidis G, Mazzotti M (2019) Robust and optimal design of multi-energy systems with seasonal storage through uncertainty analysis. Appl Energy 238:1192–1210. https://doi.org/10.1016/j.apenergy.2019.01.064
11. Göke L (2021) Anymod. jl: a julia package for creating energy system models. SoftwareX 16:100871. https://doi.org/10.1016/j.softx.2021.100871
12. Grams CM, Beerli R, Pfenninger S, Staffell I, Wernli H (2017) Balancing Europe's wind-power output through spatial deployment informed by weather regimes. Nat Clim Change 7(8):557–562. https://doi.org/10.1038/nclimate3338
13. Gurobi Optimization L (2021) Gurobi optimizer reference manual. http://www.gurobi.com
14. Hoffmann M, Kotzur L, Stolten D, Robinius M (2020) A review on time series aggregation methods for energy system models. Energies 13(3):641. https://doi.org/10.3390/en13030641
15. Hörsch J, Hofmann F, Schlachtberger D, Brown T (2018) PyPSA-Eur: an open optimisation model of the European transmission system. Energy Strateg Rev 22:207–215. https://doi.org/10.1016/j.esr.2018.08.012
16. Klotz E, Newman AM (2013) Practical guidelines for solving difficult linear programs. Surv Oper Res Manag Sci 18(1):1–17. https://doi.org/10.1016/j.sorms.2012.11.001
17. Koch T, Achterberg T, Andersen E, Bastert O, Berthold T, Bixby RE, Danna E, Gamrath G, Gleixner A, Heinz S, Lodi A, Mittelmann H, Ralphs T, Salvagnin D, Steffy DE, Wolter K (2010) Miplib 2010. Tech. Rep. 10-31, ZIB, Takustr. 7, 14195 Berlin
18. Kotzur L, Markewitz P, Robinius M, Stolten D (2018) Impact of different time series aggregation methods on optimal energy system design. Renew Energy 117:474–487. https://doi.org/10.1016/j.renene.2017.10.017
19. Kotzur L, Markewitz P, Robinius M, Stolten D (2018) Time series aggregation for energy system design: modeling seasonal storage. Appl Energy 213:123–135. https://doi.org/10.1016/j.apenergy.2018.01.023
20. Lombardi F, Pickering B, Colombo E, Pfenninger S (2020) Policy decision support for renewables deployment through spatially explicit practically optimal alternatives. Joule.https://doi.org/10.1016/j.joule.2020.08.002
21. Lougee-Heimer R (2003) The common optimization interface for operations research: promoting open-source software in the operations research community. IBM J Res Dev 47(1):57–66. https://doi.org/10.1147/rd.471.0057
22. Luce R, Tebbens JD, Liesen J, Nabben R, Grötschel M, Koch T, Schenk O (2009) On the factorization of simplex basis matrices. https://nbn-resolving.org/urn:nbn:de:0297-zib-11392
23. Majewski D, Wirtz M, Lampe M, Bardow A (2017) Robust multi-objective optimization for sustainable design of distributed energy supply systems. Comput Chem Eng 102:26–39. https://doi.org/10.1016/j.compchemeng.2016.11.038
24. Mavromatidis G, Orehounig K, Carmeliet J (2018) Comparison of alternative decision-making criteria in a two-stage stochastic program for the design of distributed energy systems under uncertainty. Energy 156:709–724. https://doi.org/10.1016/j.energy.2018.05.081
25. Pedersen TT, Victoria M, Rasmussen MG, Andresen GB (2021) Modeling all alternative solutions for highly renewable energy systems. Energy 234(121):294. https://doi.org/10.1016/j.energy.2021.121294
26. Pfenninger S (2017) Dealing with multiple decades of hourly wind and PV time series in energy models: a comparison of methods to reduce time resolution and the planning implications of inter-annual variability. Appl Energy 197:1–13. https://doi.org/10.1016/j.apenergy.2017.03.051
27. Pfenninger S, Pickering B (2018) Calliope: a multi-scale energy systems modelling framework. Journal of Open Source Software 3(29):825, https://doi.org/10.21105/joss.00825
