MIE1666: A Machine-Learning Accelerated Mixed-Integer Program for Optimization of Wind Farm Layouts

Khodr Jaber[a], Lana Hassoune[a], Nancy Awad[a]

[a]*Mechanical and Industrial Engineering Department, University of Toronto, 5 King's College Rd, Toronto, M5S 3G8, Ontario, Canada*

Abstract

A novel machine learning approach for accelerating the solution of mixed-integer programming (MIP) algorithms for Wind Farm Layout Optimization (WFLO) problems is proposed. In particular, the parameters of the MIP solver implemented in the software `Gurobi` are tuned using Sequential Model-Based Algorithm Configuration - a machine learning tool based on random forests - with the objective of reducing total run-time of the WFLO program. For training and testing, flow fields are generated over randomized terrains in a highly simplified way in order to bypass the need for standard computational fluids dynamics (CFD) tools, which are computationally expensive. The results of applying Sequential Model-Based Algorithm Configuration (SMAC) to the WFLO program are evaluated by comparison with the run-time values of the program with default parameters.

The main code used in this paper can be found using the following link: Main Code. The repository containing the C++ code that generates the inputs to the main code can be found using the following link: C++ Code.

Keywords: Wind Farm Layout Optimization, Sequential Model-Based Algorithm Configuration, Mixed-Integer Programming

1. Introduction

The optimization of wind farm layouts is an active topic of research whose results have led to more efficient renewable energy systems in an effort to curtail climate change. It is a complex problem involving nonlinear dynamics, difficult geometries and a variety of physical parameters that affect the energy output of turbines including their quantity, positions, angles and pairwise distances. Detailed simulations of the flow physics involved in the transformation of kinetic energy to electrical energy are further complicated by the presence of turbulence, which imposes constraints on the choice of resolution and computational model. This has motivated the development of optimization algorithms capable of identifying ideal turbine locations in a way that involves computational fluid dynamics (CFD) so that results are precise (or at least physically meaningful). These Wind Farm Layout Optimization (WFLO) algorithms aim to maximize energy production, while satisfying numerical and geometric constraints amongst the turbines. A common way of formulating the optimization problem for wind farm layouts is the mixed integer program (MIP) in which the potential locations for turbines is on a discrete grid over the terrain. MIPs and their formulations for the present work will be elaborated on in the following section.

WFLO algorithms will typically compromise between the accuracy that would be obtained from highly detailed problem descriptions and the computational time needed for solving them. One way of accelerating the solution procedure is to use machine learning (ML), which will (in a rough sense) identify the best sub-step directions to take as the algorithm runs. Applying ML to an MIP solver would help reduce the run time to find an optimal solution by tuning the amount of time spent on heuristics, the choice of which child node to explore first, and the branch variable strategy. High level research on this topic suggests that minimal work has been done to implement ML to WFLO problems, so exploring this potential will be a step forward in wind farm layout planning.

Email addresses: `khodr.jaber@mail.utoronto.ca` (Khodr Jaber), `lana.hassoune@mail.utoronto.ca` (Lana Hassoune), `nancy.awad@mail.utoronto.ca` (Nancy Awad)

2. Related Work

2.1. Algorithms for WFLO

There are a variety of ways to model the optimization of wind flow layouts, including evolutionary algorithms [3, 4, 5], particle swarm [6] and mixed integer programming [7, 8]. A detailed discussion of the differences in performance of these algorithms can be found in Zhang's thesis on the wake modeling, noise propagation and energy production subtopics in WFLO [2]. These methods can be classified as finding optimal turbine locations over either continuous or discrete regions. While evolutionary and swarm algorithms were said to require a similar number of function evaluations before convergence in the studies in which they were reported, the MIP framework was explicitly stated as having the advantage of well-defined upper bounds on optimality (an advantage which is reported to have not been exploited in the literature at the time of publication of the thesis). Another advantage is embedded in the branch-and-bound scheme most commonly used for solving MIPs (see next subsection), which allows for a degree of flexibility in tailoring solutions for specific problems.

More recent literature has focused on identifying more precise problem statements that account for as much detail as feasibly possible (such as accurate terrain shape [11] and turbine wake effects [10]). This, of course, comes with the price of additional computational expense, whose primary bottleneck is in the calculation of the atmospheric flow field over the terrain. Wakes generated by turbines affect the flow around other turbines, so this interaction must be modeled in a way that minimizes computational cost while not spoiling physical behaviour.

