Automatic modeling of a gas turbine using genetic programming: An experimental study

Josué Enríquez-Zárate a,∗, Leonardo Trujillo b, Salvador de Lara c, Mauro Castelli d, Emigdio Z-Flores b, Luis Muñoz b, Aleš Popovič e

a Universidad Autónoma Metropolitana, Departamento de Energía, México, D.F., Mexico
b Tree-Lab, Posgrado en Ciencias de la Ingeniería, Instituto Tecnológico de Tijuana, Blvd. Industrial y Ave. ITR Tijuana S/N, Mesa de Otay, Tijuana, B.C. 22500, Mexico
c Instituto de Investigaciones Eléctricas (IIE), Cuernavaca, Morelos, Mexico
d NOVA IMS, Universidade Nova de Lisboa, 1070-312 Lisboa, Portugal
e Faculty of Economics, University of Ljubljana, Kardeljeva Ploščad 17, 1000 Ljubljana, Slovenia

A R T I C L E   I N F O

Article history:
Received 22 January 2016
Received in revised form 4 November 2016
Accepted 14 November 2016
Available online 21 November 2016

Keywords:
Gas turbine
Data-driven modeling
Genetic programming
Local search

A B S T R A C T

This work deals with the analysis and prediction of the behavior of a gas turbine (GT), the Mitsubishi single shaft Turbo-Generator Model MS6001, which has a 30 MW generation capacity. GTs such as this are of great importance in industry, as drivers of gas compressors for power generation. Because of their complexity and their execution environment, the failure rate of GTs can be high with severe consequences. These units are subjected to transient operations due to starts, load changes and sudden stops that degrade the system over time. To better understand the dynamic behavior of the turbine and to mitigate the aforementioned problems, these transient conditions need to be analyzed and predicted. In the absence of a thermodynamic mathematical model, other approaches should be considered to construct representative models that can be used for condition monitoring of the GT, to predict its behavior and detect possible system malfunctions. One way to derive such models is to use data-driven approaches based on machine learning and artificial intelligence. This work studies the use of state-of-the-art genetic programming (GP) methods to model the Mitsubishi single shaft Turbo-Generator. In particular, we evaluate and compare variants of GP and geometric semantic GP (GSGP) to build models that predict the fuel flow of the unit and the exhaust gas temperature. Results show that an algorithm, proposed by the authors, that integrates a local search mechanism with GP (GP-LS) outperforms all other state-of-the-art variants studied here on both problems, using real-world and representative data recorded during normal system operation. Moreover, results show that GP-LS outperforms seven other modeling techniques, including neural networks and isotonic regression, confirming the importance of GP-based algorithms in this domain.

© 2016 Elsevier B.V. All rights reserved.

1. Introduction

Gas turbines (GTs) are one of the key elements in the electrical power industry since they transform mechanical power into the desired electrical output. In particular, GTs are widely used in so-called Open Cycle Plants and Combined Cycle Plants. They are considered to be high-risk units since they operate at high speed, temperature and pressure, conditions that impose strict control requirements. The control system executes the startup and shutdown sequences, controls speed, power and temperature, while also performing functions related to safety and protection. Startup and shutdown sequences consist of a series of complex events where each step has to be satisfactorily completed before continuing to the next. Moreover, in central power generation it is of utmost importance to ensure high levels of availability and reliability of the generating units, particularly due to the high costs associated to system failure or an interruption in the availability of electrical power. Therefore, condition monitoring of GTs within power plants is highly desirable to guarantee the long-term safety of the units and their efficient operation [1].
For instance, a condition monitoring system can be connected in parallel to the control system for the early detection of potential abnormalities. This can allow supervisors to perform an orderly shutdown of the unit, thus avoiding unexpected stops or failures of a greater magnitude [1]. An important element of such a monitoring system is the model that is used to describe or predict the behavior of the unit. If the models are sufficiently accurate, they can be used to detect unwanted deviations from what is considered to be normal behavior. Indeed, many different physical models of GTs have been developed in the past [2]. These models capture the dynamics of the turbine, but their precision can significantly vary based on the specific unit that is used. In practice, it is not always possible to have complete mathematical models that describe all of the turbine behaviors accurately. Therefore, another alternative is to develop statistical or computational models that might provide a more accurate characterization of the behavior of such units.

