CHALMERS
UNIVERSITY OF TECHNOLOGY

SCALFLEX: A Scalable Heuristic Framework for Distributed Energy Flexibility and Grid Tariff Optimization

Downloaded from: https://research.chalmers.se, 2026-10-15 05:07 UTC

Citation for the original published paper (version of record):
Duvignau, R., Khan, W., Papatriantafilou, M. et al (2026). SCALFLEX: A Scalable Heuristic Framework for Distributed Energy Flexibility and Grid Tariff Optimization. ACM Sustainability Week Companion 2026 Proceedings of the 2026 ACM Sustainability Week: 315-321. http://dx.doi.org/10.1145/3765611.3815592

N.B. When citing this work, cite the original published paper.

research.chalmers.se offers the possibility of retrieving research publications produced at Chalmers University of Technology. It covers all kind of research output: articles, dissertations, conference papers, reports etc. since 2004. research.chalmers.se is administrated and maintained by Chalmers Library

(article starts on next page)

SCALFLEX: A Scalable Heuristic Framework for Distributed Energy Flexibility and Grid Tariff Optimization

Romaric Duvignau
Department of Computer Science and Engineering, Chalmers University of Technology and University of Gothenburg
SE-412 96 Gothenburg, Sweden
duvignau@chalmers.se

Wania Khan
Department of Computer Science and Engineering, Chalmers University of Technology and University of Gothenburg
SE-412 96 Gothenburg, Sweden
wania@chalmers.se

Marina Papatriantafilou
Department of Computer Science and Engineering, Chalmers University of Technology and University of Gothenburg
SE-412 96 Gothenburg, Sweden
ptrianta@chalmers.se

Vincenzo Gulisano
Department of Computer Science and Engineering, Chalmers University of Technology and University of Gothenburg
SE-412 96 Gothenburg, Sweden
vinmas@chalmers.se

Abstract

The increasing deployment of distributed energy resources is rapidly expanding the flexibility available in power systems (e.g., shifting consumption or storage in response to price signals), creating significant computational challenges for large-scale coordination. We present ScalFlex, a scalable heuristic framework for efficient optimization of distributed flexibility. ScalFlex approximates new optimization tasks by reusing previously computed solutions for similar load profiles, enabling high-throughput scheduling with limited optimality loss. We evaluate the framework in two applications: (i) large-scale household battery scheduling under dynamic pricing, and (ii) the computation of hourly grid tariffs to reduce peak demand through coordinated flexibility shifts. Using data from over 2,000 households and more than 100,000 optimization tasks, ScalFlex achieves up to 18× higher throughput than a linear programming baseline while remaining within a few percent of optimality. These results show that computation-aware heuristics enable scalable coordination of distributed flexibility and tariff-driven mechanisms.

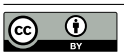
CCS Concepts

• **Applied computing**; • **Hardware** → *Energy generation and storage*; • **Computing methodologies** → *Optimization algorithms*;

Keywords

energy flexibility, heuristic optimization, grid tariff, prosumer coordination, battery scheduling, smart grids

ACM Reference Format:
Romaric Duvignau, Wania Khan, Marina Papatriantafilou, and Vincenzo Gulisano. 2026. SCALFLEX: A Scalable Heuristic Framework for Distributed Energy Flexibility and Grid Tariff Optimization. In *ACM Sustainability Week 2026 (ACM Sustainability Week Companion '26), June 22–25, 2026, Banff, AB, Canada.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3765611.3815592



This work is licensed under a Creative Commons Attribution 4.0 International License.
ACM Sustainability Week Companion '26, Banff, AB, Canada
© 2026 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-2199-1/26/06
https://doi.org/10.1145/3765611.3815592

1 Introduction

The rapid deployment of solar photovoltaic (PV) systems and battery energy storage systems (BESS), driven by the global energy transition, is transforming many households into *prosumers* capable of consuming, generating, and storing electricity. These resources provide flexibility by enabling households to shift consumption across time, for example by charging during low-price periods and discharging during peak demand [9]. At the same time, increasing electrification and the growing integration of inverter-based renewable resources are placing significant stress on distribution grids, creating a need for scalable coordination of distributed flexibility.

