

Improving the Optimization in Model Predictive Controllers: Scheduling Large Groups of Electric Vehicles

Leoni Winschermann, Marco E.T. Gerards, Johann Hurink
 Department of Electrical Engineering, Mathematics and Computer Science
 University of Twente
 Enschede, The Netherlands
 L.Winschermann@utwente.nl

Abstract—In parking lots with large groups of electric vehicles (EVs), charging has to happen in a coordinated manner, among others, due to the high load per vehicle and the limited capacity of the electricity grid. To achieve such coordination, model predictive control can be applied, thereby repeatedly solving an optimization problem. Due to its repetitive nature and its dependency on the time granularity, optimization has to be (computationally) efficient.

The work presented here focuses on that optimization subroutine, its computational efficiency and how to speed up the optimization for large groups of EVs. In particular, we adapt FOCS, an algorithm that can solve the underlying optimization problem, to better suit the repetitive set-up of model predictive control by adding a pre-mature stop feature. Based on real-world data, we empirically show that the added feature speeds up the median computation time for 1-minute granularity by up to 44%. Furthermore, since FOCS is an algorithm that uses maximum flow methods as a subroutine, the impact of choosing various maximum flow methods on the runtime is investigated. Finally, we compare FOCS to a commercially available solver, concluding that FOCS outperforms the state-of-the-art when making a full-day schedule for large groups of EVs.

Index Terms—algorithm, optimization, computational performance, real data, electric vehicle

I. INTRODUCTION

With the on-going transition to electric mobility, the number of electric vehicles (EVs) in the Netherlands is increasing rapidly [1]. Especially near-work locations have been found to be good locations to charge large groups of EVs, partially due to the long stays [2] and the potential availability of solar energy [3]. However, due to limited grid capacity [4], high synchronicity [5] and possible temporal mismatches with said solar energy production [6], charging has to occur in a coordinated manner. For successful coordination, a controller needs information on the charging sessions, for example the (expected) energy demand and departure time of an EV in the parking lot. In practice, the EVs currently do not communicate such information [7]. Therefore, next to the sheer amount of charge required per vehicle, the biggest challenge in managing the charging processes of the EVs is the absence of information.

To bridge this information gap, model predictive control (MPC) can be applied (e.g., [8]). Based on the information available at the current point in time and an underlying

system model, MPC generates and solves a suitable offline optimization problem and an (in time) initial portion of this solution is realized. This process is repeated frequently over the given time horizon. Specifically to solve this offline problem, the Flow-Based Offline Charging Scheduler (FOCS) has been developed [9]. FOCS is a polynomial time algorithm which in theory has a complexity of $\mathcal{O}(n^2\mu)$, where n is the number of EVs to be charged and μ is the complexity of the maximum flow algorithm embedded in FOCS.

Notably, research has shown that for some algorithms theoretical and empirical runtime do not align. For example, the simplex algorithm, while theoretically exponential in runtime, performs well in practice. The other way around, Ellipsoid methods for linear programs are known to be polynomial on paper, while performing poorly in practice. This motivates us to investigate the empirical runtime of the novel algorithm FOCS based on real world EV data.

The main contributions of the work presented here are:

- Introduction and validation of a pre-mature stop feature that increases computational efficiency in model predictive controllers using FOCS.
- Extensive comparative runtime analysis of FOCS, and its validation using real world EV charging data.

The remainder of the paper is organized as follows. Section II discusses the mathematical model of the considered EV scheduling problem (Section II-A), briefly introduces FOCS (Section II-B) and proposes a pre-mature stop feature for FOCS (Section II-C) expected to increase its efficiency for MPC applications. Then, Section III specifies the experimental setup and gives some information on the implementation of the used algorithms. Furthermore, the section describes the real world data set used for the experiments (Section III-C). Section IV presents the experimental results, followed by a discussion in Section V and a conclusion in Section VI.

II. EV SCHEDULING PROBLEM AND ALGORITHMS

In this section, we describe the considered EV scheduling problem in terms of notation, constraints and considered objective function. Furthermore, we briefly introduce FOCS, an algorithm that can be used to determine an optimal solution for said problem. For more details on FOCS, we refer to [9].

A. Problem definition

We consider jobs $1, \dots, n$, where each job j corresponds to a pending charging session (or EV) with associated energy requirement e_j , arrival time a_j , departure time d_j and job-specific charging power limit p_j^{\max} . We denote the set of all jobs by $[n]$. We consider a problem instance feasible if

$$e_j \leq p_j^{\max}(d_j - a_j) \quad \forall j \in [n].$$

Moreover, we discretize the time horizon $[\min_j a_j, \max_j d_j]$ into atomic intervals by using all arrival and departure times of EVs as breakpoints. The resulting (ordered) sequence of breakpoints we denote as t_1, \dots, t_{m+1} , resulting in $m \leq 2n - 1$ atomic intervals $I_i = [t_i, t_{i+1}]$ for $i = 1, \dots, m$.

To relate jobs to atomic intervals, we denote by $J(i)$ the jobs that are available in interval I_i , i.e.,

$$J(i) = \{j | (a_j \leq t_i) \wedge (t_{i+1} \leq d_j)\}.$$

Similarly, $J^{-1}(j)$ is defined as the set of indices i of the intervals I_i where j is available. Finally, as atomic intervals are in general not unit-sized, we introduce maximum energy limits $e_{i,j}^{\max} = p_j^{\max}(t_{i+1} - t_i)$ per job j and interval I_i .

As decision variables, let $e_{i,j}$ be the energy that EV j charges in I_i . Note that preemption is allowed, meaning the charging can be suspended for a few intervals to be continued at a later interval. In this work, we do not consider V2G, implying that $e_{i,j} \geq 0$.