An MIP is an optimization problem involving both integer and real-valued decision variables [13]. Such problems are representative of situations where a set of objects of interest must be allocated over a number of potential locations. The integer variables are thus binary (either the object is located at a point or not), while the real-valued variables define the geometry or physics of the problem. The most common method of solving MIPs is the branch-and-bound method of Land and Doig [14, 12]; this method has been implemented in commercial / academic software such as `CPLEX` [18] and `Gurobi` [19]. A number of MIP formulations have been considered in the literature for WFLO, the differences between them being their structure (in terms of objective function and constraints) and included parameters. Zhang [2] presents a set of quadratic sum-of-squares formulations (one constraint program and one MIP) and a set of linear superposition formulations (both MIP). The linear superposition formulation was first investigated by Donovan [9] and later extended by Zhang. The quadratic formulation is retained by Kuo et al. [1] in an investigation of WFLO over complex terrains. Kuo et al. also discuss the impact of initial wake approximations on the final layout, and propose a relaxed quadratic formulation for when wake effects are being underestimated. For simplicity, the un-modified quadratic MIP will be used under the same framework described by Kuo et al. in the present work.

2.2. Algorithm Configuration

Algorithms used to solve hard computational problems that are based on local search or tree search are highly parametrized. For example, the `CPLEX` solver has 76 parameters used in its search strategy [15]. It is known that optimizing parameter settings can improve the performance of the algorithm however doing so manually has been shown to be an extremely tedious and time-consuming task. Due to this, automated approaches to algorithm configuration have been developed and Sequential Model-Based Algorithm Configuration (SMAC) is one such tool. SMAC has significantly helped improve the speed of local search and tree search algorithms, optimize hyper-parameters of machine learning algorithms, and provide important information about models (such as the most important input variables [16]). Since SMAC has been developed for general algorithm configuration, it can be applied to algorithms for many applications, including the WFLO problem.

SMAC is a sequential model-based search paradigm, which means ”it uses the information gained from the configurations evaluated so far to build (and maintain) a model of the configuration space, based on which configurations are chosen to be evaluated in the future” [17]. The way it works is considered to be a ”black-box optimization problem” based on random forest models. These methods are usually restricted to dealing with real- and integer-valued target algorithm parameters, although techniques have been introduced recently that can handle categorical parameters [16, 17].

3. Methodology

This section describes the methodology that will be employed to set up and run the MIP solver needed to solve the WFLO problem, along with the procedure for implementing SMAC so that the solver's parameters can be tuned. Parameter tuning will ultimately reduce the solver's run-time on given instances and potentially generalize this reduction in run-time to new instances in the future.

3.1. MIP Optimization Model

The MIP formulation for the WFLO problem can be written as [1]:

$$\max \quad \sum_{i=1}^{N} \sum_{s \in S} p_s x_i \Big[U_{0,s,i}^2 - \sum_{j=1}^{N} (U_{0,s,j}^2 - u_{s,ij}^2) x_j \Big] \tag{1a}$$

$$\text{s. t} \quad \sum_{i=1}^{N} x_i = K \tag{1b}$$

$$d_{ij} x_i + d_{ji} x_j \leq 1, \qquad i, j \in \{1, ..., N\} \tag{1c}$$

$$x_i \in \{0, 1\}, \qquad i \in \{1, ..., N\} \tag{1d}$$

where p_s is the probability of wind state s (i.e. the probability of wind being in a direction of index $s \in S$), x_i is the binary variable denoting the existence of a turbine at location i, $d_{ij} \in \{0, 1\}$ is a binary parameter denoting a distance criterion between turbines at locations i and j and $U_{0,s,i}, u_{s,ij}$ are the wind velocities at turbine i without wake effects[1] and due to a wake from upstream turbine j, respectively [2]. The objective function of this MIP maximizes the kinetic energy of the wind that is absorbed by each turbine, while satisfying capacity and distance constraints:

- The capacity constraint states that there can only be K turbines in the solution. $K = 4$ was fixed arbitrarily for the present work.
- The second constraint relates to not having two or more turbines placed in the same exact location. d_{ij} and d_{ji} denote the distance between two different turbines. If d_{ij}=d_{ji}=1, the distance constraint is violated. Note that $d_{ii} = 0$ for all $i \in \{1, ..., N\}$.

