Multi Tree Operators for Genetic Programming to Identify Optimal Energy Flow Controllers

Kathrin Kefer
Fronius International GmbH
Thalheim, Austria
kefer.kathrin-maria@fronius.com

Roland Hanghofer*
Research & Develoment FH OÖ
Forschungs und Entwicklungs GmbH,
Research Group Heuristic and
Evolutionary Algorithms Laboratory
Hagenberg, Austria
Roland.Hanghofer@dynatrace.com

Patrick Kefer
Research & Develoment FH OÖ
Forschungs und Entwicklungs GmbH,
Research Group ASIC
Wels, Austria
patrick.kefer@fh-wels.at

Markus Stöger
Bernd Hofer
stoeger.markus@fronius.com
hofer.bernd@fronius.com
Fronius International GmbH
Thalheim, Austria

Michael Affenzeller
Stephan Winkler
michael.affenzeller@fh-hagenberg.at
stephan.winkler@fh-hagenberg.at
Research & Develoment FH OÖ
Forschungs und Entwicklungs GmbH,
Research Group Heuristic and
Evolutionary Algorithms Laboratory
Hagenberg, Austria

ABSTRACT

Genetic programming is known to be able to find nearly optimal solutions for quite complex problems. So far, the focus was more on solution candidates that hold just one symbolic regression tree. For complex problems like controlling the energy flows of a building in order to minimize its energy costs, this is often not sufficient. This is why this work presents a solution candidate implementation in HeuristicLab where they hold multiple symbolic regression trees. Additionally, also new crossover and mutation operators were implemented as the existing ones cannot handle multiple trees in one solution candidate. The first type of operators applies them on all trees in the solution candidate, whereas the second one only applies them to one of the trees. It is found that applying the mutator to only one of the trees significantly reduces the training duration. Applying the crossover to one of the trees instead of all needs longer training times but can also achieve better results.

CCS CONCEPTS

• **Computing methodologies** → **Optimization algorithms**; **Genetic programming**; *Genetic algorithms*; • **Mathematics of computing** → **Genetic programming**.

*Now working at Dynatrace Austria GmbH, but was with the stated instititution at the time of this work's development.

GECCO '21 Companion, July 10–14, 2021, Lille, France

ACM ISBN 978-1-4503-8351-6/21/07…$15.00
https://doi.org/10.1145/3449726.3463181

KEYWORDS

Genetic Programming Operators; Symbolic Regression

ACM Reference Format:
Kathrin Kefer, Roland Hanghofer, Patrick Kefer, Markus Stöger, Bernd Hofer, Michael Affenzeller, and Stephan Winkler. 2021. Multi Tree Operators for Genetic Programming to Identify Optimal Energy Flow Controllers. In *2021 Genetic and Evolutionary Computation Conference Companion (GECCO '21 Companion), July 10–14, 2021, Lille, France.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3449726.3463181

1 INTRODUCTION

Genetic programming is nowadays well known to be able to find nearly optimal solutions for quite complex problems. One example for such complex problems is controlling the energy flows of a building in order to minimize its energy costs using energy management systems. As more and more renewable energy sources are built in order to drive forward the energy transition and slow down the global warming, such systems have become increasingly important during the last years. However, renewable energy sources are subject to constantly changing environmental conditions which cause their energy production to fluctuate a lot. Therefore, energy management systems which use, store and distribute this produced energy as intelligent and efficient as possible become more important. In recent years there has been an increasing tendency to use meta-heuristic algorithms for that instead of common control algorithms like rule based optimizers (e.g. [6, 23]), model predictive controls (e.g. [3, 14]) or linear programming algorithms (e.g. [4, 5]). So far, the focus has mainly been on swarm-based algorithms, i.e. the Particle Swarm Optimization (PSO). One example was published by Pedrasa et al. in 2010, who use an improved PSO algorithm to optimize the schedule of energy services for net benefits [20]. In 2016, Eseye et al. published their work on optimizing the energy management system for an isolated industrial microgrid for minimal

energy costs and maximum economical benefit using a modified Particle Swarm Optimization algorithm [9]. Despite the PSO algorithms, also genetic algorithms were used to optimize or manage energy sources and loads. Two examples for that were presented by Morganti et al. in 2009 [19] and by Soares et al. in 2017 [26]. However, none of them used genetic programming to optimize such a system. Due to that, also the appropriate genetic programming methods were not available so far.

In genetic programming, new solution candidates are generated using operators like crossover and mutation, where especially the crossover operator is known to have a big influence on the optimization ability [18, 27, 32]. This means that the better the operators work, also the better the found solution candidates will be. So far, the focus in genetic programming was mainly on solution candidates, that hold just one symbolic regression tree. For that case, various different crossover strategies were already implemented. The first one, the standard crossover developed by John Koza in 1992 [16], selects two parent solution candidates and for each parent and randomly chooses a cut off point at the included tree. The branch cut off from the first parent is then attached to the cut off point of the branch of the second parent and vice versa. In that way, two new children solution candidates are produced, which are then, together with all the other generated children solution candidates, evaluated for their fitness and become part of the new parent population if they are good enough [16]. Besides this standard crossover, multiple other crossover were developed, e.g. the single point crossover by Poli and Langdon [22], where the cut off point for the branches of the two parents is only chosen once and then used for both parents, or the semantic crossover from Ngyen et al. [29]. A size fair crossover, where the cut off branches have the same size, and a homologous crossover based on this size fair crossover were developed by Langdon in 2000 [17]. However, these existing techniques are not sufficient for the described energy flow optimization problem as the solution candidates only contain one symbolic regression tree while the problem proposed in this work requires multiple trees per solution candidate.