Summarizing, the EV model is constrained by

$$\sum_{i \in J^{-1}(j)} e_{i,j} \geq e_j \quad \forall j \in [n] \quad (1a)$$

$$e_{i,j} \geq 0 \quad \forall j \in [n], i \in J^{-1}(j) \quad (1b)$$

$$e_{i,j} \leq e_{i,j}^{\max} \quad \forall j \in [n], i \in J^{-1}(j). \quad (1c)$$

From a grid perspective, the aggregated power level resulting from an EV schedule is of interest. For a given schedule, the average aggregated power level p_i in atomic interval I_i is given by

$$p_i = \frac{\sum_{j \in J(i)} e_{i,j}}{t_{i+1} - t_i}.$$

The two most frequently considered objectives for the overall grid usage are the ℓ_∞ and ℓ_2 -norms of the aggregated power. FOCS was developed for minimization of objective functions

$$F(p_1, \dots, p_m) \quad (2)$$

that are convex, differentiable and for which increasing any value p_i increases the value of the objective function. The complexity of FOCS is independent on the exact form of the objective function. In this work, we specifically consider the square of the ℓ_2 -norm given by

$$F(p_1, \dots, p_m) = \sum_{i=1}^m L_i \cdot (p_i^2) \quad (3)$$

where L_i is a normalization term to account for the length of the interval.

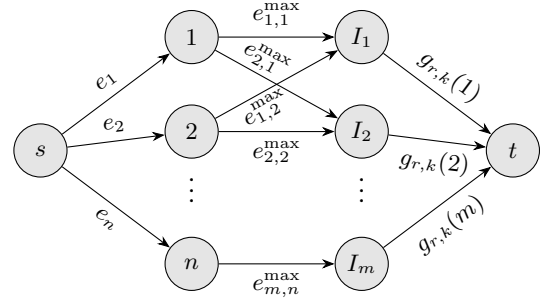


Fig. 1: Schematic of flow network structure of EV charging schedule.

B. The general working of FOCS

FOCS is a recursive algorithm that finds an optimal solution to the problem described above. To this end, it makes use of various properties of optimal solutions to this problem. First, any optimal aggregated power profile that minimizes an objective function (2) is a step function in the sense that within each atomic interval, the aggregated power is constant. Based on this, FOCS iteratively identifies a critical interval where the highest speed is needed, determines the minimum speed for this interval, and fixes the schedule for this interval, ultimately minimizing the maximum peak. The selected critical interval and the charge provided in that interval are then removed from the problem instance.

To determine the critical intervals, FOCS computes multiple iterations of maximum flows. Figure 1 depicts the basic structure of the considered flow network. Here, at each iteration, FOCS adapts the capacities of edges adjacent to sink node t . The iterations until a critical interval is identified, form a round r . The edge capacities $g_{r,k} : \{I_i | i \in [m]\} \rightarrow \mathbb{R}_{\geq 0}$ in round r and iteration k represent a fill-level that is increased over the iterations until critical interval(s) are found.

C. Adapting FOCS for model predictive control

FOCS was developed to determine the optimal solution of the offline EV scheduling problem described in Section II-A. To this end, it solves the instance over the entire planning horizon. However, for MPC, often only the next (few) time intervals and their control actions are of interest. Based on this, we propose an adaptation to FOCS. For convenience, we assume in the following that only the first time interval is of interest. The proposed concept extends naturally to include the first few intervals.

FOCS recursively identifies intervals that require the highest aggregated power in an optimal schedule (so-called critical intervals), determines the partial schedule corresponding to that interval, and repeats this with the remaining problem instance. In particular, each of those recursion steps fixes a part of the optimal schedule. Therefore, if we are only interested in the schedule of the first interval, we can pre-maturely terminate FOCS once the schedule for that interval was found, without losing any information on the first interval.

We illustrate this concept in Figure 2 for an instance with two EVs and three intervals. The annotations at the edges of the flow network are of the form x/y , where x is the

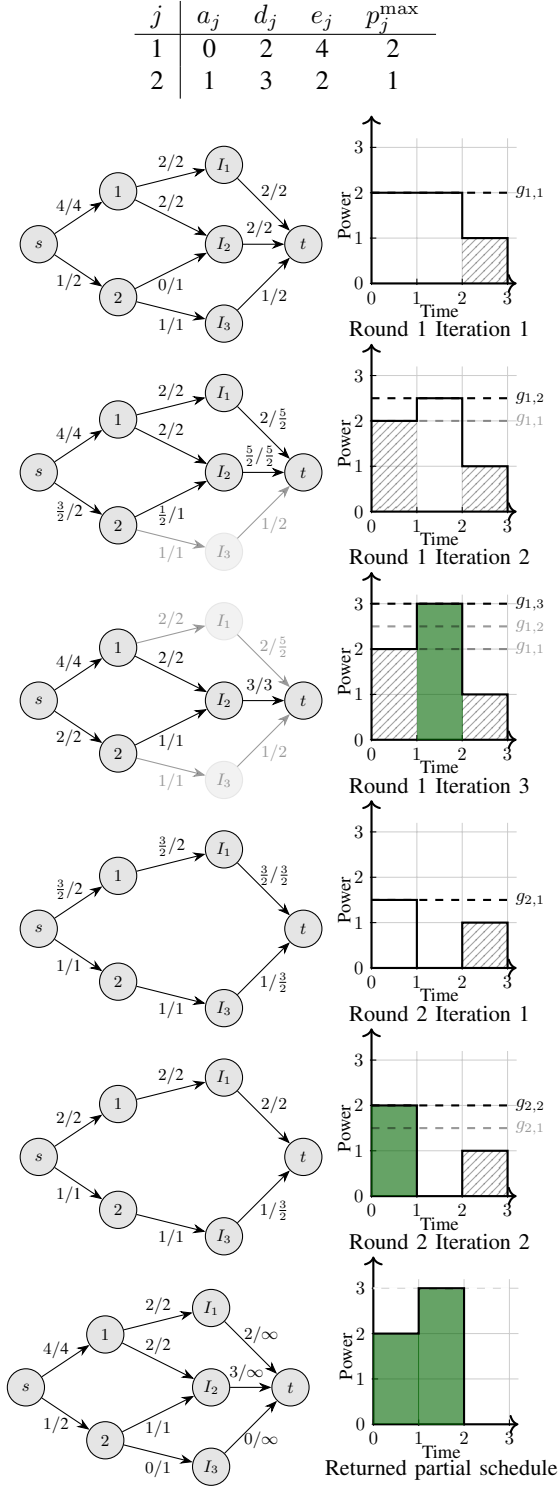


Fig. 2: Intermediate states of FOCS with pre-mature stop for an example instance, tracked over rounds and iterations.