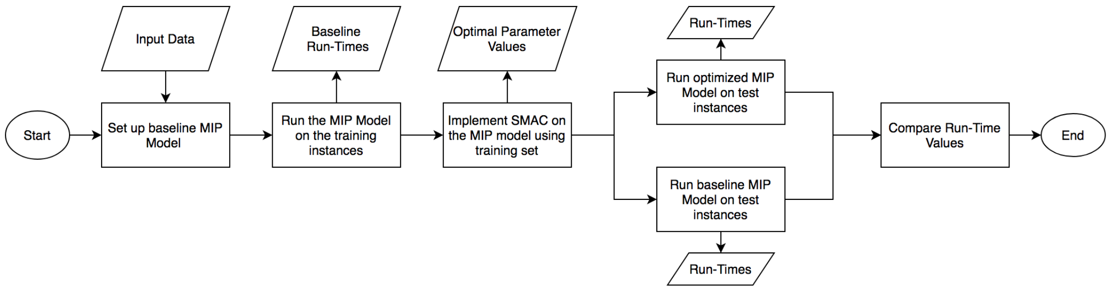
3.2. Overall Algorithm

The overall algorithm to optimize the MIP solver for the WFLO problem was divided into two phases. The first phase involved the complete set up of the baseline MIP model and first round of SMAC implementation focused on optimizing three selected parameters. Once those optimal parameters were found and the results were shown to be positive for the auto-tuned MIP model, it was decided to tune an additional parameter. The second phase involved updating the baseline MIP model based on the results in phase 1 and re-implementing SMAC on the updated baseline MIP model to return optimal values this time for 4 parameters.

Figure 1 illustrates the steps taken in phase 1. First, a specific terrain surface was selected with $18 \times 18 = 324$ possible locations for turbines. The number of locations was chosen to be 324 since the MIP was able to converge within only a few seconds for all smaller sets while also taking an exponentially longer time to solve for all larger sets (see Figure 2). Since the main objective of the present work is to implement SMAC and optimize the MIP solver and considering time and computational power limitations, a decision was made to limit the number of locations and, therefore, the run-time of the baseline MIP model. However, we expect that following the same steps for a larger number of locations will lead to similar results.

Once the terrain and number of locations were selected, 100 random instances were generated by randomizing the placement of hills and therefore the potential wind velocities on the terrain. Within each instance, the wind velocities $U_{0,s,j}, u_{s,ij}$ are generated for all possible turbine locations. A set of $S = 8$ randomly generated wind state probabilities were generated such that $\sum_s p_s = 1$. These values are fixed for all generated instances. Details of how the velocities

[1]A wake is defined as the region of reduced velocity due to an obstruction of wind flow.



Start
Input Data
Set up baseline MIP Model
Baseline Run-Times
Run the MIP Model on the training instances
Optimal Parameter Values
Implement SMAC on the MIP model using training set
Run-Times
Run optimized MIP Model on test instances
Run baseline MIP Model on test instances
Run-Times
Compare Run-Time Values
End

Figure 1: Flowchart of Phase 1 methodology

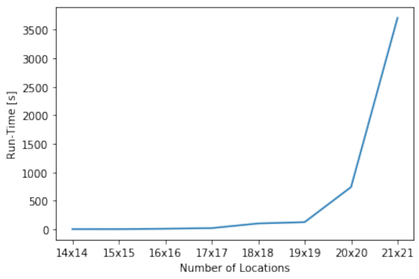


Figure 2: Plot of MIP run-time against Number of Locations

were simulated and generated are provided in section §4.1. The data set of instances was then split into training and testing sets; 80 instances were used to train the SMAC algorithm to optimize the selected MIP parameters while the remaining 20 instances were used to test if the results generalized to unseen instances.

The training set was then imported as the inputs to the MIP model created using `Gurobi`. The `Gurobi` MIP solver was run with all its default parameters (i.e. the baseline model) on the 80 training instances and returned a list containing the run-time of each instance, as well as the average run-time. These two outputs formed the baseline results of the `Gurobi` MIP solver before optimization. Once the baseline results were captured, SMAC was set up using a python module called pySMAC. Through pySMAC, SMAC was implemented on the `Gurobi` MIP model, setting the objective to minimize the run-time by changing the following three `Gurobi` MIP parameters:

- BranchDir (Branch direction preference)
  - Parameter range: Integer [-1,1]; default = 0
- Heuristics (Turn MIP Heuristics up or down)
  - Parameter range: Real [0,1]; default = 0.05
- VarBranch (Branch variable selection strategy)
  - Integer [-1,3]; default = -1

These parameters were selected (under the guidance of the instructor) because of their stronger impact on run time (as opposed to other parameters which would compromise the quality of the objective function). Due to computational and time limitations, the maximum evaluations parameter for pySMAC was set to 250. SMAC was implemented on the 80 training instances to return the optimal values of the selected parameters. Once the optimal values were found, both the auto-tuned `Gurobi` MIP model (with the optimal parameter values) and baseline `Gurobi` MIP model (with the default parameter values) were run on the 20 test instances, after which the run-time values were compared.