One further development of the original genetic programming that uses multiple trees in one solution candidate is the so-called multigene symbolic regression. There, the multiple trees contained in one solution candidate are each referred to as one gene. The overall result of the symbolic regression model is then calculated using a weighted linear combination of the outputs from these trees. The linear coefficients are therefore estimated using ordinary least squares techniques on the training data [24]. Sheta et al. used exactly this technique for stock market predictions in their work from 2015 [25]. This multigene symbolic regression is very similar to the approach developed for this work, however, here no weighted linear combination of the trees could be used due to the trees representing some specific system parameters that should be optimized.

This is why this work focuses on the extension and further development of these existing techniques in order to be able to optimize the energy flows of a building and minimize its energy costs. For that, the existing solution candidate representation is extended to be able to hold not only one but multiple symbolic regression trees, where each tree models the behaviour of one parameter of the system that should be optimized. In order to use genetic programming to solve this energy flow optimization problem, also two new types of crossover and mutation operators, which are able to handle the multiple trees contained in one solution candidate, were developed and implemented in the optimization framework HeuristicLab [31]. The first type of genetic operators applies the crossover and mutator on all symbolic regression trees in the solution candidate, whereas the second one only applies them to one of the trees. As the changes from generation to generation using the first operator type are quite big, it is expected that the optimization will converge faster than with the second type of operators. But as the second operator type allows a more fine-grained optimization of the system, it should find the overall better solutions for the problem.

The remaining work is structured as follows: chapter 2 describes the developed approach together with the crossover and mutation operators needed to perform the genetic programming, followed by their evaluation in chapter 3. The last two chapters 4 and 5 show the results of the evaluation and the conclusions drawn from them.

2 APPROACH

As mentioned in the introduction, for this approach the number of trees in one solution candidate was increased. For that, the data structure holding the one symbolic regression tree was adapted to be encapsulated in a list, so that any number of trees can be added to one solution candidate. In the course of this development, also the two main operators for genetic programming, the crossover and mutation operators needed to be adapted. The first implemented type of the operators applies the crossover and mutation on all trees in the solution candidate, whereas the second operator type allows a more fine-grained optimization due to only applying them on one of the trees. All operators are explained in more detail below.

2.1 All Trees Operators

The *All Trees* operators are based on the subtree swapping crossover and the following four different mutators, which are applied randomly at the specified mutation rate:

- Change node type mutator: mutates one of the symbols in the symbolic regression tree
- Replace branch mutator: removes a branch of the tree and replaces it with a randomly initialized one
- One-point shaker: changes one of the parameters in the symbolic regression tree
- Full-tree shaker: applies a uniform mutation on the parameters of the symbolic regression tree

The *All Trees* operators apply the crossover and chosen mutator on all trees contained in the solution candidates by leaving the actual operator implementation as it is and just looping over all trees contained in the solution candidates and applying the operators.

2.2 Single Tree Operators

For the *Single Tree* operators, the algorithm first decides on which of the symbolic regression trees contained in the solution candidate the crossover and mutator should be applied. This is done by randomly choosing the one tree to which the operators should be applied and writing its name into a list holding all tree names to which the operators should be applied. All other trees from the solution candidate are then marked to be ignored by writing their

names into an *ignore* list. These lists are then used as decision basis during the actual application of the operators by applying the subtree swapping crossover as well as the four mutators described before on only the tree specified in the *apply* list. For the remaining trees included in the *ignore* list the algorithm decides based on its settings which tree to use for the child solution candidate. There, three different settings options are available: always using the trees from the first parent, always using the trees from the second parent or using the trees from a randomly chosen parent solution candidate. This makes sure that the child solution candidate contains all necessary symbolic regression trees and that the change from the parents to the children is not as big as when applying the operators to all trees contained in the solution candidates.

3 EVALUATION

The newly implemented operators are evaluated using a further development [13] of the model-based energy flow optimization approach developed by Kefer et al. [12]. There, three parameters of a simulation model of a real-world building, which is explained together with these parameters in detail in chapter 3.1, are optimized to minimize the system's energy costs as shown in equation 1 with the help of a close cooperation between the optimization framework Heuristiclab [31] and MATLAB Simulink [28].

$$\text{min. } f(x) \text{ where } x = costs_{total} \tag{1}$$

For that, HeuristicLab initiates the C-code generation from a Simulink model, extends the generated code with additional functionality and then generates a DLL from that. This DLL is then used during the training process to evaluate the solution candidates by forwarding the solution candidate symbolic regression trees exported as formulas together with the input values to the DLL, running the DLL to simulate the model and then reading the quality of the solution candidate in form of the energy costs of the system back to HeuristicLab. Based on that quality value, the next generation's solution candidates are selected [13].

For the evaluation, in total 40 controllers are trained using the first day of the artificially generated data basis explained in chapter 3.2, two different genetic algorithms and four operator configurations (table 1). The first algorithm is the Offspring Selection Genetic Algorithm (OSGA) implemented by Affenzeller et al. [1]. It is a single-objective optimization algorithm that just aims at minimizing the energy costs of the simulated building. This algorithm was configured to use a mutation probability of 30%, maximum 100 generations and 250 000 evaluated solutions, a maximum selection pressure of 100 which serves also as the training termination criterion, a population size of 500 with 1000 selected parents and as selector the *GenderSpecific Selector* [30] with a *ProportionalSelector* as female and a *RandomSelector* as male selector.