flow going through the edge, and y is the edge capacity. Moreover, the green shaded blocks are the identified critical intervals. Those are fixed, and in the last step combined into the algorithm output. After finding the schedule for the first interval (i.e., when we find the green block for the first interval in the fifth step in Figure 2), we terminate the algorithm. Note that even though the schedule for the third interval has not yet been found and thereby the returned schedule (last power profile in Figure 2) is not feasible considering the entire problem instance, there exists an optimal schedule that complements the found partial solution. In particular, the aggregated power in the first interval is optimal. This follows by optimality of FOCS, and the following lemmas paraphrased from [9].

Lemma 1: (Lemma 2 in [9]) If a critical interval I_i and its aggregated power are found, and if there is at least one interval $I_{i'}$ left that at that point has not yet been scheduled, then no scheduled charging work can feasibly be moved from I_i to $I_{i'}$ for any such $I_{i'}$.

Lemma 2: (Lemma 3 in [9]) The aggregated power of the subsequently identified (sets of) critical intervals by FOCS is strictly decreasing. In particular, if FOCS fixes the schedule for I_i before that of $I_{i'}$, then

$$p_i > p_{i'}$$

in the optimal solution.

For the precise formulation and proof of the lemmas, we refer the reader to [9].

Notably, for EV parking lots near office locations, the optimal schedule over a day typically resembles a concave curve. The pre-mature stop feature we propose here terminates the algorithm once the first interval has been scheduled, where intervals are scheduled by order of their aggregated power in the optimal solution. Therefore, the time of day at which we apply the pre-mature stop impacts its effectiveness. In particular for an office parking lot, we expect the greatest reduction in computation time with pre-mature stop compared to the unrestricted FOCS to be around noon. In the following, we refer to the implementation of FOCS with pre-mature stop as FOCS-pm.

III. METHODS

In this section, we provide some details on the implementation of the three tested optimization models (Quadratic program in Gurobi (QP), FOCS and FOCS with pre-mature stop (FOCS-pm)), after which we describe the experimental setup and real-world data set used.¹

A. QP implementation

Based on the problem constraints (1) and the quadratic objective function (3), we use the commercial solver Gurobi [10] to solve this problem. Gurobi is used for comparison with state-of-the-art solvers. Note that while FOCS leads to optimal

¹The source code with all implementations and the experimental setup can be found under <https://github.com/lwingschermann/FlowbasedOfflineChargingScheduler>.

solutions also for more general objective functions, for practical reasons we compare its runtime to a Gurobi model and for that we restrict us to a quadratic objective function.

B. FOCS implementation

The FOCS implementation used for the empirical runtime analysis of FOCS has been developed specifically as a proof of concept for this paper. The code is written in python, heavily relying on the `networkx` package [11]. This package is chosen for its user friendly flow models, and because various maximum flow algorithms are readily available within the package. An overview of the considered algorithms is provided in Table I. Next to that, we also implemented the adapted FOCS with pre-mature stop as introduced in Section II-C by adding a feature that stops the algorithm once the relevant parts of the schedule are computed. In the following, we refer to FOCS-pm for results achieved with this pre-mature stop.

C. Dataset

The experiments described in this paper use data collected at a real life EV charging parking lot in Utrecht, the Netherlands [13]. The data was collected between September 1st 2022 and August 31st 2023, resulting in a total of 13694 charging sessions. Each session specifies (among others) an arrival time, departure time and total energy charged during the session. At the parking lot, most charging stations have two sockets each. If only one socket is occupied, the maximum charging power is 22 kW. Else, it is 11 kW per socket. Therefore, for the experiments, we set the maximum charging power per session to either 11 kW or 22 kW, depending on whether the average observed charging power is below or above 11 kW. From the 13694 sessions in the data, a maximum of 113 sessions was recorded in a single day.

D. Numerical experiments

The main focus of this paper is on optimization in MPC for EV charging parking lots and in particular its efficiency. To this end, we present numerical experiments based on real world data, evaluating the empirical runtime of FOCS and FOCS-pm compared to the benchmark results by Gurobi.

The main parameters to classify the experiments are the

- instance size n , i.e., the number of EV charging sessions;
- time granularity, i.e., either 1 minute, 15 minutes, 30 minutes or 1 hour;
- used maximum flow method in FOCS (and FOCS-pm).

Given a set of parameters for the experiment, we randomly sample n sessions from the dataset described in Section III-C and apply all three optimization models. We record the CPU runtime using the function `time.process_time()` in-built in the python package `time`. In particular, we record the CPU runtime for building the models (QP, FOCS, FOCS-pm) and solving them. For a given model and experiment, the total runtime is the sum of CPU time taken to build and solve the model.

As pointed out in Section II-C, FOCS-pm is most effective when the first interval is expected to have one of the highest

power peaks in the optimal profile. Therefore, next to solving the entire instance over one day, we also repeat the experiment assuming optimization starting at noon. In that case, due to the characteristics of an office parking lot, the first interval is expected to have quite a high power. To get meaningful results, we repeat this process 500 times, starting at the instance sampling. The values reported in this work are the median runtimes over those 500 runs. All experiments are run on an Intel Xeon E5-2630 v3 processor.