Several authors have developed dynamic mathematical and computational models, as well as specialized test equipment for GTs [3]. The modeling of GTs for diagnostics and prognostics has been carried out with a variety of techniques, which can be categorized into two major groups: physics-based methods and data-driven methods. The physics-based methods are limited to the cases where failure mechanisms can be quantified [3], and require specialized personnel to aid in the modeling process as well as a detailed description of the particular system to be modeled with specific technical information from the manufacturer of the equipment, which are often not available. The data-driven methods attempt to derive the models directly from recorded system data without digging into the physics of the system, using tools derived from artificial intelligence, machine learning, statistical analysis and computational intelligence [3]. For instance, a popular approach is to use neural networks [4,5].

This work proposes the use of genetic programming (GP) [6–8] to model a Mitsubishi single shaft Turbo-Generator, Model MS6001, with a 30 MW generation capacity. GP is a data-driven approach, a form of evolutionary algorithm (EA) [8] intended for automatic program or model induction. GP has recently gained popularity given its ability to generate syntactic models in a wide variety of problem domains [9], in many cases outperforming more common machine learning paradigms [10–13] while using very little in the way of prior knowledge. While EAs have been used before in GT systems, they are mostly used for optimizing or improving system efficiency using previously derived models [14]. To the extent of our knowledge, this is the first work to address the GT modeling problem directly using a GP approach. Moreover, this work compares recent proposals in GP literature, including the fairly recent geometric semantic GP (GSGP) [15,16] and improved variants of standard GP [17,18]. The experimental work compares several variants of these methods, particularly emphasizing algorithms that combine the global search process provided by the basic GP paradigm with greedy local search operators. Such combinations have been shown to improve algorithm convergence and predictive accuracy [19], while also reducing model size and complexity [17,18]. Comparisons are also made with seven non-GP modeling methods, showing that GP search outperforms all other methods in terms of predictive accuracy and model size, particularly on the most difficult modeling task.

The remainder of this paper proceeds as follows. Section 2 describes the power generation process of GTs in general and the particular unit studied in this work, describing how data was collected to apply the data-driven approach. Section 3 provides a general overview of GP, focusing on the specialized variants employed in this work and developed by the authors. The experimental work and results are presented in Section 4. Finally, conclusions and future work are outlined in Section 5.

2. Gas turbine operation and case study

This section provides a brief introduction to GTs and how they are used for power generation, as well as a detailed description of how the dataset used in this study was extracted.

2.1. Process description

A GT power unit is mainly composed of a starting device, a compressor, a combustion chamber, a turbine, an electric generator and other auxiliary components such as a fuel system, a lube oil system, a starting device and other subsystems. A graphical depiction of a generic GT power unit is shown in Fig. 1.

The starting device is a diesel engine, it assists the GT unit during the startup phase to overcome inertia and to accelerate the GT up to the ignition speed. The centrifugal compressor pressurizes the incoming air and sends it to the combustion chamber. High-pressure combustion exhaust gases are expanded within the turbine stages to rotate the unit. The electric generator that is coupled to the GT can generate the desired electricity when it is operated at the rated speed. Finally, the combustion exhaust gases are discharged to the atmosphere [20].

2.2. Case study

The case study of the current paper considers a Mitsubishi single shaft Turbo-Generator Model MS6001, which has a 30 MW generation capacity. The plant is located in Cozumel, Quintana Roo, in the southeast of Mexico in the complex known as the Chankanab Turbo-Gas Central (see Fig. 2). For this model, the combustion zone is formed by 10 combustion chambers and nozzles with their respective baskets. It also has two spark plugs and two flame detectors. In this area the oxygen supplied by the compressor is combined with the diesel fuel delivered by spray nozzles. Afterward, the spark plugs ignite the mixed fuel to produce the required combustion, generating a large volume of gases.
The gases produced by the combustion pass through the turbine which consists of three stages of blades. The power of the exhaust gases is thus converted to mechanical energy. However, the temperature at the turbine inlet is not directly regulated or monitored. The control of the gas turbine is done by monitoring the exhaust temperature, which should not exceed 1067 °F. This temperature is then used to regulate the fuel supply to the turbine.