Motivations. Addressing limited grid capacity, excess generation, and peak demand through infrastructure expansion alone would require prohibitively high investments. Improving the utilization of existing infrastructure has therefore become a critical challenge. Dynamic electricity pricing, in particular flexible grid tariffs that adapt to demand conditions, has emerged as a promising mechanism to shift consumption and reduce peak loads. Compared to alternative flexibility mechanisms, grid tariffs are simple to implement, can be applied at scale, and provide direct economic incentives to end-users. However, designing effective tariffs is computationally challenging, as it requires anticipating the aggregated response of large populations of prosumers. In practice, this involves repeatedly solving large numbers of cost-optimization problems under different pricing signals, leading to thousands of tasks that must be processed within tight time constraints.

Contributions. To address the aforementioned challenges, we propose ScalFlex, a scalable heuristic framework for large-scale, repeated flexibility optimization tasks. By leveraging previously solved tasks, the framework efficiently approximates large volumes of cost-optimization problems while maintaining near-optimal solution quality. Our main contributions are: (1) a scalable framework that significantly improves throughput and response time compared to Linear Programming (LP) based optimization, and (2) an

Algorithm 1: *GridFlex* iterative tariff computation

Input : User set H, time horizon T, base grid tariff price α
Output: Grid tariffs $gt(t)$ for all $t \in T$

Initialize $gt(t) \leftarrow 0$ for all $t \in T$;
$bestPeak \leftarrow \infty, streak \leftarrow 0$;
repeat
  Run *BatFlex* for all $h \in H$ using grid tariffs gt;
  Compute aggregated demand $load(t)$ for all $t \in T$;
  $pd \leftarrow \max_{t \in T} load(t)$; // compute peak demand
  if $pd < bestPeak$ **then**
    $bestPeak \leftarrow pd, streak \leftarrow 0$;
  else if $pd > bestPeak$ **then**
    $streak \leftarrow streak + 1$;
  else
    break; // peak demand has not changed
  foreach $t \in T$ **do**
    $r \leftarrow load(t)/pd$;
    $gt(t)$ += $\alpha \cdot f(r)$; // where $f(r)$ is a step function with thresholds at $f(0.80) = 1/10$, $f(0.85) = 1/4$, $f(0.90) = 1/2$, $f(0.95) = 1$, and $f(0.99) = 2$; $f(r) = 0$ for $r < 0.8$.
until $streak \geq 3$;
return gt;

efficient approach for simulation-based computation of dynamic grid tariffs. The framework can support multiple optimization applications. We demonstrate its applicability through two use cases: (i) *BatFlex*, which computes battery operation decisions for large populations of prosumers, and (ii) *GridFlex*, which computes grid tariffs by iteratively simulating aggregate prosumer responses to pricing signals using *BatFlex*. While we focus on two use cases for brevity, the framework generalizes to other scenarios such as peer-to-peer energy sharing [1, 5, 6] and power-based cost optimization [4].

Outline. § 2 and § 3 start by presenting the studied cost optimization tasks handled by our framework. § 4 then details ScalFlex's design and architecture as well as the novel heuristic approach. § 5 evaluates its performance against a baseline under various configurations. § 6 discusses related work, and § 7 concludes the paper and outlines future directions.

2 Energy Applications

We present two applications used to evaluate ScalFlex under large-scale flexibility optimization workloads.

BatFlex. BatFlex provides battery optimization as a service to a large population of prosumers. Each user seeks optimal *battery usage decisions* (charge, discharge, or idle) to minimize electricity costs over a given time horizon. For each user, the application combines historical consumption data, PV and battery characteristics, electricity prices (e.g., day-ahead markets), and solar intensity derived from weather data. When required, inputs are forecast over the target horizon. Each user request corresponds to a separate optimization task, resulting in thousands of tasks that must be solved efficiently.

GridFlex. GridFlex (Alg. 1) supports DSOs in designing time-varying grid tariffs to mitigate peak demand. Grid tariffs act as local signals that complement wholesale market prices, helping to prevent congestion by incentivizing load shifting. Designing such tariffs is computationally challenging, as it requires anticipating the aggregated response of a large number of flexible users. As a usecase of our framework, we propose a simple strategy to dynamically design grid tariffs, namely an iterative approach assuming prosumer-level battery scheduling based on costs. In more details, starting from an initial tariff (e.g. no grid tariff), user responses are computed (via *BatFlex*), the resulting load profile is evaluated, and tariffs are adjusted to penalize peak periods using heuristically chosen steps. The objective is to minimize aggregated peak demand (after the tariffs are applied) while avoiding unnecessary price increases. The number of iterations is unbounded as long as peak demand continues to decrease. Following our design, *GridFlex* provides a representative example of a strategy requiring the computation of large numbers of optimization tasks.