IV. RESULTS

In this section, we present and analyze the results of the runtime experiments. In particular, we are interested in the efficiency and therefore usability of the various algorithms for MPC in (large) EV parking lots. To this end, we focus on the dependency of runtime on the instance size n . In Section IV-A we first discuss the pre-mature stop feature for FOCS introduced in Section II-C, and how it reflects in the empirical results. Then, in Section IV-B, we focus on the impact that the different maximum flow methods have on the performance of FOCS. Lastly, we compare the impact of various time granularities under invariant maximum flow methods in Section IV-C.

A. FOCS-pm

As discussed in Section II-C, an MPC approach is mostly interested in the next (few) control action(s). Therefore, we propose FOCS-pm as an adapted FOCS that speeds up computation by stopping the optimization pre-maturely when the control action for the first upcoming intervals is known. While by design we expect the feature to speed up computation, this section investigates the extend of the effect, and validates it empirically.

Figure 3 shows the total runtimes of FOCS and FOCS-pm relative to instance size n for the four time granularities, and maximum flow method `shortest_augmenting_path()`. All instances are solved for the full day. First, we note that the runtimes of FOCS and FOCS-pm are very similar for full day instances. Comparing the values over all instance sizes, the improvement of FOCS-pm relative to FOCS is at most 8%, 11%, 12% and 15% for time granularities of 1 minute, 15 minutes, 30 minutes and 1 hour respectively. The average improvements are 0%, 2%, 6% and 9% respectively. We see that especially for finer time granularities, the computational gain of the feature is rather small for this use case.

In the analysis above, we considered instances for a full day. In a next step, we present results of experiments that consider instances starting at noon. Figure 4 shows the total runtimes of FOCS and FOCS-pm relative to instance size n for the four time granularities, and maximum flow method `shortest_augmenting_path()`. However, as opposed to Figure 3, instances are solved starting from noon. As expected, for situations with the same instance size and time granularity, solving the (smaller) problem at noon is faster than solving the entire day. Furthermore, the added value of the pre-mature stop feature is even more pronounced in the noon

TABLE I: Complexity table for four maximum flow algorithms. Here, N , M and U are respectively the number of nodes, edges and the maximum capacity in a given flow network.

Max flow algorithm	Complexity [12]	Networkx complexity	Complexity on FOCS network
<code>shortest_augmenting_path()</code>	$\mathcal{O}(NMU)$	$\mathcal{O}(N^2M)$	$\mathcal{O}(n^4)$
<code>edmonds_karp()</code>	$\mathcal{O}(M^2N)$ or $\mathcal{O}(m^2 \log U)$	$\mathcal{O}(M^2N)$	$\mathcal{O}(n^5)$
<code>preflow_push()</code>	$\mathcal{O}(MN \log(\frac{N^2}{M}))$	$\mathcal{O}(N^2\sqrt{M})$	$\mathcal{O}(n^3)$
<code>dinitz()</code>	$\mathcal{O}(N^2M)$ or $\mathcal{O}(MN \log U)$	$\mathcal{O}(N^2M)$	$\mathcal{O}(n^4)$

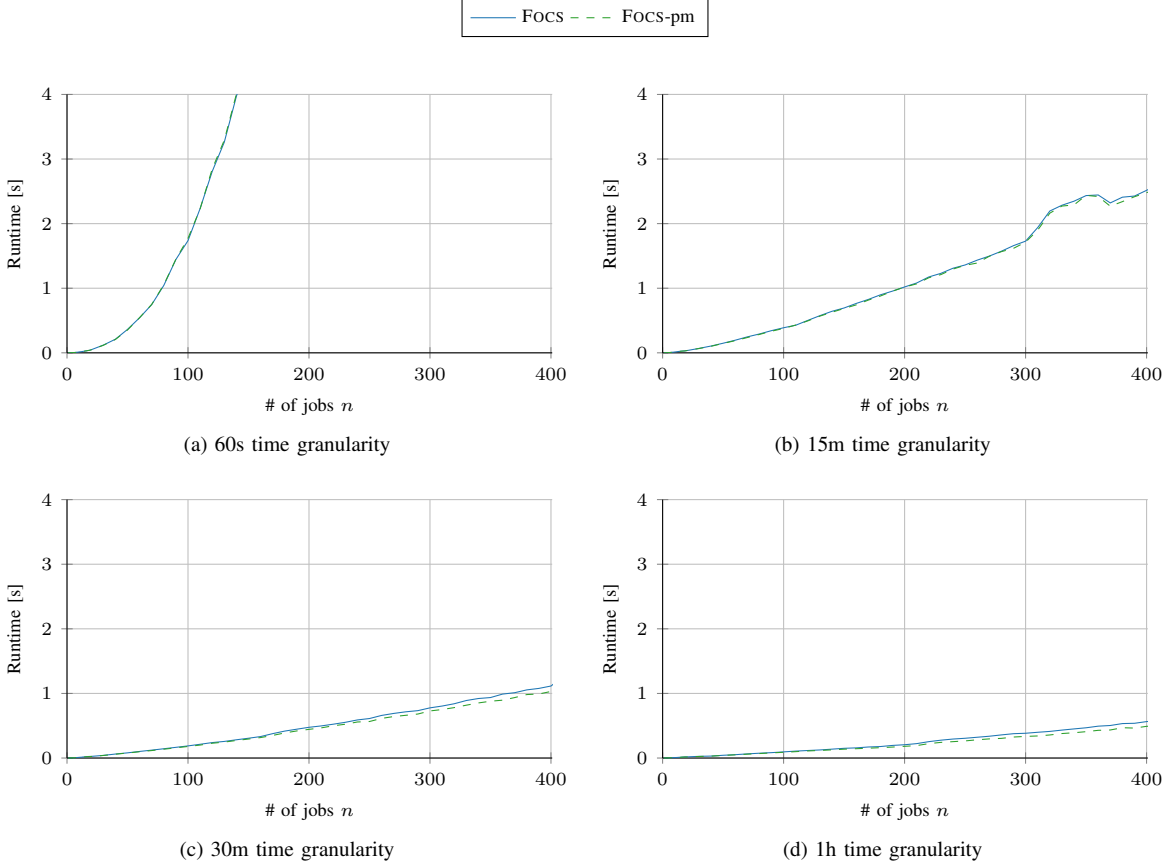


Fig. 3: Runtimes relative to instance size, for various time granularities, solved for the whole day and using `shortest_augmenting_path()`.

experiments. For time granularities of 1 minute, 15 minutes, 30 minutes and 1 hour, the experiments show runtime improvements of at most 44%, 39%, 37% and 38% respectively for up to 400 EVs. That maximum improvement of the median runtime corresponds to instance sizes 400, 390, 290 and 340, all rather large instances. The average improvements amount to 33%, 28%, 26% and 23% respectively.