The second algorithm used is the further development of Kommenda et al. [15] of the well-known Non-Dominated Sorting Genetic Algorithm II (NSGA-II) originally developed by Deb et al. [8]. This algorithm tries not only to minimize the energy costs but contemporaneously also the complexity of the model, i.e. the symbolic regression trees contained in the solution candidates [15]. The NSGA-II was configured to use a population of 500 solution candidates with 1000 selected parents, a maximum of 100 generations which serves also as termination criterion, a crossover probability of 100% and a mutation probability of 30%. As selector, a *CrowdedTournamentSelector* [7] with a group size of six was used. These two genetic algorithms were chosen because of the OSGA being a further development of the normal genetic algorithm, which promises to find better solutions for the given problem due to its self-adaptive offspring selection scheme. However, as this algorithm is a single-objective one, the NSGA-II was chosen as second genetic algorithm to be used to also evaluate the influence of the new operators on a multi-objective optimization and the complexity of the controllers.

Configuration	Crossover	Mutator
#1	All Trees	All Trees
#2	All Trees	Single Tree
#3	Single Tree	All Trees
#4	Single Tree	Single Tree

Table 1: The operator configurations used to conduct the experiments and evaluations.

As maximum symbolic regression tree depth 50 and as maximum tree length 100 was chosen and it was specified that when using the *Single Tree* operators the not-changed syntax trees should always be taken from the first parent solution candidate in order to ensure a good comparability of the four operator configurations. Besides that, the following mathematical operators were used as grammar for the symbolic regression: the four arithmetic functions addition, subtraction, multiplication and division, the trigonometric functions sine, cosine and tangent, exponential and logarithm operators and the power functions square, power, square root and root. The newly implemented crossover and mutation operator implementations, the *All Trees* operators and the *Single Tree* operators, are compared with each other using the four experiment configurations shown in table 1.

The 40 trained heuristic controllers are evaluated towards their ability to optimize the system proposed in section 3.1 by taking a closer look on their training performance including the best training result, the number of evaluated solutions and needed generations as well as the symbolic regression tree depth and length and also testing them for their achieved energy costs in simulation using the same simulation model that was also used for training.

3.1 Evaluation Model

The evaluation model was built in MATLAB Simulink and models the electrical (Fig. 1) and thermal (Fig. 2) energy flows of a real-world building in which currently a school is located. The electrical parts of the system include a hydroelectric power plant, whose energy production is abstracted in the model by just adding it to the household load, a 1.5 kWp photovoltaic (PV) system, a 12kWh battery storage and an Ohmpilot device[10], which turns electric power into hot water using four 9 kW heating rods.

The thermal part of the system models the building itself in form of a one-node simulation model together with its space heating, the hot water boiler which can be heated up using the heating rods controlled by the Ohmpilot or the oil heating, and the respective oil heating using the Carnot 2016b blockset [11]. As input values for the model the energy consumption and feed-in tariff, the household

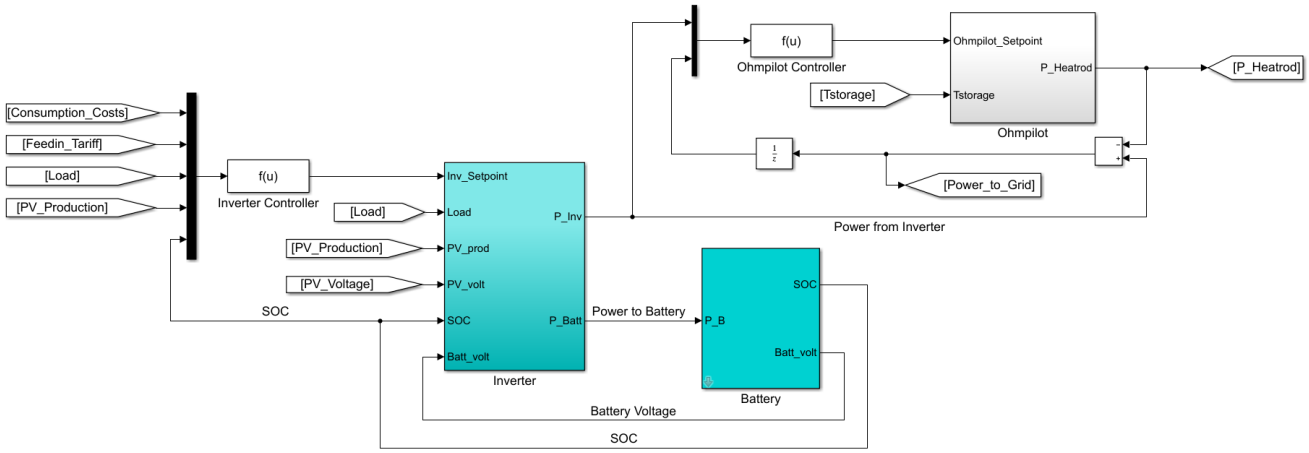

Figure 1: The electrical part of the simulation model with the inverter, the battery and the Ohmpilot. The *Inverter Controller* gets the energy tariffs, the household load, the production from the PV system and the state of charge of the battery as inputs to calculate the respective set point. For calculating the Ohmpilot set point, the *Ohmpilot Controller* uses the currently available power from the inverter and the energy consumed or fed into the grid at the previous time step. Using the Ohmpilot, the heating rods then heat up the hot water boiler of the system as shown in the thermal part of the system in Figure 2.