3 Energy Cost Optimization

We formalize the core problem addressed by ScalFlex as a set of *optimization tasks*. We first introduce their definition, then present an *LP* formulation in § 3.2 to compute optimal decisions. We describe in § 3.3 a *greedy* approach [2, 7] commonly used to reduce computation time. LP will serve as baseline, while our approach falls back to greedy decisions if needed.

3.1 Optimization Tasks

A *cost-optimization task* determines battery usage decisions for a prosumer over a time horizon discretized into intervals (e.g., 15 minutes or 1 hour). Each task is defined by time series data including electricity consumption, solar generation, market prices, and battery constraints.

The objective is to minimize the electricity cost, defined as (refer to Table 1 for the definition of all variables):

$$\begin{aligned} cost(h,t) = {} & el_{buy}(h,t) \cdot (price(t) \cdot tax + p_{tax} + gt(t)) \\ & - el_{sell}(h,t) \cdot (price(t) + p_{net}) \end{aligned} \tag{1}$$

For clarity, we omit subscription fees or peak-based grid tariffs, as our focus is on computational scalability.[1]

The electricity balance is defined as $el_{bal}(h,t) = el_{cons}(h,t) - el_{gen}(h,t)$, where electricity generation is derived from the solar intensity profile and PV capacity as $el_{gen}(h,t) = sun(h,t) \cdot PV_h$. In the absence of storage, the problem is trivial: consumption is directly matched by grid transactions. With PV but no battery, excess generation is either consumed locally or sold to the grid. The optimization problem becomes non-trivial when a battery is available, as decisions must balance storage, consumption, and grid interaction over time.

3.2 Linear Programming (LP) formulation

We formulate the problem as a *Linear Program*, following [5–7], to compute optimal battery decisions $\{bat_h(t)\}_{t \in T}$ over a time horizon $T = [t_{start}, t_{end}]$ for a prosumer h:

- **Objective:** minimize $\sum_{t \in T} cost(h,t)$;
- **Decision variables:** $\{bat_h(t), el_{buy}(h,t), el_{sell}(h,t)\}_{t \in T}$;
- **Constraints:**
  (1) $\forall t \in T$, $bat_h(t) \geq 0$, $el_{buy}(h,t) \geq 0$, $el_{sell}(h,t) \geq 0$,

[1] More complex pricing schemes would further increase computation time and thus strengthen the benefits of our heuristic-driven approach.

Table 1: Nomenclature; all variables are non-negative except for the cost, possibly negative for net sellers.

$cost(h,t)$	Electricity cost (€) for user h at time t.
$el_{sell}(h,t)$	Electricity sold (kWh) to the grid by user h at time t.
$el_{buy}(h,t)$	Electricity bought (kWh) from the grid by h at time t.
$el_{grid}(h,t)$	Grid interaction (kWh): $el_{buy}(h,t) - el_{sell}(h,t)$.
$sun(h,t)$	Solar intensity (kWh/kWp) for user h at time t.
$el_{gen}(h,t)$	Electricity generated (kWh) by user h at time t.
$el_{cons}(h,t)$	Electricity consumed (kWh) by user h at time t.
$el_{bal}(h,t)$	Electricity balance (kWh) for user h at time t.
$price(t)$	Electricity price (€/kWh) at time t.
$gt(t)$	Grid tariff (€/kWh) at time t, 0 if not used.
PV_h	PV capacity (kWp) of prosumer h.
$bat_h(t)$	Battery level (kWh) of prosumer h at time t.
B_h	Battery capacity (kWh) of prosumer h.
ρ_h^-, ρ_h^+	Maximum battery dis-/charging (kWh per timestep).
p_{tax}	Fixed electricity tax (€/kWh).
p_{net}	Revenue (€/kWh) from selling electricity.
tax	Relative tax factor applied to the market price.

(2) Battery bounds and charging limits:
- $0 \le bat_h(t) \le B_h$,
- $-\rho_h^- \le bat_h(t) - bat_h(t-1) \le \rho_h^+$,

(3) Energy balance:

$$bat_h(t) = bat_h(t-1) + el_{bal}(h,t) + el_{buy}(h,t) - el_{sell}(h,t),$$

with initial battery level $bat_h(t_{start} - 1)$ given as input.