B. Maximum flow method comparison

The efficiency of an algorithm in the context of an MPC among others depends on the implementation. As mentioned in Section II-B, FOCS relies on solving multiple maximum flow problems. Hereby, in theory it does not matter which concrete algorithm is used. However, since according to Table I the available maximum flow algorithms are non-linear, this section investigates their impact on the scalability of FOCS for large groups of EVs.

As mentioned in Section III-B, `networkx` provides a number of functions that compute maximum flows. In particular, our experiments use the

`shortest_augmenting_path()`, `edmonds_karp()`, `preflow_push()` and `dinitz()` functions from the `networkx` package. Table I provides an overview of those functions, and the following three properties. First, per function, we cite the theoretical complexity of its namesake based on [12]. Note that the implementations of `shortest_augmenting_path()`, `edmonds_karp()`, `preflow_push()` and `dinitz()` likely do not strictly follow the original algorithms ([14], [15], [16]/[17] and [18] respectively). Since their publication, various improvements and variants have been introduced that are usually referred to by the same name. Secondly, Table I presents the theoretical complexity as reported by `networkx`². Finally, the last column takes the complexity reported by `networkx` and bounds the theoretical complexity specifically for flow networks of FOCS instances. We note that the number of nodes (compare flow network in Figure 1) is bound by 2 plus the number of jobs and intervals, being

²last accessed 19 December 2023

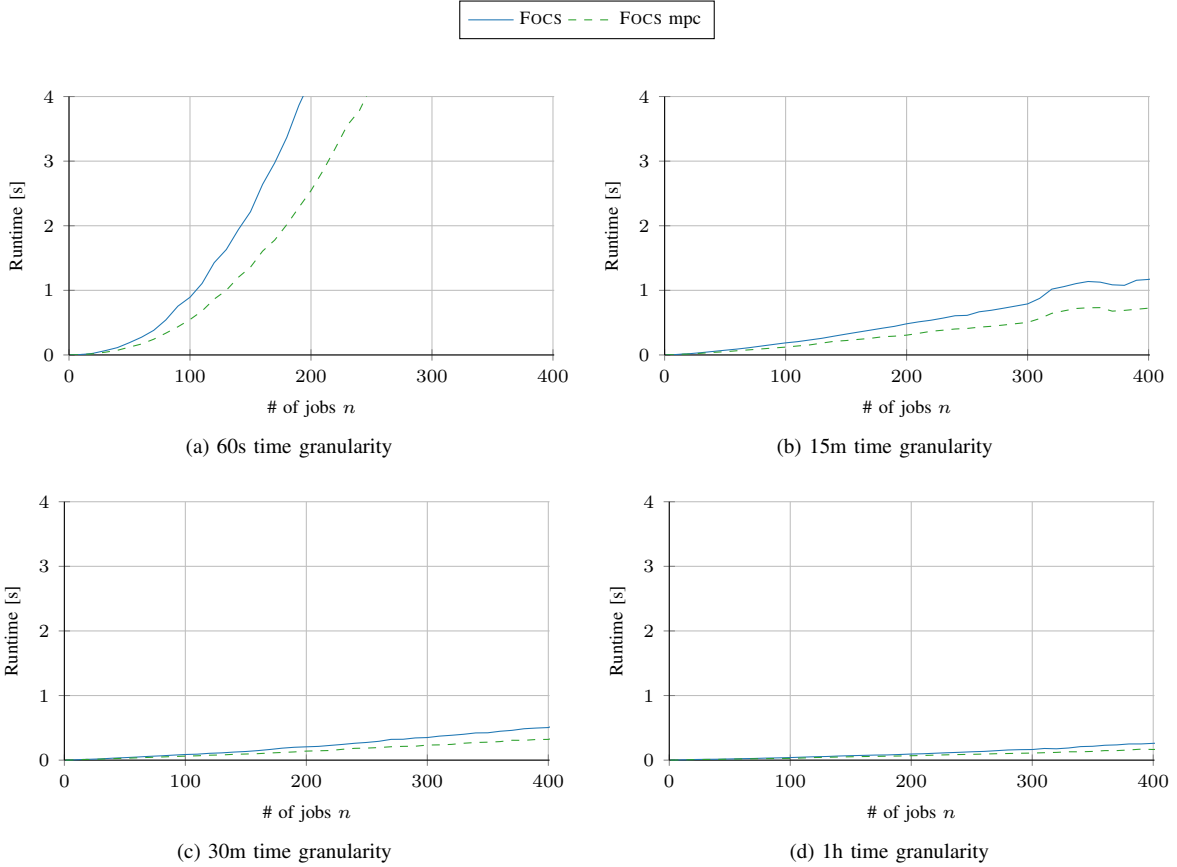


Fig. 4: Runtimes for instances solved at noon using `shortest_augmenting_path()`.

$2 + n + (2n - 1) = 3n + 1$. The number of edges is roughly bound by $n + n(2n - 1) + (2n - 1) = 2n^2 + 2n - 1$. We derive the theoretical complexity in terms of the input size of FOCS instances by substituting this in the bound reported by `networkx`. Since these maximum flow solvers are only used in the solving of FOCS (and therefore do not influence the model building), we restrict the discussion of the results to the solving part of the runtime.