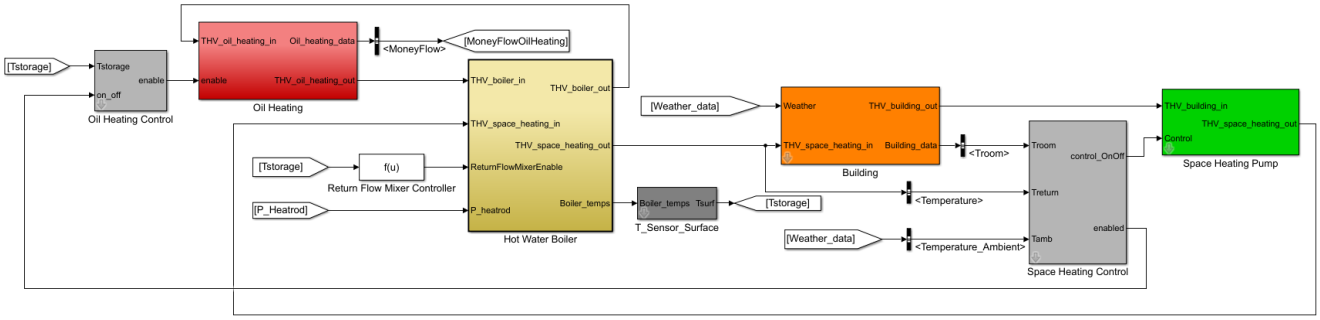

Figure 2: The thermal part of the simulation model gets the power for the heating rods from the Ohmpilot as shown in Figure 1 and heat up the hot water boiler according to that. Despite that, also the oil heating is started by its control if the boiler temperature drops below a certain threshold. The water from the hot water boiler is then used for the hot water supply in the building and by the space heating control, which decides whether the building needs to and can be heated up or not based on the room temperatures, the boiler temperatures and the outside temperature. For simplicity reasons, this graphic of the model was slightly adapted from the original one, but the blocks and the connections between them were kept the same.

load including the hydroelectric power plant production, the energy production of the PV system, the voltage of the PV system as well as the weather data is needed. The three parameters of the system that are optimized by the trained controllers are the grid feed-in set point of the inverter using the *Inverter Controller* block, the *Ohmpilot Controller*, which calculates the energy available for heating up the hot water boiler using the heating rods (Fig. 1) and the *Return Flow Mixer Controller*, which enables or disables the return flow of the hot water boiler (Fig. 2).

From the input data, the energy tariffs, the household load, the PV system production and, in addition to that, the current state of charge of the battery are used to calculate the optimal grid feed-in set point for the inverter in the *Inverter Controller* block as shown in Fig. 1. The *Ohmpilot Controller* uses the energy provided by the inverter together with the energy fed into the grid at the previous simulation step to calculate the current power available for the heating rods and the *Return Flow Mixer Controller* enables or disables the mixer based on the current storage temperatures. As simulation result, the energy costs caused by the system are calculated by multiplying the power consumed from and fed into the grid from the *Power_to_grid* signal in Fig. 1 with the respective energy tariffs and summing them up together with the energy costs

caused by the oil heating using the *MoneyFlowOilHeating* signal in Fig. 2 as shown in equation 2. There, N denotes the number of total simulation steps while t is the current step.

$$costs_{total} = \sum_{t=1}^{N} P_{fromGrid}(t) \times costs_{consumption}(t) - P_{toGrid}(t) \times costs_{feedin}(t) + costs_{oilHeating}(t) \quad (2)$$

3.2 Data Basis

As data basis for the training and evaluation, artificially generated data for one year was used. This data is based on real world data and covers all input values needed to simulate the previously explained evaluation model, including the PV system production and voltage, the household load, the production of the hydroelectric power plant, the variable energy tariffs for consumption and feed-in and the respective weather data. The methods with which this data is generated is explained in more detail below.

3.2.1 PV production and Voltage data. The generation of the PV production and voltage data is based on recorded weather data from 2018 and a MATLAB Simulink simulation model parametrized with the exact PV parameters from the real world building. The weather data was recorded by the company Meteonorm for the year 2018 in ten different locations in the vicinity of the real world building. For the data generation, two weather datasets are randomly selected from this available ones and are averaged. If the data to be generated should have a length of up to one year, the averaged weather data is shortened to the desired length. If the data should be longer than 365 days, the selection and averaging of the weather datasets a is repeated until the desired length is reached. This artificially generated weather data file is then used as input for the PV simulation model. The simulation of this model then generates the desired PV production and voltage data.

3.2.2 Hydroelectric Power Plant Production. To generate the artificial hydroelectric power plant production data measured production and loss data of another real-world hydroelectric power plant just a few kilometres upstream from the original site from 2010 until 2015 are used. They are split up into single days and grouped by their month of measurement. Starting with the month in which the newly generated data should start, the days to be used are selected randomly from the pool of data. The loss data gets subtracted from the respective production data, which results in the net production which gets then appended to its predecessor until the desired length of the new dataset is reached.