The liquid fuel system delivers a high pressure flow modulated by a servo controlled bypass valve assembly. The servo valve controls the bypass valve stroke according to the system requirements and the sensed fuel flow. Fuel oil goes to a flow divider whose purpose is the apportion of fuel oil to each of the fuel nozzles of the turbine. The distribution has ten pump elements (in line) with the inlet port located at the mid-point where the fuel oil enters the unit and is distributed by an internal manifold to the inlet side of each pump element. The flow from each pump element is proportional to the speed at which the unit operates.

Fuel flow is measured by magnetic pickups and represents the feedback signal in the outer control loop of the system. The speed of the flow divider is a direct measure of the fuel flow nozzles in the combustion chambers, as depicted in Fig. 3. The correct measurement of the fuel flow is crucial for a good control performance.

Concerning the exhaust gases temperature, it reflects the fuel flow variations; therefore, it is also critical for the control system to maintain a temperature profile that protects the unit from any thermal excess. In both cases (fuel flow and exhaust temperature), it is also desirable to have accurate predictive models that can be used as monitoring tools. These models can be used to detect deviations over time from the expected behavior of the GT unit, thus anticipating possible failure situations.

The unit is equipped with a control system that offers a user interface to operators. The interface collects data from the system using specialized acquisition modules, providing a historical database for analysis purposes. The monitoring process of this plant can therefore be used to collect a representative dataset related to flow and temperature in the GT. Table 1 shows the available data related to fuel flow that is provided by the system, while Table 2 shows the same for the exhaust gas temperature.

### 2.3. Problem statement and data

The goal of this work is to derive predictive models of both the fuel flow and the exhaust gas temperature. In particular, the intention of the fuel flow prediction problem is to find a function \( K_f \) that takes as inputs and produces as output an approximation of the actual fuel flow \( f_f \) of the GT unit; i.e., \( K_f(f_f, ts) = f_f \). Conversely, the goal of the exhaust gas temperature prediction problem is to find a function \( K_t \) that takes the fuel flow \( f_f \) and the compressor discharge pressure \( cp \) as inputs and produces as output an approximation of the actual exhaust gas temperature \( c_t \) of the GT unit; i.e., \( K_t(f_f, cp) = c_t \).

Both functions \( K_f \) and \( K_t \) should approximate the fuel flow and the exhaust gas temperature, respectively, within the normal operating ranges of the independent variables \( f_f, ts, f_f \) and \( cp \). In both cases, our intent was to use a small amount of independent variables that could be easily and reliably monitored, while capturing sufficient information to predict the dependent variables. Therefore, in both problems the desired models \( (K_f(f_f, ts) \) and \( K_t(f_f, cp)) \) take two measurements as inputs and produce a single output.

The GT unit was allowed to operate in normal conditions after a control tuning process during the commissioning phase. At this time, the above-mentioned independent variables were recorded. In the case of fuel flow prediction, the GT unit operated for approximately 23 min, using a sampling period of 100 ms, recording a total of 13,788 records (input/output pairs). For the prediction of the exhaust gas temperature, the unit operated for approximately the same amount of time with the same sampling period, producing a total of 13,820 records. These datasets allowed us to pose two supervised learning problems and solve them with GP, as will be described in the following section. Fig. 4 shows a graphical representation of each data set plotted as 3D point clouds, to visualize what is the behavior that needs to be modeled. Notice that in both cases the datasets show highly irregular behavior with some discontinuities.

### 3. Genetic programming

GP is part of the larger research area known as evolutionary computation, which deals with the development of global search and optimization algorithms that are designed based on some of the core principles of neo-Darwinian evolutionary theory [6–8]. However, the GP paradigm distinguishes itself from other EAs in several key respects. GP is intended to solve problems that can be broadly defined as automatic program induction, using a supervised learning methodology. In other words, the goal of GP is to evolve syntactic expressions that perform some form of computation, attempting to find the relationship between a set of independent variables (inputs) and dependent variables (outputs). Most EAs, such as genetic algorithms (GAs), focus on function optimization. In the case of modeling, while a GA or other similar EAs can be used to optimize model parameters, GP is intended to automatically derive the structure or syntax of the model. Moreover, unlike other machine learning paradigms such as neural networks,
GP uses the EA approach to automatically define the shape and size of the model with a very limited amount of user input.