3.3 Greedy Decisions

A common alternative is a greedy (myopic) strategy [2], which prioritizes local battery usage over grid interaction. At each timestep, consumption is first met using stored energy with surplus energy stored in the battery or deficit energy taken from the battery, and the grid is used only when necessary. The battery evolution is defined as:

$$bat_h(t) = \begin{cases} \max(0,\ bat_h(t-1) - el_{bal}(h,t)), & \text{if } el_{bal}(h,t) \ge 0, \\ \min(B_h,\ bat_h(t-1) - el_{bal}(h,t)), & \text{otherwise.} \end{cases}$$

Grid interactions follow from the battery change, with surplus energy sold (when the battery is full) and deficits purchased (when the battery is empty). While computationally efficient, this approach ignores current and future price variations and may lead to suboptimal decisions.

4 A Scalable Heuristic Framework

We present ScalFlex, a scalable heuristic framework for efficiently solving large volumes of cost-optimization tasks. § 4.1 provides an overview of the framework, § 4.2 describes its implementation, and § 4.3 details the proposed heuristic.

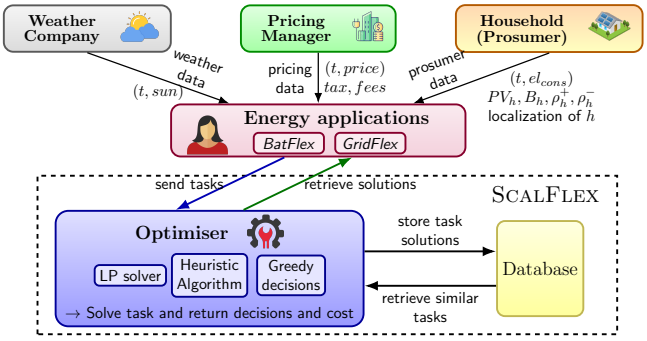


Figure 1: ScalFlex's design with data sources (handled by the applications using the optimization framework).

4.1 System Overview

ScalFlex (cf. Fig. 1) processes cost-optimization tasks by combining external data sources with a scalable optimization backend. Each task computes optimal flexibility decisions for a prosumer based on time-dependent inputs such as consumption, generation, and pricing. The framework relies on three main data sources: (i) weather data, providing solar intensity time series to estimate PV generation; (ii) price data, including electricity prices and/or grid tariffs; and (iii) household data, describing consumption profiles as well as PV, battery capacities and charging limits. Architecturally, ScalFlex comprises two main components: a database that stores previously solved tasks and their solutions for reuse, and an optimizer that processes tasks using either exact (LP) or heuristic methods (§ 4.3).

4.2 System Architecture

We implement ScalFlex as a client–server system in Python (v3.8), designed for high-throughput processing of optimization tasks. As illustrated in Fig. 2, the system leverages multiprocessing to enable concurrent task handling. Applications (e.g., *BatFlex* or *GridFlex*) act as **clients**, generating cost-optimization tasks ("opt. tasks") for individual prosumers. Each task encapsulates the required inputs, including consumption and generation profiles, as well as PV and battery characteristics. The **server** orchestrates task processing. Clients maintain two parallel threads: one for submitting tasks to a shared problem queue and another for retrieving solutions from a shared solution queue. Both queues enable efficient communication between clients and the server and are implemented using Redis (https://redis.io/). Task execution is handled by multiple **worker processes**. Each worker retrieves a task pb_h from the queue and determines how to solve it based on the contents of the database. If no prior solutions are available, the task is solved using [5], i.e., an LP solver relying on `pyomo`'s wrapper over `cbc` solver[2] by default. Otherwise, the worker applies the proposed heuristic method (Alg. 2), leveraging previously computed solutions to accelerate computation.

[3]

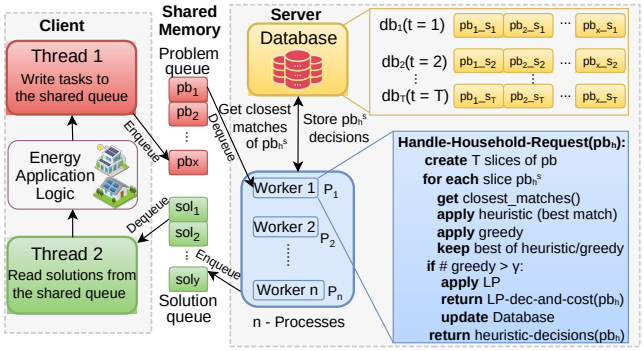


Figure 2: ScalFlex's software architecture.