Figure 5 presents the solving time of FOCS for a full day and quarterly time granularity. Among the methods using augmenting paths, we see a clear difference in performance between methods. While the time taken to solve FOCS using `shortest_augmenting_path()` seems to grow almost linearly for instance sizes up to 400, solving times with `edmonds_karp()` and `dinitz()` clearly increase non-linearly. Finally, the preflow-push method behaves similarly to `shortest_augmenting_path()`.

If we compare this to the theoretical complexity of the various maximum flow methods embedded in `networkx` (see Table I), the most striking observation is that while `preflow_push()` has the smallest theoretical runtime on FOCS networks ($\mathcal{O}(n^3)$), the theoretically quartic runtime of `shortest_augmenting_path()` ($\mathcal{O}(n^4)$) outperforms the other tested methods. `edmonds_karp()` shows the highest theoretical complexity, and empirically is the second slowest method, overtaken only by `dinitz()`.

Overall, `shortest_augmenting_path()` appears to

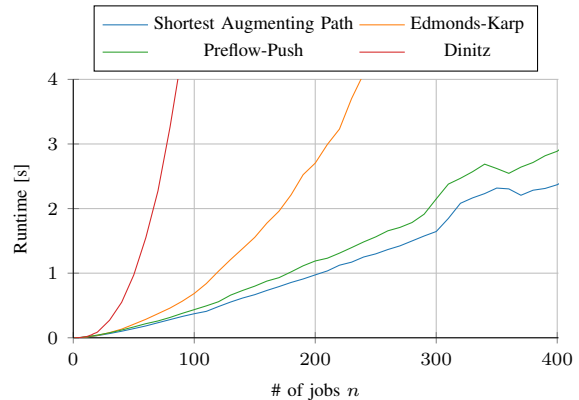


Fig. 5: Runtimes for solving only for quarterly granularity, comparing maximum flow algorithms applied in FOCS.

be the dominating method in our experiments with respect to runtime. The experiments further indicate that the proper choice of maximum flow method greatly increases the usability of FOCS for EV scheduling.

C. Runtime comparison

In the following, we compare FOCS to the QP implementation for comparison with (commercial) optimization tools already available for MPCs. The total runtime (i.e., the sum of the model building and solving), relative to instance size n for the four time granularities is depicted

in Figure 6. Note that for FOCS, we assume that the flow method `shortest_augmenting_path()` is applied. All instances are solved for the full day.

First, we note that across subfigures, the quadratic program solved in Gurobi initially either outperforms FOCS or runs at similar speed. However, we observe a gradually steeper increase in runtime relative to instance size for QP compared to the FOCS model. As a result, the median runtime of FOCS outperforms QP starting from instance sizes of approximately 90, 120, 90 and 80 for time granularities of 1 minute, 15 minutes, 30 minutes and 1 hour respectively. Furthermore, we observe a sudden steep increase in the runtime of the QP implementation, at instance sizes from around 90, 130, 170 and 230 in Figures 6a, 6b, 6c and 6d respectively. This behaviour possibly reflects that a certain memory threshold is reached after which solving becomes more costly for Gurobi due to swapping memory.

Overall, as expected, a finer time granularity for a given instance size results in a larger runtime for both the QP and FOCS. Notably, the slope of the runtime for FOCS seems to be almost linear for the instance sizes considered, except in Figure 6a. There, a non-linear increase can be observed. To give an indication of this growth, the median runtimes for FOCS and QP with instance size 400 under time granularity 1 minute are respectively 36 and 62 seconds.

In a next step, we disaggregate the runtimes into model building and solving components (see Figure 7). Figure 7a presents the results for FOCS, whereas Figure 7b presents the results for QP. Both figures are stackplots, depicting the total runtime as sum of the model building (bottom part, orange shading) and solving (top part, blue shading). The results correspond to experiments with time granularity of 15 minutes and `shortest_augmenting_path()` applied within FOCS.

Notably, FOCS spends most of its computation time on solving. The almost negligible contribution of model building can be attributed to the mostly dictionary-based implementation of flows in the `networkx` package. For QP, on the other hand, model building makes up a considerable portion of the runtime. Moreover, Figure 7b clearly attributes the earlier observed sudden steep increase in the runtime of the QP (discussed above) to its solving. This can be seen in the continued smooth increase in the bottom (orange) area, whereas approximately at instance size 130 the contribution of the solving time to the total runtime increases drastically.

V. DISCUSSION

In the previous section, we have presented the empirical results of our experiments. Next, we want to shortly discuss some of the limitations of this research, and put the results in perspective to physical parking lots for EV charging with MPC.

First, we consider the implementation of FOCS as used for the experiments. While the overall FOCS method builds on augmentation of flows, within the used implementation the number of considered intervals is reduced between consecutive iterations and a new maximum flow with the empty flow

as initialization is executed. A more efficient implementation would use the flow from the previous iteration, and augment on top of that. Especially if FOCS uses a maximum flow method that relies on augmenting paths, this would speed up the optimization.

Moreover, in MPC settings, both efficiency and quality of the optimization are crucial. Therefore, it is important to note that for the conclusions of Section IV-B we have not considered the exact form of the found schedule but focused solely on the runtime of the various flow methods. However, in the setting of EV scheduling, the fairness, robustness and user acceptability of the found schedule itself are of great importance. To illustrate, consider the example in Figure 8. Here, jobs 1 and 4 share the same properties. The example provides two schedules. In the schedule on the left those two jobs are scheduled consecutively in disjoint intervals, whereas on the right they are scheduled in parallel. Both schedules are optimal solutions to the problem with constraints (1) minimizing objective function (3). One may argue, however, that the schedule on the right is fairer to the users and more robust to early departure of either job. Such considerations were left out of scope of this work, but should be considered in future work considering optimization in MPC settings.