Like most other EAs, the GP process proceeds as follows: (1) a random generation of a set (population) of candidate solutions (individuals); (2) the use of a domain-specific objective or cost (fitness) function that grades the quality of each solution; (3) a stochastic selection mechanism to probabilistically choose individuals (parents) that will be used to construct a new set of solutions (offspring); (4) stochastic search (genetic) operators called mutation and crossover, that take selected parents as inputs and produce offspring as output, such that useful traits are inherited with the goal of progressively generating better solutions; (5) this process is iteratively repeated (each iteration is called a generation) until a stopping criterion is met, such as a maximum computational effort which can be measured using the number of generations or total evaluations of the fitness function.

This process is very similar for all EAs, with GP having some specific characteristics. The most common application of GP is to solve what are known as symbolic regression problems, where a model that best fits a dataset is sought. Unlike other forms of regression, the form of the model is not defined a priori, as is done in linear regression for example. The goal of GP is to search for the symbolic expression $K^* : \mathbb{R}^p \rightarrow \mathbb{R}$ that best fits a particular training set $T = \{(x_1, y_1), \ldots, (x_n, y_n)\}$ of $n$ input/output pairs with $x_i \in \mathbb{R}^p$ and $y_i \in \mathbb{R}$, stated as

$$\begin{align*}
(K^*, \theta^*) &= \arg\min_{K \in \mathbb{G}; \theta \in \mathbb{R}^m} f(K(x_i, \theta), y_i) \quad \text{with} \quad i = 1, \ldots, n,
\end{align*}$$

where $\mathbb{G}$ is the search space that contains all possible syntactic expressions, $f$ is the fitness function which is based on the difference between a program’s output $K(x_i, \theta)$ and the desired output $y_i$, and $\theta$ is a particular parameterization of the symbolic expression $K$, assuming in real-valued parameters.

As stated above, the representation used for individuals should be able to encode programs, mathematical formulas or other syntactic expressions. The most common representation is to use tree structures [6,7], but other representations are possible [8]. When using a tree representation individuals are evaluated from the bottom up, such that inputs appear on the leaves of the trees and the output is produced at the root node. Leaves can contain the problem’s independent variables, constants and 0-arity functions. All of these are referred to as terminals and conform the Terminal set $T$. Internal nodes are taken from a Function set $F$ that contains the primitive operations used to construct the function or model, with simple functions such as arithmetic and trigonometric operations, or even more complex operations such as image filters [10]. In general, the sets $T$ and $F$ are chosen based on the problem domain, and together define the primitive set $P = (T \cup F)$. If we set a maximum allowable depth for the evolved trees, then set $P$ uniquely defines the space of all possible program syntaxes $\mathbb{G}$, that is also the search space for the GP algorithm.

The search operators need to be applicable $\forall K \in \mathbb{G}$, and should be complete; i.e., they can generate any $K \in \mathbb{G}$. The search operators are functions that take one or more program trees as input, and produce one or more programs as output. In standard tree-based GP, the operators are called subtree mutation $SM$ and subtree crossover $SX$, where $SM(K) : \mathbb{G} \rightarrow \mathbb{G}$ and $SM(K, K') : \mathbb{G} \times \mathbb{G} \rightarrow \mathbb{G}$. Subtree mutation takes a single parent as input and creates a single offspring as output. $SM$ entails the selection and deletion of a random subtree from the parent that is then replaced by a randomly generated tree. Subtree crossover takes two trees as input and produces two trees as output. $SX$ combines two parents by swapping subtrees between them, each subtree being randomly chosen in each parent.

Finally, fitness evaluation is usually expressed as an error function between the desired outputs and the actual output of each individual solution (program or model) evaluated over the entire dataset used for training, such as the Mean Absolute Error (MAE) given by

$$f(K) = \frac{1}{n} \sum_{i=1}^{n} |K(x_i, \theta) - y_i|, \quad (2)$$

where $n$ is the number of samples in data. Fig. 5 provides a graphical depiction of the basic GP process.
3.1. Applying GP to model real-world systems