Figure 3 illustrates the steps taken in phase 2. For this phase, the baseline `Gurobi` MIP Model was set to be

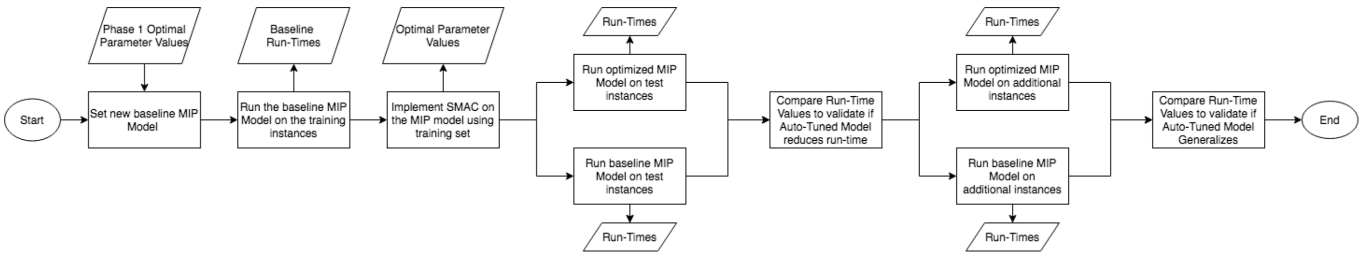

Phase 1 Optimal Parameter Values
Baseline Run-Times
Optimal Parameter Values
Run-Times
Run-Times
Start
Set new baseline MIP Model
Run the baseline MIP Model on the training instances
Implement SMAC on the MIP model using training set
Run optimized MIP Model on test instances
Run baseline MIP Model on test instances
Compare Run-Time Values to validate if Auto-Tuned Model reduces run-time
Run optimized MIP Model on additional instances
Run baseline MIP Model on additional instances
Compare Run-Time Values to validate if Auto-Tuned Model Generalizes
End
Run-Times
Run-Times

Figure 3: Flowchart of Phase 2 methodology

the auto-tuned `Gurobi` MIP Model from phase 1. Therefore, the optimal values for the BranchDir, Heuristics, and VarBranch parameters were now set as the default values. The updated baseline `Gurobi` MIP solver was run again on the 80 training instances and returned a list containing the run-time of each instance, as well as the average run-time. These two outputs formed the new baseline results of the `Gurobi` MIP solver before the optimization. Once the baseline results were captured, SMAC was re-implemented on the `Gurobi` MIP model, setting the objective to minimize the run-time by changing the following four `Gurobi` MIP parameters:

- BranchDir (Branch direction preference)
  - Parameter range: Integer [-1,1]; default = -1
- Heuristics (Turn MIP Heuristics up or down)
  - Parameter range: Real [0,1]; default = 0.0817
- VarBranch (Branch variable selection strategy)
  - Integer [-1,3]; default = 1
- SubMIPNodes (Nodes explored by sub-MIP heuristics)
  - Integer [0,2000000000]; default = 500

The maximum evaluations parameter for pySMAC remains fixed at 250 due to the same limitations. SMAC was implemented on the 80 training instances to return the optimal values of the selected parameters. Once the optimal values were found, both the auto-tuned `Gurobi` MIP model (with the optimal parameter values) and baseline `Gurobi` MIP model (with the default parameter values) were run on the 20 test instances, after which the run-time values were compared.

In order to test if the auto-tuned `Gurobi` MIP model generalizes, 20 of each of two kinds of additional testing instances were generated. The first set of additional instances were generated in the same way as before, however, a different terrain form was used so that generalization of the auto-tuned model to different terrains could be tested. The second set of additional instances were generated using the same terrain form as before, however, this time with $19 \times 19 = 361$ possible locations so that generalization of the model towards terrains with a higher number of locations could be tested.

4. Experimental Setup

4.1. Datasets: Generating the Flow Field

The flow field in an atmospheric boundary layer is governed by a set of partial differential equations known as the Navier-Stokes (NS) equations. These equations are notoriously difficult to solve for a variety of reasons:

- The NS equations are nonlinear due to the so-called convective term, which causes standard methods of discretization to fail. Discretization of the convective term is an active research topic on its own, and the schemes most commonly used in commercial software strike a balance between accuracy and computational expense.

- In an atmospheric boundary layer the flow is generally turbulent, meaning that a large range of physical scales must be resolved. A direct resolution of these scales (i.e. a Direct Numerical Simulation) can quickly become infeasibly expensive as the smallest distance between grid elements in the solver must be smaller than the smallest-scale vortex in the flow. Most engineering solvers employ turbulence models, which simplify the NS equations in some way so that the grid resolution constraint can be relaxed.

- The initial and boundary conditions required to simulate accurate flows over complex terrains can be difficult to implement depending on the choice of grid structure. Specifying these conditions for approximate or placeholder quantities such as turbulent kinetic energy is even more challenging as equivalent mathematical formulations consistent with the governing equations must be identified before being sent to the solver as input.