3.2.3 Household load. For the household load data generation, the program LoadProfileGenerator [21] (LPG) is used. There, the system to be optimized was modelled as precisely as possible and adjusted to the approximately same annual energy consumption. In addition, input data such as production of the hydroelectric power plant and weather data was added. Subsequently, a realistic load curve for the specific real world building can be generated with the LPG for different years and times.

3.2.4 Variable Energy Tariffs. For the generation of a new energy tariffs dataset, aWATTar [2] data from the years 2015 to 2018 are split up month-wise. From this data pools, one set is randomly selected from each of the different months and appended to the previous ones until the desired data set length is reached. This is done for the energy consumption costs and the feed-in tariffs.

4 RESULTS

This chapter presents the results of the evaluation of the four different crossover and mutator configurations as explained in chapter 3 and shown in table 1. First, the training metrics are presented in section 4.1, followed by the results for evaluation of the optimization ability in section 4.2.

4.1 Training Metrics

Taking a closer look on the training metrics achieved by the controllers trained with the OSGA and which are shown in table 2, it can be found that the *Single Tree* mutator reduces the number of generations and therefore also of the evaluated solutions. Using the *All Trees* crossover together with the *Single Tree* mutator instead of the *All Trees* mutator reduces the evaluated solutions on average from 261 300 to 70 100 and the needed generations from 25.8 to 3.4. Using the *Single Tree* mutator together with the *Single Tree* crossover instead of the *All Trees* crossover reduces the evaluated solutions on average from 260 500 to 77 300 and the generations from 25 to 3.4.

Configuration	Controller	Generations	Evaluated Solutions
All Trees crossover, All Trees mutator	1	27	275000
	2	28	262500
	3	24	263000
	4	23	253000
	5	27	253000
All Trees crossover, Single Tree mutator	1	3	61000
	2	3	64500
	3	3	66000
	4	4	74000
	5	4	85000
Single Tree crossover, All Trees mutator	1	29	265500
	2	26	263000
	3	23	254000
	4	23	267000
	5	24	253000
Single Tree crossover, Single Tree mutator	1	3	62000
	2	4	80000
	3	4	96000
	4	3	82000
	5	3	66500

Table 2: The training metrics for the five controllers trained with the OSGA for each configuration including the number of evaluated solutions and needed generations.

As shown in table 3, the best results, i.e. the minimum energy costs, during the training are achieved with both algorithms with the *All Trees* crossover. There, the average energy costs are around

2.68€ when trained together with the *All Trees* mutator and both training algorithms or the NSGA-II and the *Single Tree* mutator. The controllers trained with the *All Trees* crossover, the *Single Tree* mutator and the OSGA, however, perform only slightly worse with average energy costs of 3.03€ for the one training day. For these configurations, also the respective standard deviations are with a maximum of 0.0386€ very low, which indicates that the training process is very stable and that all trained controllers work equally well. However, taking a closer look on the training results for configurations #3 and #4 where the *Single Tree* crossover is used together with the two different mutators, it can be found that, except for the controllers trained with the *All Trees* mutator and the OSGA, which work equally good as the controllers trained with the *All Trees* crossover and the *All Trees* mutator (configuration #1), the average energy costs are worse and the respective standard deviations are also much higher. The average energy costs for the controllers trained with the *Single Tree* crossover and the NSGA-II rise to 7.64€ with a standard deviation of 7.15€ when using the *All Trees* mutator (configuration #3) and to 6.67€ with a standard deviation of 7.34€ when using the *Single Tree* mutator (configuration #4), whereas the controllers trained with the OSGA and configuration #4 even have average training energy costs of 8.27€ with a standard deviation of 3.75€.

Configuration	Training Algorithm	Best Result	Average Result	Standard Deviation
All Trees crossover, All Trees mutator	NSGA-II	2.6808	2.6834	0.0024
	OSGA	2.6826	2.6838	0.0011
All Trees crossover, Single Tree mutator	NSGA-II	2.6840	2.6857	0.0022
	OSGA	2.9552	3.0299	0.0386
Single Tree crossover, All Trees mutator	NSGA-II	2.7665	7.6417	7.1511
	OSGA	2.6813	2.6820	0.0005
Single Tree crossover, Single Tree mutator	NSGA-II	2.8939	6.6660	7.3411
	OSGA	3.0551	8.2697	3.7471

Table 3: The best and average energy costs of the trainings with their respective standard deviations in € for the five controllers trained with the two genetic algorithms and the different operator configurations.

For the evaluation of the symbolic regression tree depths and lengths, the depth is calculated by running from the top most node that holds a mathematical symbol straight down to the bottom most leaf node and counting each of them except for the *StartSymbol* node added by HeursticLab. The length of the symbolic regression tree is calculated by counting every single node contained in the tree including the leaf nodes but also except for the *StartSymbol*. As shown in table 4, in general the trees for the *Inverter* and *Ohmpilot Contorllers* are less deep and also shorter than the tree for the *Return Valve Controller*, while a difference between the *Inverter* and *Ohmpilot Controller* cannot be explicitly determined. This might be caused by the different number of input values that are used for the controllers, e.g. five by the *Inverter Controller*, two by the *Ohmpilot Controller* and only one by the *Return Valve Controller*.