Providing an exhaustive review of how GP has been used to solve real-world modeling problems is beyond the scope of this manuscript. Our goal is more modest, to provide some recent examples of successful applications of GP to solve complex modeling problems in real-world domains. For example, in [21] the authors use a hybrid GP and GA algorithm to derive models based on systems of ordinary differential equations and higher-order differential equations, using several real-world applications to illustrate their approach which is based on tree-based GP. Recently, GP was applied to model the consumption of natural gas in a real-world chemical plant [22], outperforming the specialist prediction in minimizing costs to the plant. Another related example to the work presented here is [23], where GP is used to model the exhaust gas temperature of a commercial jet engine, and [24] where GP is used to aid in the practical modeling process of Stirling engines. The prediction of fuel consumption for a gasoline engine was addressed with GP in [25], achieving very high correlation with the real behavior of the engine and a low predictive error. Modeling the start-up process of an aero-engine is studied in [26], using a specialized GP system to generate linear (in parameter) models, and comparing with support vector machines shows that GP models outperform the standard technique and produce smaller and more parsimonious solutions. Similarly, [27] presents an approach based on a stack-based GP system for the automatic identification of the dynamics of a wind turbine. The approach posed a multi-objective search, deriving models that are as accurate as those produced with competing techniques and are more parsimonious and easier to analyze and interpret. Research has also shown that GP models can outperform standard regression techniques in predicting the material characteristics of specialized alloys [28], and focus on the fact that since GP is a population-based and stochastic method it can generate a variety of different models that can be considered for further analysis.

3.2. Genetic programming with local search

In standard GP, only the symbolic expression $K$ in problem (1) is evolved by the search process, while the underlying parameterization $\theta$ is not considered. Recent works have shown that concurrently optimizing both the structure and the parameterization of the evolved models can speed up convergence and improve performance. In particular, we adopt our proposal in [17,18], which has been applied to both symbolic regression and classification problems. The approach can be summarized as follows.

First, as suggested in [29], for each individual $K$ in the population we add a small linear uppertree above the root node, such that

$$K' = \theta_1 + \theta_2(K)$$

where $K'$ represents the new program, while $\theta_1$ and $\theta_2$ are the first two parameters from $\theta$.

Second, for all the other nodes $n_k$ in the tree $K$ we add a weight coefficient $\theta_{1k}$, such that each node is now defined by

$$n'_k = \theta_{1k}n_k$$

where $n'_k$ is the new modified node, $k \in \{3, \ldots, r\}$, $r = |K|$ which is the total number of nodes (size) of $K$. Notice that each node has a unique parameter that can be modified to help meet the overall optimization criteria of the non-linear expression. At the beginning of the GP run, each parameter is initialized as $\theta_1 = 1$. During the GP syntax search, subtrees belonging to different individuals are swapped, added or removed (following the standard crossover/mutation rules) together with their corresponding parameters, without affecting their values. This follows a memetic search process with Lamarckian inheritance [17,18], where its main goal is to preserve the information acquired during the local search process and pass it to the following generation. Therefore, we consider each tree as a non-linear expression and the local search operator will search for the best fit parameters of the model $K'$. The problem can be solved using a variety of techniques, but following [17,18] we employ a trust region algorithm.

Finally, it is important to consider that the local search optimizer can substantially increase the underlying computational cost of the search, particularly when individual trees are very large. While applying the local search strategy to all trees might produce good results [17,18], it is preferable to reduce to a minimum the amount of trees to which it is applied. Therefore, we use the heuristic proposed in [30], where the local search is applied stochastically based on a probability $p(s)$ determined by the tree size $s$ (number of nodes) and the average size of the population $\bar{s}$, based on

$$p(s) = \begin{cases} 1.5 - \frac{s}{\bar{s}} & \text{if } 0.5\bar{s} \leq s \leq 1.5\bar{s} \\ 1 & \text{if } s < 0.5\bar{s} \\ 0 & \text{otherwise.} \end{cases}$$

In this way, smaller trees are more likely to be optimized than larger trees, which also reduces the computational effort and improves the convergence of the trust region optimizer by keeping the parameter vectors relatively small. We will refer to this version of GP as GP-LS.

3.3. Geometric semantic genetic programming

In the last few years, a rising topic in the field of GP has been the use of what are referred to as semantic methods [15,16,31,32]. Semantics in GP can be loosely defined as a description of what a program actually does, instead of a description of its syntax. Specifically, for supervised learning problems with a training set $\mathbb{T}$ with $n$ input/output pairs $(x_i, y_i)$, GP literature usually defines the semantics of a program $K$ as the vector of program outputs $s = (K(x_1), \ldots, K(x_n))$ [15]. From this perspective, the vector of ideal outputs $y = (y_1, \ldots, y_n)$ can be referred to as the target semantics and the fitness of a program $K$ can be quantified by a distance between vectors $y$ and $s$.