In the present algorithm, a placeholder solver is developed that generates flow fields nearly instantaneously to hasten the production of training sets for SMAC. The terrains for this solver are a set of 'hills' defined as circular disks with radii fixed as fractions of the terrain length. The flow field is composed of two components - a passive field (which is a function of the terrain) and a disturbed field (which accounts for wake effects of turbines). This is written as:

$$\mathbf{u}(\mathbf{x}) = \prod_{i=1}^{K} \mathcal{D}_i(\mathbf{x})\Big[\sum_{j=1}^{N_h} \mathbf{u}_{j,h}(\mathbf{x}) + \mathbf{u}_s(\mathbf{x})\Big], \qquad 0 \le s \le 7 \tag{2}$$

$$\mathbf{u}_{j,h}(\mathbf{x}) = \left(\frac{-(y-y_j)}{(x-x_j)^2+(y-y_j)^2}\right)\hat{x} + \left(\frac{(x-x_j)}{(x-x_j)^2+(y-y_j)^2}\right)\hat{y} \tag{3}$$

$$\mathbf{u}_s(\mathbf{x}) = \sin\left(\frac{s\pi}{4}\right)\hat{x} + \cos\left(\frac{s\pi}{4}\right)\hat{y} \tag{4}$$

$$\tag{5}$$

where (x_j, y_j) is the location of the j$^{\text{th}}$ hill, $\mathbf{u}_s, \mathbf{u}_{j,h}$ are the passive velocity field components due to wind state s and hill j, respectively, and $\mathcal{D}_i$ is the damping factor due to turbine i (which is defined over a conic section in the direction of wind at state s). the damping factor of i is equal to 0.7 over the region defined by the wake caused by turbine i and 1.0 otherwise. A sample flow field generated with this definition without the damping factors is displayed in **Figure 4**. The code takes the input in the form a text file, applies the wake effects to locations designated with turbines, and outputs a .csv file containing $U_{s,0,i}$ and $u_{s,ij}$ - the velocities at locations i without wake effects and with wake effects from turbines j, respectively.

The primary advantage in using this placeholder solver is in the construction of the MIP's objective function, which requires knowledge of the effects of all turbines on all other turbines. This latter condition normally necessitates a CFD simulation involving only two turbines at a time, which can take several days to complete depending on the choice of solver. MIP-CFD algorithms in the literature deal with this by iterating the MIP and involving only a few select turbines at a time chosen based on the initial flow field. At the end of each iteration, the choice of turbines is updated based on an educated guess of where the actual optimal set of turbines might be located. With the placeholder, all turbine interactions are generated instantaneously and only one MIP run is needed.

Since multiple instances of terrain were needed for training, the terrain generator was randomized in two variables - hill location and hill size. The size of hill h was defined as a random number r_h selected in the range $[0.05, 0.25]$ such that the radius of the hill was $r_h L$, where L is the terrain length (for the present work, terrain lengh and width were fixed equal). A random location was chosen in the range $[r_h, L - r_h]^2$, and three hills were added to each terrain instance.

4.2. Baselines

The baseline model in phase 1 was the `Gurobi` MIP solver with all its default parameters which was implemented using a python module Gurobipy. An educational license was required to be installed in order to run the MIP solver with the desired number of locations (the free version could only run on a limited number of turbine locations). The baseline model in phase 2 was the `Gurobi` MIP solver using the optimal values for the tuned parameters obtained in phase 1.

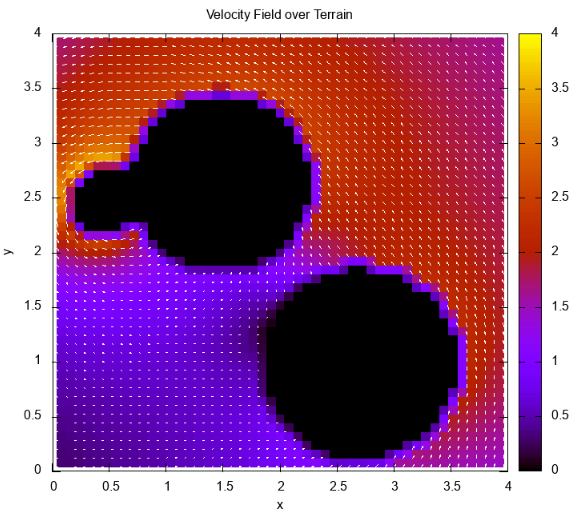


Figure 4: A sample flow field generated over three randomly-positioned hills (with flow assumed to be zero within the disc-shaped regions defining them). Colors represents the magnitude of the velocity field $||\mathbf{u}|| = \sqrt{u^2 + v^2}$ at each point.

4.3. Evaluation Metrics

Since SMAC was set up to minimize the run-time value, the evaluation metric used in both phases was a comparison of run-time values between the baseline and the auto-tuned model. Therefore, once SMAC returned the optimal parameter values, its performance was evaluated on the test instances by comparing the run-time of each instance in the auto-tuned `Gurobi` MIP model to the run-time in the baseline `Gurobi` MIP model.