Comparing the average tree depths and lengths of the trainings done with the *All Trees* crossover (configurations #1 and #2), it is found that when using the NSGA-II algorithm and the *Single Tree* mutator instead of the *All Trees* mutator together with the *All Trees* crossover, the depths and lengths are decreased for all three trees contained in the solution candidate. However, when using the OSGA as training algorithm, the average depth and length of the *Inverter Controller* tree is increased, while there are still reductions for the *Ohmpilot* and *Return Valve Controller* trees. Using the *Single Tree* crossover instead of the *All Trees* crossover together with the *All Trees* mutator (configuration #3), does not have an effect on the trees that are trained with the NSGA-II. There, the depth is increased for the *Inverter Controller* and *Return Valve Controller* trees, while it is reduced for the *Ohmpilot Controller* tree. The tree lengths, however, are reduced for the *Inverter Controller* and *Ohmpilot Controller* trees, while they are increased for the *Return Valve Controller* tree. In contrast to that, the depths and lengths of the trees trained with the OSGA are similar (*Inverter Controllers* tree) or even reduced (*Ohmpilot* and *Return Valve Controllers* trees). Comparing the results for configurations #1 (both *All Trees* operators) and #4 (both *Single Tree* operators), similar results for the NSGA-II trained controllers can be found: for the *Inverter* and *Return Valve Controller* trees, the depth is increased and the length is decreased, while both, the average depth and the length are decreased for the *Ohmpilot Controller* tree. The average depth and length of the *Inverter Controller* trees trained with the OSGA and both *Single Tree* operators is bigger while they are reduced for the *Ohmpilot* and *Return Valve Controller* trees.

To sum up, using the *Single Tree* mutator shortens the training duration for the OSGA trained controllers by reducing the number of evaluated solutions and needed generations remarkably without having much influence on the training results. Using the *Single Tree* crossover, however, has a much bigger influence on the results. Due to the more fine-grained training that the *Single Tree* crossover performs, of course more generations and evaluated solutions are necessary to find near-optimal solutions. This is why when using this crossover in combination with the *Single Tree* mutator as done in configuration #4, the training results become worse. Despite that, when evaluating the controller depths and lengths no clear tendency for their reduction or increase could be found for the different operator configurations and algorithms.

4.2 Test Results Energy Cost Optimization

For the evaluation of the cost optimization abilities of the trained controllers, the energy costs for the system are calculated for different lengths by running the simulation with each controller for 30, 60, 180 and 364 days, starting at the day after the training (2nd

		Inverter Controller		Ohmpilot Controller		Return Valve Controller	
Configuration	**Training Algorithm**	Depth	Length	Depth	Length	Depth	Length
All Trees crossover, All Trees mutator	NSGA-II	10.40 (6.25)	47.20 (35.26)	21.20 (2.48)	91.00 (8.00)	19.40 (2.15)	75.00 (20.52)
	OSGA	6.40 (4.59)	14.20 (12.32)	17.60 (4.72)	70.00 (24.84)	23.20 (6.91)	64.80 (21.45)
All Trees crossover, Single Tree mutator	NSGA-II	5.80 (4.07)	17.60 (19.63)	15.40 (3.72)	64.80 (25.84)	18.20 (2.93)	73.40 (22.04)
	OSGA	9.00 (4.38)	27.20 (26.78)	6.60 (2.42)	16.00 (11.37)	17.00 (8.32)	41.40 (18.60)
Single Tree crossover, All Trees mutator	NSGA-II	12.60 (8.50)	32.80 (31.76)	5.60 (3.83)	9.20 (8.57)	24.20 (10.19)	78.60 (19.02)
	OSGA	6.40 (4.22)	16.20 (14.02)	14.20 (2.56)	43.00 (8.63)	22.40 (6.15)	58.00 (20.12)
Single Tree crossover, Single Tree mutator	NSGA-II	12.20 (9.36)	26.00 (25.24)	6.80 (3.19)	12.80 (7.57)	21.80 (7.30)	56.80 (22.61)
	OSGA	13.80 (8.23)	39.00 (29.48)	7.60 (3.32)	12.80 (8.03)	21.80 (12.02)	48.80 (22.09)

Table 4: The average tree depth and length together with the respective standard deviations in the brackets beside for the five controllers trained with each of the two different genetic algorithms and four operator configurations.

of January). It was expected that the controllers trained with configuration #4 (both *Single Tree* operators) will perform worst due to also having achieved the worst training results. However, the results presented in table 5 show that all controllers on average work pretty similar and nearly equally good, especially when running them for the longer evaluation periods like half a year or nearly a year. There, the controllers trained with the OSGA and the two *Single Tree* operators even achieve the overall best average energy costs of 10984.48€. However, these controllers are among the worst ones for the shorter training times up to 180 days, which indicates that there is a turnover point somewhere between 180 and 364 days. For the shorter training times, the controllers trained with configuration #3 (*Single Tree* crossover and *All Trees* mutator) and the NSGA-II perform best by achieving the overall best average energy costs of 1072.64€ for 30 days, 3062.71€ for 60 days and 5226.74€ for 180 days. Despite that, it is also found that the standard deviations for the energy costs achieved by the controllers trained with the *All Trees* crossover together with the *Single Tree* mutator and vice versa are much smaller than the ones achieved by the controllers trained with both *All Trees* operators or both *Single Tree* operators. Nevertheless, these results indicate that the *Single Tree* operators can achieve better results than the *All Trees* operators.