Finally, in the comparison of FOCS to QP, we do not take the adaptivity of the models in an MPC setting into account. While the model building for QP takes considerable time, Gurobi is known for efficiently altering and re-solving its models, using previous results in its initialization. This was left out of scope for the comparison. While we do recommend future work in that direction, we first suggest to conduct theoretical research on the efficient initialization of FOCS and to implement the subsequent results to allow for a reasonable comparison.

VI. CONCLUSION

This work investigated potential algorithms for an optimization subroutine in model predictive controllers for EV scheduling. To this end, we introduced a pre-mature stop feature for FOCS, an algorithm suitable for this subroutine. The proposed feature can decrease runtime in MPC settings. It is particularly effective close to peak hours, for example around noon at an office parking lot when parking lot occupancy is expected to be at its peak. Using a 15 minute granularity, the proposed variant of FOCS shows a median runtime improvement of on average 28% compared to the original unadapted version to calculate a charging schedule at noon. Moreover, this work has illustrated the considerable impact of the chosen maximum flow method for FOCS. Finally, for large EV parking lots, the presented runtimes show the competitiveness of FOCS with commercial state-of-the-art optimization software.

Future work may dive deeper into the integration of FOCS into an MPC, and into the properties of the found schedules depending on the used maximum flow method.

ACKNOWLEDGEMENTS

This research is conducted within the SmoothEMS met GridShield project subsidized by the Dutch ministries of EZK and BZK (MOOI32005).

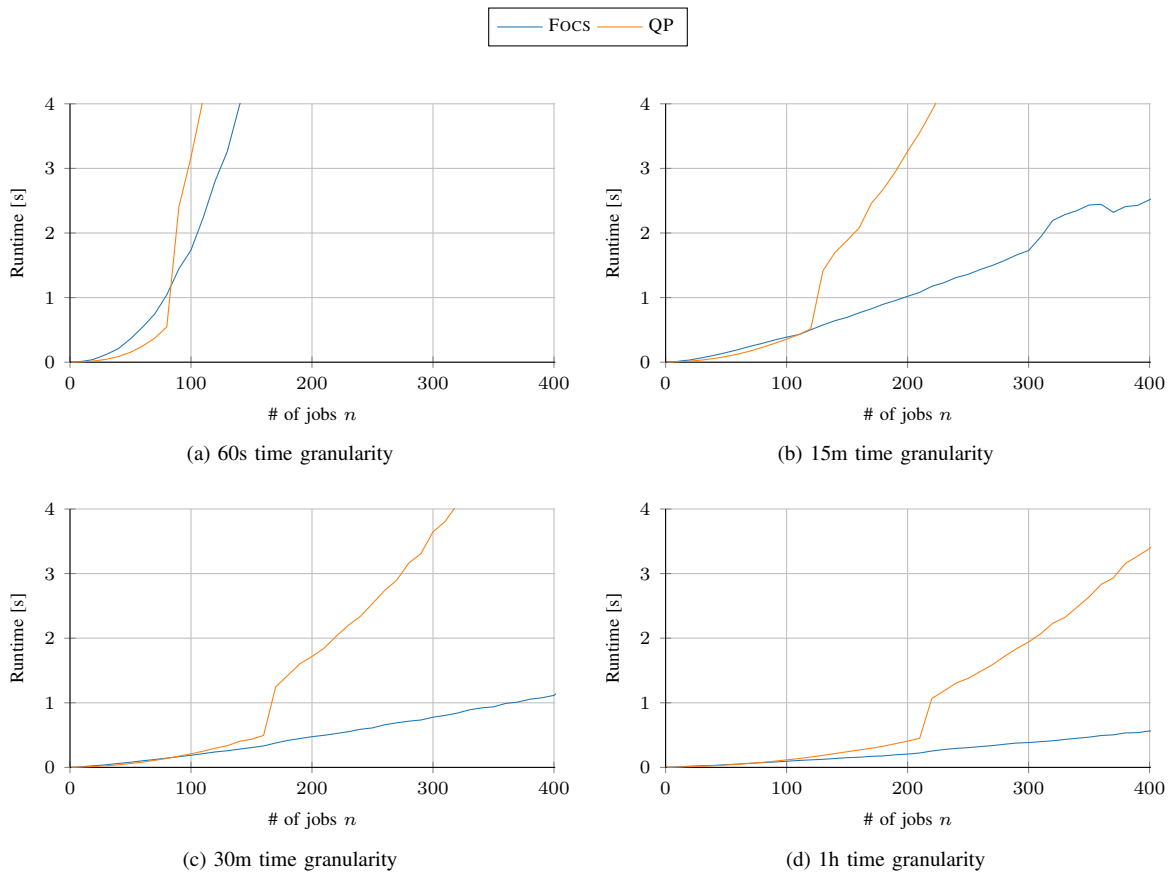


Fig. 6: Runtimes relative to instance size, for various time granularities, solved for the whole day and using `shortest_augmenting_path()`.