Algorithm 2: Handle-Household-Request(pb_h)

Input : pb_h optimization task of a household h.
Global variable: pb_db sorted lists of solved slices with solution.
Parameters : K', K, γ, W_S, W_A (cf. Table 2).
Output : $cost_h$, best_dec cost/decisions for household h.

dec_seq, best_dec, eu_list ← {}, {}, {}; // empty lists
slice_cost ← ∞, init_bat ← 0;
/* Calculate the total number of windows to create */
total_windows ← len(pb_h)/W_A, bat_level ← init_bat;
if pb_db = {} **then**
  return solve-with-lp(pb_h);
for i = 0 **to** *total_windows* **do**
  start, end ← i * W_A, min(start + W_S, len(pb_h));
  db ← extract-segment(pb_db,start, end) ;
  /* Create slice of pb_h for the desired period */
  pb_h_s ← create-slice(pb_h, start, end); // s-th slice of pb_h
  target_netload ← compute-netload(pb_h_s);
  $pb_h_s.avg$ ← compute-average-netload(pb_h_s);
  candidates ← closest-matches(db, K', $pb_h_s.avg$);
  foreach cand ∈ candidates **do**
    cand_netload ← compute-netload(cand);
    dist ← euclidean-dist(target_netload, cand_netload);
    eu_list.append(cand, dist);
  /* Select K candidates based on euclidean distance */
  top_cand ← select-top(K, eu_list, dist)
  foreach s_cand ∈ top_cand **do**
    /* Calculate battery levels for pb_h slice */
    bat_dec ← calculate-battery-level(s_cand, pb_h_s);
    /* Compute decisions using battery levels */
    result ← compute-decisions(pb_h_s, bat_dec, bat_level)
    if result ≤ slice_cost **then**
      slice_cost ← result;
      dec_seq ← bat_dec;
  if dec_seq ≠ {} **then**
    (greedy_cost, greedy_dec) ← greedy_algo(pb_h_s);
    if greedy_cost < slice_cost **then**
      dec_seq ← greedy_dec;
      nb_greedy_used ← nb_greedy_used +1;
    best_dec.append(dec_seq[start, min(start + W_A, end)]);
  if nb_greedy_used ≥ γ **then**
    /* LP is applied when γ threshold is exceeded */
    return solve-with-lp(pb_h);
  bat_level ← get-last-element(best_dec);
return compute-decisions-cost(pb_h, best_dec, init_bat), best_dec;

Table 2: ScalFlex parameters used in the evaluation.

Parameter	Description	Values
K'	Candidate end-users	25, 50, 100, 150, 200
K	Selected end-users	5, 20, 40, 60, 80
γ	Greedy threshold	0.3, 0.4, 0.5, 0.6
W_S	Window size	48
W_A	Window advance	24

4.3 Heuristic-based approximation

ScalFlex accelerates task processing by reusing previously computed solutions for similar optimization tasks, identified based on their load profiles. Solved optimization tasks are stored in a database segmented by time period to ensure consistency with time-varying prices. Given a new task pb_h of length $\texttt{len}(pb_h) = |T|$, the framework retrieves similar past tasks based on the average net load over the T timesteps:

$$pb_h.avg = \frac{1}{|T|} \sum_{t \in T} el_{bal}(h, t).$$

To better capture temporal variations, the task is divided into shorter overlapping slices using a sliding window of size W_S and advance W_A. Each slice is processed independently using candidate solutions from the corresponding database segment.

Approximate solution. For each slice, a two-step selection identifies similar past solutions, i.e., to save $O(\texttt{len}(pb_h))$ distance computations, K' candidates are first retrieved using average net load (via a sorted index); then the K closest are selected based on Euclidean distance between load profiles. The selection is performed using unnormalized data, which yielded better results in our experiments. For each candidate h', battery trajectories are adapted by scaling to the target capacity while ensuring charging bounds, setting $bat_h(t)$ as:

$$\max\left\{\min\left\{\frac{bat_{h'}(t) \cdot B_h}{B_{h'}},\ bat_h(t-1) + \rho_h^+\right\},\ bat_h(t-1) - \rho_h^-\right\}.$$

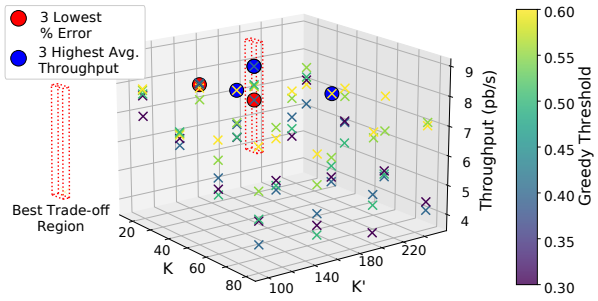
Battery difference is then derived and grid interaction follows: $el_{grid}(h,t) = el_{bal}(h,t) + bat_h(t) - bat_h(t-1)$. The resulting cost is computed using Equation 1, and the best candidate is retained. To improve robustness, the greedy solution (§ 3.3) is also evaluated for each slice, and the lower-cost option is selected. The window then advances and the process repeats for all slices. If the fraction of slices solved using greedy decisions exceeds a threshold γ, the entire task is recomputed using the LP solver; otherwise, the assembled heuristic solution (cost and decisions) is returned.