For phase 2 specifically, in addition to using the test instances, generalization of the auto-tuned model to instances using a different terrain or a higher number of locations was evaluated. Therefore, additional testing instances were generated - ones using the same terrain with a higher number of potential locations, and ones using another terrain with the same number of locations. Next, the run-times of the auto-tuned `Gurobi` MIP model were compared to the run-time of the baseline `Gurobi` MIP model of phase 2 using these additional instances.

4.4. Computing Setup

All `Gurobi` MIP models and SMAC were run on a personal computer with the educational license installed for `Gurobi` (as previously mentioned). Since `Gurobi` does not yet support GPUs, the models where run using a CPU. On average, in both phases, the time taken to run the training and testing models was approximately 22 hours (4 hours to run the baseline `Gurobi` MIP model over the training set, 15 hours for training SMAC with 250 iterations, 1.3 hours to run the baseline `Gurobi` MIP model over the testing set, 1.15 hours to run the auto-tuned `Gurobi` MIP model over the testing set.
Additionally, for phase 2, the time taken to run the baseline `Gurobi` MIP model over the additional testing set was 1.5 hours, and the time taken to run the auto-tuned `Gurobi` MIP model over the additional testing set was 1.5 hours.

As previously stated, SMAC iterations were restricted by selecting the maximum number of evaluations as 250 due to time and computational power limitations. Otherwise, a larger number of maximum evaluations (e.g. 1000) would have been selected, which would allow for an investigation of the effect of larger combinations of values of the parameters.

5. Experimental Results and Discussion

Phase 1 Results: The baseline `Gurobi` MIP model was run on each of the training instances, with run-times being computed and stored in a list. A histogram of the individual run-times was plotted (displayed in **Figure 5**) in order to better visualize their distribution. Inspection of the histogram reveals that the run-times of the instances appear to have an exponential distribution. This means that just by randomizing the location of the hills in the simulation, different run-times were obtained when solving the MIP at each instance. However, since the training and testing instances were generated from the same random CFD simulation model, it is hypothesized that the results that will be obtained from implementing SMAC to reduce the run-time over the training instances will also reduce the run-time for testing instances.

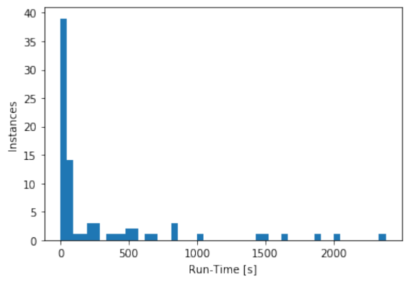


Figure 5: Histogram showing the distribution of the run time of the training instances

Following the configuration details and steps mentioned in §3.2, SMAC was implemented on the training instances. The optimal combination of parameter values that SMAC returned were the following:

- BranchDir (Branch direction preference)
    - Optimal = -1; default = 0
      The value of BranchDir -1 corresponds to always explore the down branch first.
- Heuristics (Turn MIP Heuristics up or down)
    - Optimal = 0.0817; default = 0.05
      The value of Heuristics 0.0817 is larger than 0.05, which means that spending more time on solving the heuristics and selecting the right node will reduce the run time to solve the MIP model.
- VarBranch (Branch variable selection strategy)
    - Optimal = 1; default = -1
      The value of VarBranch 1 means that Pseudo Shadow Price Branching is the optimal way of choosing which variable to branch on next.

The results of the overall algorithm were evaluated in two steps - it was first established whether the SMAC had succeeded in reducing run-times of the MIPs over the training set as a validation step of the methodology, then the algorithm was applied to the testing set to ensure that the parameters chosen by SMAC were indeed successful in accelerating the solution of the MIP given a randomly generated terrain. In both steps the results were compared with the baseline MIP model. The new parameter settings for branch direction, time spent on heuristics, and branch variable selection strategy decreased the total run-time by 21%. A summary of the total and average run-times for the MIPs in baselines and auto-tuned runs is tabulated in **Table 1**.

It is clear that there has been a decrease in run-time after auto-tuning in both the training and testing sets, with total run-time for training being reduced from 14550 s to 11147 and, for testing, 7485 s to 4140 s. The geometric averages for the two reductions in run-time are 29 s to 23 s, and 25 s to 22 s. The averages for the testing set are higher than those of the training set due to the sample size and skewness in distribution of the individual run-times (which was exponentially distributed, as mentioned earlier).

The effect of the newly-chosen parameters on individual MIP run-times was also investigated. A scatter plot of run-times with auto-tuned parameters against run-times with default parameters is displayed in **Figure A.6a**. Note that we used the log of the run time because of the exponential difference among the run times of the instances, which

Summary of Results				
	Baseline		Auto-Tuned	
	Total	Geometric Average	Total	Geometric Average
Run-Time (Training)	14550 s	29 s	11147 s	23 s
Run-Time (Testing)	4785 s	25 s	4140 s	22 s

Table 1: Total run-times of training and testing instances performed by `Gurobi` with both default and auto-tuned parameters.

helps in better comparing the performances of the models. Most points in the plot can be seen below the diagonal line, indicating that the newly-chosen parameters were successful in reducing run-times for most individual MIP runs.