5 CONCLUSIONS

This work presents new operators for genetic programming to identify optimal energy flow controllers where the solution candidates contain multiple symbolic regression trees. The two different types of operators proposed are the *All Trees* operators, where the crossover and mutator is applied to every tree in the solution candidate, and the *Single Tree* operators, where the crossover and mutator is only applied to one randomly chosen tree while the others are either taken from one specified parent or randomly from both. As the *Single Tree* operators allow a more fine-grained optimization, it is expected that the results are better but the training might take longer. For the evaluation of the proposed operators, four different configurations were used, being bot *All Trees* operators, the *All Trees* crossover together with the *Single Tree* mutator, the *Single Tree* crossover together with the *All Trees* mutator and both *Single Tree* operators. They were each evaluated using two different genetic algorithms with which five controllers were trained to optimize the energy flows of a complex thermal-electrically coupled system in order to minimize the system's energy costs. The simulation-based evaluation shows that the *Single Tree* mutator is able to significantly reduce the training length by reducing the number of evaluated solutions and generations. Despite that, it is shown that the *Single Tree* crossover is able to achieve better optimization results than the *All Trees* crossover but needs longer training times for that.

Despite these promising results, the topic needs further investigations. First, a detailed evaluation of the influence of the parent from which the remaining trees are taken during the use of the *Single Tree* operators would be beneficial as in this work it was specified to always use the trees from the first parent solution candidate. Additionally, it would be interesting to implement a selection for the "better parent", where the remaining trees are not taken from either the first, the second a randomly chosen parent, but always from the better one. Another further development especially for this application area would be to implement a grouping possibility for the symbolic regression trees, e.g. into thermal and electrical, and then not just select one of the symbolic regression trees but all trees from a specific group for applying the operators.

ACKNOWLEDGMENTS

This project was financed by the European Regional Development Fund and the Province of Upper Austria. It was carried out by Fronius International GmbH and partners from the University of Applied Sciences Upper Austria.

REFERENCES

[1] Michael Affenzeller and Stefan Wagner. 2005. Offspring selection: A new self-adaptive selection scheme for genetic algorithms. In *Adaptive and natural computing algorithms*. Springer, 218–221.
[2] aWATTar. 2021. *aWATTar*. Retrieved April 27, 2021 from https://www.awattar.com/

Configuration	Training Algorithm	Simulation Days 30	60	180	364
All Trees crossover, All Trees mutator	NSGA-II	1525.86 (84.03)	3154.46 (100.42)	5407.43 (269.37)	10911.80 (323.29)
	OSGA	1491.72 (21.18)	3109.93 (46.05)	5308.59 (80.56)	11143.28 (147.77)
All Trees crossover, Single Tree mutator	NSGA-II	1483.96 (13.24)	3091.25 (30.76)	5275.51 (51.61)	11029.46 (98.13)
	OSGA	1532.85 (121.51)	3146.80 (171.50)	5405.67 (381.60)	10991.21 (54.37)
Single Tree crossover, All Trees mutator	NSGA-II	1472.64 (6.71)	3062.71 (10.70)	5226.74 (12.30)	11003.33 (32.56)
	OSGA	1478.90 (11.06)	3081.33 (25.48)	5254.44 (44.21)	11072.56 (92.48)
Single Tree crossover, Single Tree mutator	NSGA-II	1603.20 (143.69)	3250.44 (200.83)	5626.26 (442.86)	11028.47 (49.46)
	OSGA	1602.28 (134.07)	3251.59 (188.09)	5664.00 (436.55)	10984.48 (177.71)

Table 5: The average energy costs together with their standard deviations in the brackets beside in € for 30, 60, 180 and 364 days of simulation for the five controllers trained with each of the genetic algorithms and operator configurations.

[3] Chen Chen, Jianhui Wang, Yeonsook Heo, and Shalinee Kishore. 2013. MPC-based appliance scheduling for residential building energy management controller. *IEEE Transactions on Smart Grid* 4, 3 (2013), 1401–1410.

[4] Zhi Chen, Lei Wu, and Yong Fu. 2012. Real-time price-based demand response management for residential appliances via stochastic optimization and robust optimization. *IEEE Transactions on Smart Grid* 3, 4 (2012), 1822–1831.

[5] Francesco De Angelis, Matteo Boaro, Danilo Fuselli, Stefano Squartini, Francesco Piazza, and Qinglai Wei. 2012. Optimal home energy management under dynamic electrical and thermal constraints. *IEEE Transactions on Industrial Informatics* 9, 3 (2012), 1518–1527.

[6] Roel De Coninck, Ruben Baetens, Dirk Saelens, Achim Woyte, and Lieve Helsen. 2014. Rule-based demand-side management of domestic hot water production with heat pumps in zero energy neighbourhoods. *Journal of Building Performance Simulation* 7, 4 (2014), 271–288.

[7] Deb. 2001. Multi-objective optimisation using evolutionary algorithms. Vol. 16. John Wiley & Sons, 247.

[8] Kalyanmoy Deb, Amrit Pratap, Sameer Agarwal, and TAMT Meyarivan. 2002. A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE transactions on evolutionary computation* 6, 2 (2002), 182–197.

[9] A. T. Eseye, D. Zheng, J. Zhang, and D. Wei. 2016. Optimal energy management strategy for an isolated industrial microgrid using a Modified Particle Swarm Optimization. In *2016 IEEE International Conference on Power and Renewable Energy (ICPRE)*. 494–498. https://doi.org/10.1109/ICPRE.2016.7871126

[10] Fronius International GmbH. 2021. *Fronius Ohmpilot.* Retrieved April 27, 2021 from https://www.fronius.com/en/solar-energy/installers-partners/technical-data/all-products/solutions/fronius-solution-for-heat-generation/fronius-ohmpilot/fronius-ohmpilot

[11] Solar-Institut Juelich. 2018. CARNOT Toolbox Ver. 6.2 2016b.