Database update. Whenever a task is solved using the LP solver (e.g., when the database is empty or upon heuristic failure), it is decomposed into slices and the corresponding optimal solutions are stored in the database for future reuse.

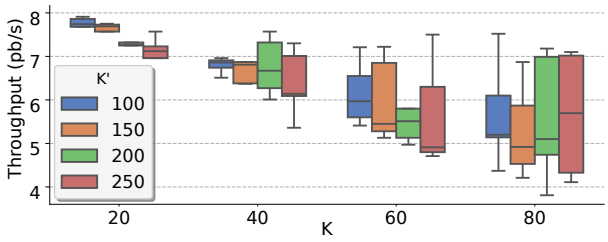
5 Evaluation

We evaluate ScalFlex against an LP-only baseline (which corresponds to ScalFlex with the heuristic method deactivated) on

[2] COIN-OR Branch and Cut solver, https://github.com/coin-or/cbc.



(a) Avg. throughput (pb/s) for various K, K', and γ.



(b) Avg. throughput (pb/s) for different K and K'; $\gamma = 0.5$.

Figure 3: ScalFlex throughput evaluation for various configurations; client rate of 8 pb/s, 1 month of data per opt. task., 2 workers.

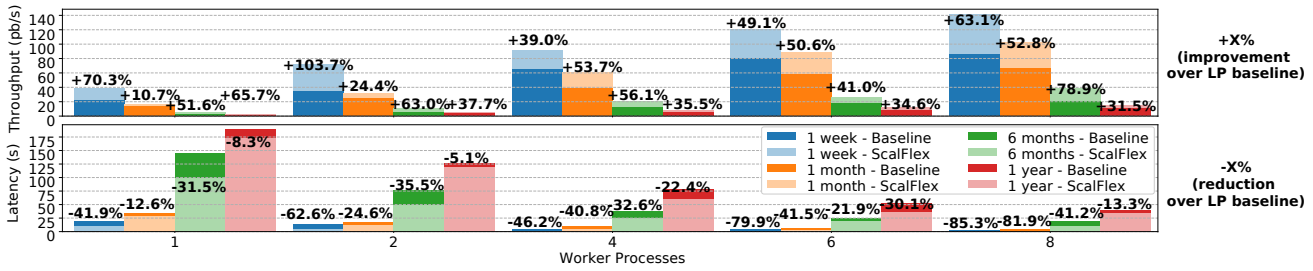


Figure 4: ScalFlex throughput (pb/s) and end-to-end latency (s) for different opt. task lengths and worker processes.

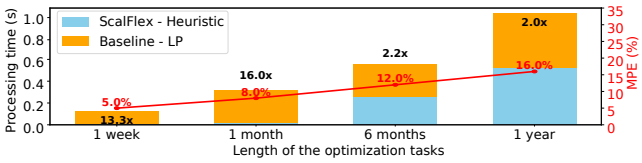


Figure 5: Avg. processing time and MPE between ScalFlex (heuristic) and baseline (LP); client rate 200 pb/s, 2k opt. tasks. (left axis processing time, right axis MPE)

large sets of cost-optimization tasks of different lengths, focusing on computational efficiency and solution quality.

5.1 Experimental Setup

Experimental platform. Experiments are conducted on comparable commodity hardware representative of practitioner environments: a 12-core CPU with 16,GB RAM for *BatFlex* and a 14-core CPU with 32,GB RAM for *GridFlex* (different systems were used due to hardware availability).

Data. We use real consumption data from 2,221 households [7], all equipped with PV and battery systems, following [6, 7]; hourly timesteps are considered with $\rho_h^+ = \rho_h^- = B_h$. We refer to [6] for set prices, taxes and fees. To avoid additional forecasting errors, historical data is used as input to *BatFlex*. We vary (i) opt. task length (48h to 1 year), (ii) the number of tasks (500 to 100k) and (iii) ScalFlex parameters (cf. Table 2). A configurable client rate controls task ingestion speed.

5.2 Performance Metrics

We evaluate: **Throughput (pb/s):** no. of tasks (problems) solved per second; **Latency (s):** end-to-end response time; **Processing time (s):** server-side execution time; **Solution quality:** Mean Percentage Error (MPE) vs. LP optimum.