Phase 2 Results: The procedure of analysis for the results of phase 2 was followed as in phase 1, with the main distinction being that the baseline is now the auto-tuned model from phase 1. As a first step, SMAC was implemented on the same training instances from phase 1 with the objective of reducing the run-time. As expected, the optimal combination of parameter values is different in this phase following the configurations details and steps mentioned in §3.2:

- BranchDir (Branch direction preference)
  - Optimal = 0; default = 0
    The value of BranchDir 0 corresponds to automatically choose if the down or up branch should be explored first.
- Heuristics (Turn MIP Heuristics up or down)
  - Optimal = 0.0978; default = 0.05
    The value of Heuristics 0.0817 is larger than 0.05, which means that spending more time on solving the heuristics and selecting the right node will reduce the run time to solve the MIP model. Interestingly, this value is larger than the one in phase 1, meaning with this new combination, it is better to increase the time spent on heuristics even further.
- VarBranch (Branch variable selection strategy)
  - Optimal = 1; default = -1
    The value of VarBranch 1 means that Pseudo Shadow Price Branching is the optimal way of choosing which variable to branch on next.
- SubMIPNodes (Number of nodes explored by MIP-based heuristics)
  - Optimal = 307; default = 500
    The value of SubMIPNodes 307 means that reducing the number of nodes explored in MIP-based heuristics will reduce the run time to solve the MIP model.

This optimal parameter configuration was found on SMAC on the 103$^{\text{rd}}$ evaluation, meaning that for future purposes, the maximum number of evaluations can be decrease to 150.

The results of the overall algorithm were evaluated on the testing instances to ensure that the parameters chosen by SMAC were successful in accelerating the solution of the MIP given a randomly generated terrain, and they were compared with the new baseline. The run-times of the auto-tuned model were plotted vs. run-times of the new baseline model (as displayed in **Figure A.6b**). Most points in the plot can be seen below the diagonal line, indicating that the new auto-tuned model is performing better by reducing the run-time of the instances in the testing set.

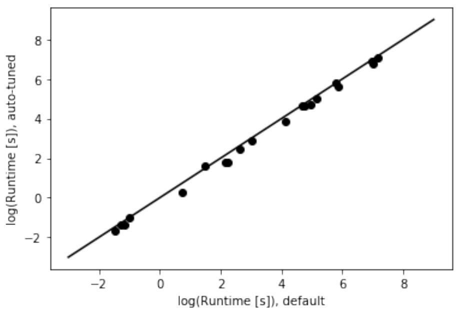
As a second step, the baseline and auto-tuned models were run on additional instances to test if the model was able to generalize to other instances generated from (1) the same terrain, but with a higher grid resolution, and (2) a different terrain but with the same resolution. The results can be found in **Figure A.7a** and **Figure A.7b**. As shown in those figures, most points are on the diagonal line, meaning the tuning was not able to be generalized to other instances that have variations within the generation of the data such as differences in the number of hills or the number of locations.

6. Conclusion

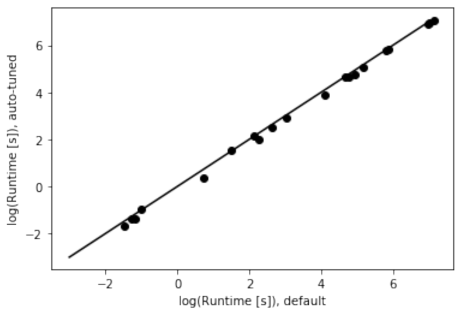
A novel machine learning approach to accelerating an MIP algorithm for wind farm layout optimization is presented. To implement the approach, a simplified wind farm data generator is assembled that can provide the wind speeds that serve as input to the objective function of the governing MIP, which is solved using `Gurobi`. Two sets of parameters of `Gurobi`'s MIP solver were tuned using Sequential Model-Based Algorithm Configuration with the objective of reducing solver run time; the first set consisted of the parameters BranchDir, Heuristics and VarBranch, and the second set included the latter variables along with SubMIPNodes.

SMAC was found to be successful in reducing the run-times of the testing instances whose terrains were of the same form as the training instances. This held true for both cases of tuning. However, the effects of parameter tuning did not generalize to terrains that were not of the same form as those using in training, or in instances with higher grid resolution (i.e. with a larger number of possible turbine locations).