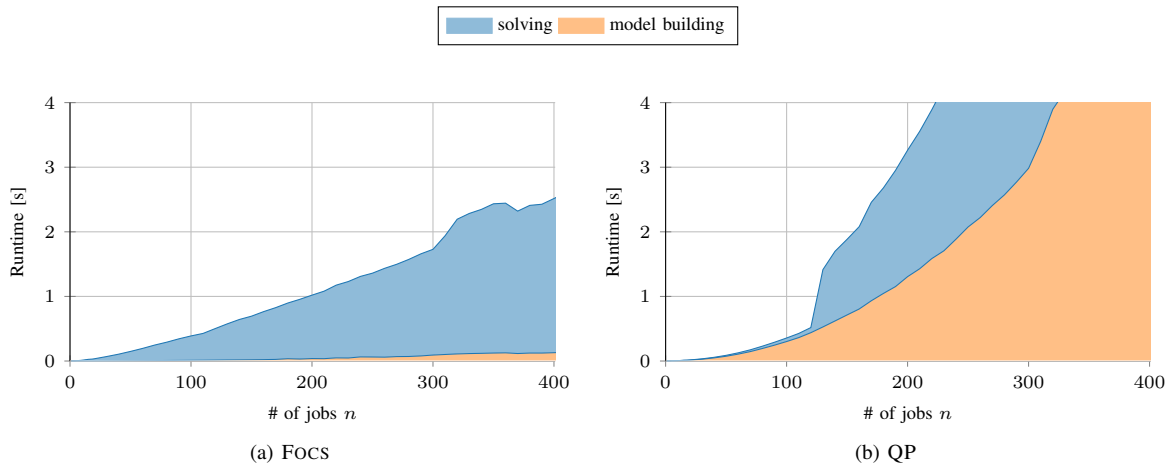


Fig. 7: Runtimes disaggregated into model building and solving for 15 m granularity.

REFERENCES

- [1] R. Wolbertus and R. van den Hoed, *Plug-in (Hybrid) Electric Vehicle Adoption in the Netherlands: Lessons Learned*. Cham: Springer International Publishing, 2020, pp. 121–143.
- [2] N. Sadeghianpourhamami, N. Refa, M. Strobbe, and C. Develder, “Quantitative analysis of electric vehicle flexibility: A data-driven approach,” *International Journal of Electrical Power & Energy Systems*, vol. 95, pp. 451–462, 2018.
- [3] G. J. Osório, M. Gough, M. Lotfi, S. F. Santos, H. M. Espassandim, M. Shafie-khah, and J. P. Catalão, “Rooftop photovoltaic parking lots to support electric vehicles charging: A comprehensive survey,” *International Journal of Electrical Power & Energy Systems*, vol. 133, p. 107274, 2021.
- [4] J. W. Eising, T. van Onna, and F. Alkemade, “Towards smart grids: Identifying the risks that arise from the integration of energy and transport supply chains,” *Applied Energy*, vol. 123, pp. 448–455, 2014.
- [5] K. Turitsyn, N. Sinityn, S. Backhaus, and M. Chertkov, “Robust broadcast-communication control of electric vehicle charging,” in *2010 First IEEE International Conference on Smart Grid Communications*, 2010, pp. 203–207.
- [6] M. Cañigüeral and J. Meléndez, “Flexibility management of electric vehicles based on user profiles: The arnhem case study,” *International Journal of Electrical Power & Energy Systems*, vol. 133, p. 107195, 2021.
- [7] G. Vecchio and L. Tricarico, ““may the force move you”: Roles and actors of information sharing devices in urban mobility,” *Cities*, vol. 88, pp. 261–268, 2019.

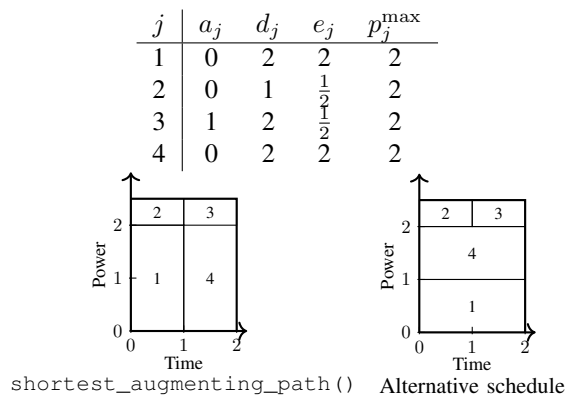


Fig. 8: Example of an instance where `shortest_augmenting_path()` results in a schedule with bias for one job over the other.

- [8] Q. Zhou and C. Du, "A quantitative analysis of model predictive control as energy management strategy for hybrid electric vehicles: A review," *Energy Reports*, vol. 7, pp. 6733–6755, 2021.
- [9] L. Winschermann, M. E. T. Gerards, A. Antoniadis, G. Hoogsteen, and J. Hurink, "Relating electric vehicle charging to speed scaling with job-specific speed limits," 2023.
- [10] Gurobi Optimization, LLC, "Gurobi Optimizer Reference Manual," 2023. [Online]. Available: <https://www.gurobi.com>
- [11] A. A. Hagberg, D. A. Schult, and P. J. Swart, "Exploring network structure, dynamics, and function using networkx," in *Proceedings of the 7th Python in Science Conference*, G. Varoquaux, T. Vaught, and J. Millman, Eds., Pasadena, CA USA, 2008, pp. 11 – 15.
- [12] O. Cruz-Mejía and A. N. Letchford, "A survey on exact algorithms for the maximum flow and minimum-cost flow problems," *Networks*, vol. 82, no. 2, pp. 167–176, 2023.
- [13] B. Nijenhuis, L. Winschermann, N. Bañol Arias, G. Hoogsteen, and J. L. Hurink, "Protecting the distribution grid while maximizing ev energy flexibility with transparency and active user engagement," in *CIREP Porto Workshop 2022: E-mobility and power distribution systems*, vol. 2022, 2022, pp. 209–213.
- [14] L. R. Ford and D. R. Fulkerson, "Maximal flow through a network," *Canadian Journal of Mathematics*, vol. 8, p. 399–404, 1956.
- [15] J. Edmonds and R. M. Karp, "Theoretical improvements in algorithmic efficiency for network flow problems," *J. ACM*, vol. 19, no. 2, p. 248–264, apr 1972.
- [16] A. Karzanov, "Determining the maximal flow in a network by the method of preflows," *Doklady Mathematics*, vol. 15, p. 434–437, 02 1974.
- [17] A. V. Goldberg and R. E. Tarjan, "A new approach to the maximum-flow problem," *J. ACM*, vol. 35, no. 4, p. 921–940, oct 1988.
- [18] Y. Dinitz, "Algorithm for solution of a problem of maximum flow in networks with power estimation," *Soviet Math. Dokl.*, vol. 11, pp. 1277–1280, 01 1970.