[12] Kathrin Kefer, Roland Hanghofer, Patrick Kefer, Markus Stöger, Michael Affenzeller, Stephan Winkler, Stefan Wagner, and Bernd Hofer. 2019. A Model-Based Learning Approach for Controlling the Energy Flows of a Residential Household Using Genetic Programming to Perform Symbolic Regression. In *International Conference on Computer Aided Systems Theory*. Springer, 405–412.

[13] Kathrin Kefer, Roland Hanghofer, Patrick Kefer, Markus Stöger, Bernd Hofer, Michael Affenzeller, and Stephan Winkler. [n.d.]. Simulation-Based Optimization of Residential Energy Flows Using Genetic Programming to Solve a Symbolic Regression Problem. ([n. d.]). in preparation for submission to the Journal of Energy and Buildings.

[14] Fabian Kennel, Daniel Görges, and Steven Liu. 2012. Energy management for smart grids with electric vehicles based on hierarchical MPC. *IEEE Transactions on industrial informatics* 9, 3 (2012), 1528–1537.

[15] Michael Kommenda, Gabriel Kronberger, Michael Affenzeller, Stephan M Winkler, and Bogdan Burlacu. 2016. Evolving simple symbolic regression models by multi-objective genetic programming. In *Genetic Programming Theory and Practice XIII*. Springer, 1–19.

[16] John R Koza and John R Koza. 1992. *Genetic programming: on the programming of computers by means of natural selection*. Vol. 1. MIT press.

[17] William B Langdon. 2000. Size fair and homologous tree genetic programming crossovers. *Genetic programming and evolvable machines* 1, 1/2 (2000), 95–119.

[18] Sean Luke and Lee Spector. 1997. A comparison of crossover and mutation in genetic programming. *Genetic Programming* 97 (1997), 240–248.

[19] G. Morganti, A. M. Perdon, G. Conte, D. Scaradozzi, and A. Brintrup. 2009. Optimising Home Automation Systems: A comparative study on Tabu Search and Evolutionary Algorithms. In *2009 17th Mediterranean Conference on Control and Automation*. 1044–1049. https://doi.org/10.1109/MED.2009.5164684

[20] M. A. A. Pedrasa, T. D. Spooner, and I. F. MacGill. 2010. Coordinated Scheduling of Residential Distributed Energy Resources to Optimize Smart Home Energy Services. *IEEE Transactions on Smart Grid* 1, 2 (2010), 134–143. https://doi.org/10.1109/TSG.2010.2053053

[21] Noah Pflugradt. 2016. *Modellierung von Wasser und Energieverbräuchen in Haushalten*. Ph.D. Dissertation.

[22] Riccardo Poli and William B Langdon. 1998. Genetic programming with one-point crossover. In *Soft Computing in Engineering Design and Manufacturing*. Springer, 180–189.

[23] Jyri Salpakari and Peter Lund. 2016. Optimal and rule-based control strategies for energy flexibility in buildings with PV. *Applied Energy* 161 (2016), 425–436.

[24] Dominic P Searson, David E Leahy, and Mark J Willis. 2010. GPTIPS: an open source genetic programming toolbox for multigene symbolic regression. In *Proceedings of the International multiconference of engineers and computer scientists*, Vol. 1. Citeseer, 77–80.

[25] Alaa F Sheta, Sara Elsir M Ahmed, and Hossam Faris. 2015. Evolving stock market prediction models using multi-gene symbolic regression genetic programming. *Artificial Intelligence and Machine Learning* 15, 1 (2015), 11–20.

[26] A. Soares, Á. Gomes, C. H. Antunes, and C. Oliveira. 2017. A Customized Evolutionary Algorithm for Multiobjective Management of Residential Energy Resources. *IEEE Transactions on Industrial Informatics* 13, 2 (2017), 492–501. https://doi.org/10.1109/TII.2016.2628961

[27] William M Spears and Vic Anand. 1991. A study of crossover operators in genetic programming. In *International Symposium on Methodologies for Intelligent Systems*. Springer, 409–418.

[28] The Mathworks, Inc. 2020. *MATLAB version 9.3.0.713579 (R2020b)*. The Mathworks, Inc., Natick, Massachusetts.

[29] Nguyen Quang Uy, Nguyen Xuan Hoai, Michael O'Neill, Robert I McKay, and Edgar Galván-López. 2011. Semantically-based crossover in genetic programming: application to real-valued symbolic regression. *Genetic Programming and Evolvable Machines* 12, 2 (2011), 91–119.

[30] S. Wagner. 2005. SexualGA: Gender-Specific Selection for Genetic Algorithms.

[31] Stefan Wagner, Andreas Beham, Gabriel Kronberger, Michael Kommenda, Erik Pitzer, Monika Kofler, Stefan Vonolfen, Stephan Winkler, Viktoria Dorfer, and Michael Affenzeller. 2010. HeuristicLab 3.3: A unified approach to metaheuristic optimization. In *Actas del séptimo congreso español sobre Metaheurísticas, Algoritmos Evolutivos y Bioinspirados (MAEB'2010)*. 8.

[32] David R White and Simon Poulding. 2009. A rigorous evaluation of crossover and mutation in genetic programming. In *European Conference on Genetic Programming*. Springer, 220–231.