Appendix A. Figures for Results

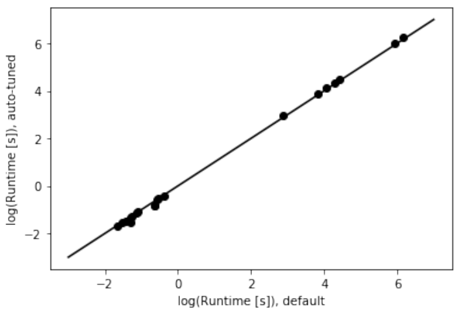


(a) Phase 1

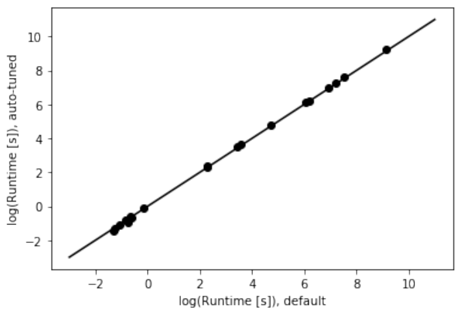


(b) Phase 2

Figure A.6: Run-Time in `Gurobi` with Default Parameters vs Run-Time in `Gurobi` with Optimized Parameters - (a) Phase 1 (b) Phase 2



(a) Same Terrain, Higher Resolution



(b) Different Terrain, Same Resolution

Figure A.7: Run-Time in `Gurobi` with Default Parameters vs Run-Time in `Gurobi` with Optimized Parameters - (a) Same Terrain, Higher Resolution (b) Different Terrain, Same Resolution

References

[1] J. Y. . Kuo, D. A. Romero, J. C. Beck, and C. H. Amon, "Wind farm layout optimization on complex terrains – Integrating a CFD wake model with mixed-integer programming," Applied energy, vol. 178, pp. 404–414, 2016, doi: 10.1016/j.apenergy.2016.06.085.

[2] P. Y. Zhang, "Topics in Wind Farm Layout Optimization: Analytical Wake Models, Noise Propagation, and Energy Production.," Thesis (M.Appl.Sc.)–University of Toronto (Canada), 2013., 2013.

[3] G. Mosetti, C. Poloni and B. Diviacco, "Optimization of wind turbine positioning in large windfarms by means of a genetic algorithm," Journal of Wind Engineering and Industrial Aerodynamics, vol. 51, (1), pp. 105-116, 1994.

[4] S. A. Grady, M. Y. Hussaini and M. M. Abdullah, "Placement of wind turbines using genetic algorithms," Renewable Energy, vol. 30, (2), pp. 259-270, 2005.

[5] M. Wagner, J. Day and F. Neumann, "A fast and effective local search algorithm for optimizing the placement of wind turbines," Renewable Energy, vol. 51, pp. 64-70, 2013.

[6] V. Aristidis, P. Maria and L. Christos, "Particle Swarm Optimization (PSO) algorithm for wind farm optimal design," International Journal of Management Science and Engineering Management, vol. 5, (1), pp. 53-58, 2010.

[7] P. Y. Zhang et al, "Solving wind farm layout optimization with mixed integer programming and constraint programming," in Anonymous Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 284-299.

[8] S. D. O. Turner et al, "A new mathematical programming approach to optimize wind farm layouts," Renewable Energy, vol. 63, pp. 674-680, 2014.

[9] S. Donovan, "Wind Farm Optimization," in 40th Annual Conference, Operational Research Society of New Zealand, Wellington, New Zealand, 2005

[10] J. Y. J. Kuo, D. A. Romero and C. H. Amon, "A mechanistic semi-empirical wake interaction model for wind farm layout optimization," Energy (Oxford), vol. 93, pp. 2157-2165, 2015.

[11] J. Allen et al, "Wind Farm Simulation and Layout Optimization in Complex Terrain," Journal of Physics. Conference Series, vol. 1452, (1), pp. 12066, 2020.

[12] J. T. Linderoth and M. W. P. Savelsbergh, ”A Computational Study of Search Strategies for Mixed Integer Programming,” INFORMS Journal on Computing, vol. 11, (2), pp. 173-187, 1999.
[13] Wolsey, Laurence A. “Mixed Integer Programming.” Wiley Online Library, John Wiley & amp, Sons, Ltd, 15 Sept. 2008.
[14] A. H. Land and A. G. Doig, ”An automatic method for solving discrete programming problems,” in Anonymous Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 105-132.
[15] https://www.cs.ubc.ca/labs/beta/Projects/SMAC/papers/11-LION5-SMAC.pdf
[16] https://www.cs.ubc.ca/labs/beta/Projects/SMAC/
[17] Hoos, Holger H. “Automated Algorithm Configuration and Parameter Tuning.” Autonomous Search, Springer Berlin Heidelberg, pp. 37–71
[18] IBM, 2013 IBM ILOG CPLEX 12.6 User Manual IBM Corp. (2013)
[19] Gurobi Optimization, LLC, “Gurobi Optimizer Reference Manual”. 2021.