5.3 Results

5.3.1 BatFlex application. Parameter selection. As shown in Fig. 3a, lower values of K and K' and higher reliance on the heuristic (larger γ), improve throughput, while larger candidate sets improve accuracy at the cost of computation time. Fig. 3b further shows that increasing K and K' expands the search space and reduces throughput. Parameters are thus empirically selected ($K = 20$, $K' = 200$, greedy threshold $\gamma = 0.5$) to balance throughput and accuracy.

Efficiency vs. accuracy. Figures 4–6 use overlapping bars for the baseline and ScalFlex to emphasize relative improvements, with percentage gains shown above the bars. ScalFlex consistently achieves higher throughput and lower latency than the baseline across all task sizes (Fig. 4). Throughput decreases with task size due to higher computational load and the increased difficulty of approximating long-horizon decisions. As shown in Fig. 5, ScalFlex achieves up to 16× faster processing for medium-sized tasks (1 month). For larger horizons, the speedup decreases (to ∼ 2× for yearly tasks) due to increased complexity and additional slice processing and database lookups. This gain comes with a trade-off: MPE increases with task size, reaching ∼ 16% for yearly tasks, reflecting the limits of decision reuse over long time horizons.

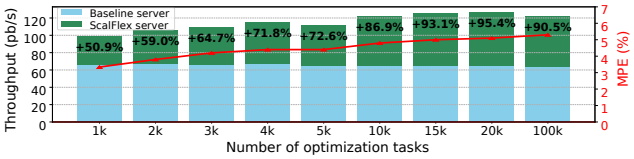


Figure 6: ScalFlex throughput (pb/s) (with % increase over baseline) for different number of optimization tasks.

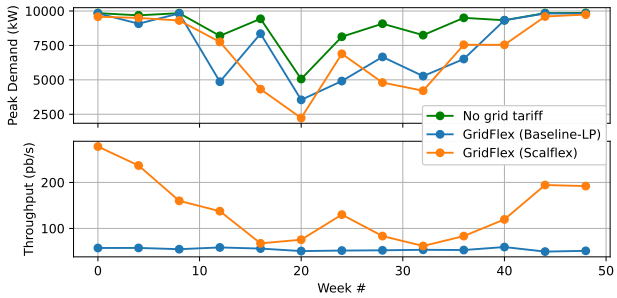


Figure 7: Calculated peak demand by *GridFlex* and system throughput; 500 end-users with 1 week of data each.

Scalability. Fig. 6 shows that ScalFlex scales effectively with the number of tasks, reaching ~ 120 pb/s (+90% over the baseline). As the workload increases, the database of prior solutions grows, improving the effectiveness of the heuristic and reducing reliance on LP solves. Despite this, solution quality remains high, with MPE staying below ~ 5% even at 100k tasks.

5.3.2 GridFlex application. For *GridFlex* (Alg. 1), we use ScalFlex with $K = 5$, $K' = 20$, $\gamma = 0.66$, 12 workers and $\alpha = el_{tax}$. Fig. 7, presenting the peak demand without grid tariffs and the peak demand obtained with LP and ScalFlex, shows that for comparable performance in reducing peak demand (average reduction of -17.2% for LP and -21.6% for ScalFlex), ScalFlex achieves between 20% and 380% higher throughput. For a shorter time window of 48h (Fig. 8), the throughput gains are even more pronounced (10–18× improvement for 500 or more end-users), while the relative peak demand difference between the two was always at most 0.2% under this setting.

6 Related Work

A large body of work addresses cost optimization and flexibility management in residential systems, including appliance scheduling, battery management, and renewable integration [1, 11]. Many approaches rely on forecasting techniques to anticipate prices or demand and optimize consumption accordingly [8]. LP formulations are commonly used to compute optimal decisions for energy dispatch and storage [3], but their computation cost grows rapidly with problem size. Distributed and decomposition-based optimization methods, including ADMM-based approaches, have also been proposed to improve scalability in flexibility coordination scenarios [10]. In parallel, recent work has explored coordination mechanisms such as peer-to-peer energy sharing and community-level

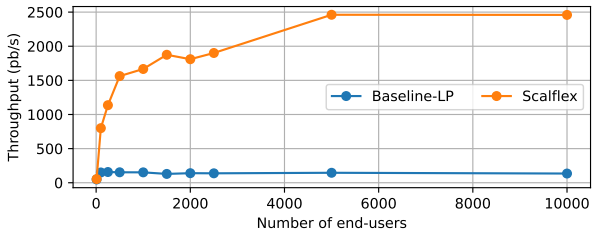


Figure 8: Application throughput by *GridFlex*; 10-10k end-users with 48 hours of data each.

optimization [2, 7]. While these approaches demonstrate the benefits of coordinated flexibility, they also highlight a key limitation: solving large numbers of optimization problems remains computationally challenging. Our approach centers on reusing previously computed optimal decisions instead of relying solely on distributed iterative optimization or greedy heuristics.