28. Pfenninger S, Hawkes A, Keirstead J (2014) Energy systems modeling for twenty-first century energy challenges. Renew Sust Energy Rev 33:74–86. https://doi.org/10.1016/j.rser.2014.02.003
29. Pickering B, Choudhary R (2019) District energy system optimisation under uncertain demand: handling data-driven stochastic profiles. Appl Energy 236:1138–1157. https://doi.org/10.1016/j.apenergy.2018.12.037
30. Pickering B, Choudhary R (2021) Quantifying resilience in energy systems with out-of-sample testing. Appl Energy 285(116):465. https://doi.org/10.1016/j.apenergy.2021.116465
31. Prina MG, Manzolini G, Moser D, Nastasi B, Sparber W (2020) Classification and challenges of bottom-up energy system models—a review. Renew Sust Energy Rev 129(109):917. https://doi.org/10.1016/j.rser.2020.109917
32. Rehfeldt D, Hobbie H, Schönheit D, Gleixner AM, Koch T, Möst D (2019) A massively parallel interior-point solver for linear energy system models with block structure. https://opus4.kobv.de/opus4-zib/files/7432/ip4energy3.pdf
33. Rehfeldt D, Hobbie H, Schönheit D, Koch T, Möst D, Gleixner A (2021) A massively parallel interior-point solver for lps with generalized arrowhead structure, and applications to energy system models. Eur J Oper Res. https://doi.org/10.1016/j.ejor.2021.06.063
34. Ringkjøb HK, Haugan PM, Solbrekke IM (2018) A review of modelling tools for energy and electricity systems with large shares of variable renewables. Renew Sust Energy Rev 96:440–459. https://doi.org/10.1016/j.rser.2018.08.002
35. Scholz Y, Fuchs B, Borggrefe F, Cao KK, Wetzel M, von Krbek K, Cebulla F, Gils H, Fiand F, Bussieck M, Koch T, Rehfeldt D, Gleixner A, Khabi D, Breuer T, Rohe D, Hobbie H, Schönheit D, Yilmaz H, Buchholz S (2020) Speeding up energy system models—a best practice guide. Tech. rep
36. Schütz T, Schraven MH, Fuchs M, Remmen P, Müller D (2018) Comparison of clustering algorithms for the selection of typical demand days for energy system synthesis. Renew Energy 129:570–582. https://doi.org/10.1016/j.renene.2018.06.028
37. Staffell I, Pfenninger S (2018) The increasing impact of weather on electricity supply and demand. Energy 145:65–78. https://doi.org/10.1016/j.energy.2017.12.051
38. Suhl L, Suhl UH (1993) A fast LU update for linear programming. Ann Oper Res 43:33–47. https://doi.org/10.1007/BF02025534
39. Suhl UH, Suhl LM (1990) Computing sparse lu factorizations for large-scale linear programming bases. ORSA J Comput 2(4):325–335. https://doi.org/10.1287/ijoc.2.4.325

40. Tomlin JA (1975) On scaling linear programming problems, Springer Berlin Heidelberg, Berlin, Heidelberg, pp 146–166. https://doi.org/10.1007/BFb0120718
41. Tomlin JA (1989) A note on comparing simplex and interior methods for linear programming, Springer New York, New York, NY, pp 91–103. https://doi.org/10.1007/978-1-4613-9617-8_6
42. Tröndle T, Lilliestam J, Marelli S, Pfenninger S (2020) Trade-offs between geographic scale, cost, and infrastructure requirements for fully renewable electricity in europe. Joule 4(9):1929–1948. https://doi.org/10.1016/j.joule.2020.07.018
43. Vanderbei RJ et al (2015) Linear programming, vol 3. Springer. https://doi.org/10.1007/978-3-030-39415-8
44. Various (2017) Ibm ilog cplex optimization studio cplex user's manual. Tech. Rep. Version 12 Release 8, IBM
45. Zappa W, Junginger M, van den Broek M (2019) Is a 100% renewable European power system feasible by 2050? Appl Energy 233–234:1027–1050. https://doi.org/10.1016/j.apenergy.2018.08.109

Publisher's Note