7 Conclusion

This paper presented ScalFlex, a scalable heuristic framework that improves the efficiency of cost-optimization tasks by reusing previously computed LP solutions. Beyond individual optimization (*BatFlex*), ScalFlex enables iterative design of grid tariffs (*GridFlex*), supporting large-scale flexibility coordination. Experiments show that ScalFlex achieves orders of magnitude higher throughput than an LP baseline, while maintaining near-optimal accuracy (within 3-5%). Therefore, computation-aware heuristics can enable scalable flexibility optimization. Future work includes extending the framework to multi-user coordination and investigating how *GridFlex* can support additional grid objectives.

Acknowledgments

This project is financed through the Swedish Electricity Storage and Balancing Centre (SESBC). The center is funded by the *Swedish Energy Agency* together with five academic and twenty-eight non-academic partners. In particular, the authors acknowledge Göteborg Energi for their contribution and support to the TANDEM project.

References

[1] Muhammad Raisul Alam, Marc St-Hilaire, and Thomas Kunz. 2016. Computational methods for residential energy cost optimization in smart grids: A survey. *ACM Computing Surveys (CSUR)* 49, 1 (2016), 1–34.

[2] Sid Chi-Kin Chau, Jiajia Xu, Wilson Bow, and Khaled Elbassioni. 2019. Peer-to-peer energy sharing: Effective cost-sharing mechanisms and social efficiency. In *Proceedings of the 10th ACM International Conference on Future Energy Systems*. 215–225.

[3] Sridhar Chouhan, Deepak Tiwari, Hakan Inan, Sarika Khushalani-Solanki, and Ali Feliachi. 2016. DER optimization to determine optimum BESS charge/discharge schedule using Linear Programming. In *IEEE PES General Meeting*.

[4] Romaric Duvignau, Vincenzo Gulisano, and Marina Papatriantafilou. 2023. Cost-optimization for win-win P2P energy systems. In *IEEE Power & Energy Society Innovative Smart Grid Technologies (ISGT)*. IEEE, 1–5.

[5] Romaric Duvignau, Vincenzo Gulisano, and Marina Papatriantafilou. 2025. PyPESOL: The Python P2P Energy Sharing Optimization Library. In *Proceedings of the 16th ACM International Conference on Future and Sustainable Energy Systems*. 1014–1015.

[6] Romaric Duvignau, Vincenzo Gulisano, Marina Papatriantafilou, and Ralf Klasing. 2024. Geographical peer matching for P2P energy sharing. *IEEE Access* 13 (2024), 9718–9738.

[7] Romaric Duvignau, Verena Heinisch, Lisa Göransson, Vincenzo Gulisano, and Marina Papatriantafilou. 2021. Benefits of small-size communities for continuous cost-optimization in peer-to-peer energy sharing. *Applied Energy* 301 (11 2021), 117402.

[8] Ghulam Hafeez, Khurram Saleem Alimgeer, Zahid Wadud, Imran Khan, Muhammad Usman, Abdul Baseer Qazi, and Farrukh Aslam Khan. 2020. An innovative optimization strategy for efficient energy management with day-ahead demand response signal and energy consumption forecasting in smart grid using artificial neural network. *IEEE Access* 8 (2020), 84415–84433.

[9] Jason Leadbetter and Lukas Swan. 2012. Battery storage system for residential electricity peak demand shaving. *Energy and Buildings* 55 (12 2012), 685–692.

[10] Pol Olivella-Rosell et al. 2020. Centralised and distributed optimization for aggregated flexibility services provision. *IEEE Transactions on Smart Grid* 11, 4 (2020), 3257–3269.

[11] Sahar Rahim, Nadeem Javaid, Ashfaq Ahmad, Shahid Ahmed Khan, Zahoor Ali Khan, Nabil Alrajeh, and Umar Qasim. 2016. Exploiting heuristic algorithms to efficiently utilize energy management controllers with renewable energy sources. *Energy and Buildings* 129 (2016), 452–470.