In particular, let us consider regression problems where $x \in \mathbb{R}^p$, and $s, y \in \mathbb{R}^n$. Then, the aim of GP is to find a program syntax that produces the target semantics of the problem. However, the search operators of standard GP operate at the syntax level with an indirect effect on the semantics of a program; i.e., standard GP operators cannot control or even constrain what will be the semantics of the offspring based on the semantics of the parents. This makes the GP search process inefficient, in the sense that while fitness is uniquely and directly defined by semantics the mapping between syntax and fitness is much more complex and difficult to predict. Therefore, Moraglio et al. [15] proposed important new search operators that operate at the level of syntax but have a predictable and bounded behavior on program semantics. This is achieved by imposing geometric constraints within semantic space. These operators are referred to as geometric semantic operators (GSOs) and a GP algorithm that uses them is called geometric semantic GP or GSGP. These operators are defined as follows.

**Geometric semantic crossover (GSC).** Given two parent functions $K_1, K_2 : \mathbb{R}^n \rightarrow \mathbb{R}$, the geometric semantic crossover returns the real function

$$K_{so} = (K_1 \cdot K_2) + ((1 - K_1) \cdot K_2),$$

where $K_2$ is a random function and $K_1$ is a real function with domain $\mathbb{R}$ and codomain $[0,1]$.

**Geometric semantic mutation (GSM).** Given a parent function $K : \mathbb{R}^n \rightarrow \mathbb{R}$, the geometric semantic mutation with mutation step $ms$ returns the real function

$$K_{sm} = K + ms \cdot (K_{r1} - K_{r2}),$$

where $K_{r1}$ and $K_{r2}$ are random real functions.
Using these operators the semantics of the offspring are completely defined by the semantics of the parents. The semantics of an offspring $K_G$ produced by GSC will lie on the segment between the semantics of both parents. On the other hand, GSM defines a mutation such that the semantics of the offspring $K_M$ lies within the $n$-sphere of radius $m$, where $m$ is the number of individuals for $g$ generations is $O(n^g)$ [33], while the cost of evaluating a new and unseen instance is $O(g)$ [33]. This is the implementation used in this work.

### 3.3.1. GSGP with local search

While GSGP has been shown to produce strong results [16,31,32], its original formulation has several disadvantages. It is easy to see that GSC will not be useful when the semantics of the parents do not surround the target semantics. Similarly, GSM can sometimes produce offspring that have a worse fitness than the parent, which is unnecessary since the target semantics is known. Moreover, the offspring produced by GSOs will always be larger than the parents, meaning that program growth cannot be eliminated.

This work follows our proposal in [19], where a local search approach is integrated into GSM, an operator we will refer to as GSM-LS. This operator exploits the fact that the geometric mutation defines a linear combination of the parent program $K$ with two random programs $K_1$ and $K_2$, expressed as

$$K_M = \alpha_0 + \alpha_1 \cdot K + \alpha_2 \cdot (K_{R1} - K_{R2})$$

where $\alpha_i \in \mathbb{R}$ notice that $\alpha_2$ replaces the mutation step parameter $m$ of GSM. GSM-LS poses a basic multivariate linear regression problem, which could be solved, for example, by Ordinary Least Square (OLS) regression. However, in this case we have $n$ linear equations given by the number of fitness cases, and only three unknowns ($\alpha_i$). This gives an overdetermined multivariate linear fitting problem, which can be solved through singular value decomposition (SVD).\(^1\) In this way the GSM-LS will be guaranteed to produce the best linear fit based on the target semantics of the program. Hereafter, we will refer to a GSGP with GSM-LS mutation as GSGP-LS. This version of GSGP has performed very well in a variety of domains [11–13], in many cases outperforming standard GP, GSGP and several other regression techniques.

### 4. Experiments

This work focuses on applying GP to model a GT unit using a data-driven approach following a supervised learning formulation. Therefore, we will use the datasets described in Section 2 to pose two symbolic regression problems and solve them with the variants of $G_F$ described in Section 3. Hereafter, we will refer to the problem of modeling the fuel flow of the GT unit as the Fuel problem, and the problem of modeling the exhaust gas temperature as the Temperature problem.

Five different algorithms are compared based on the GP systems described in the previous section; these are:

1. The standard GSGP algorithm as described in [15] and implemented in [34];
2. The GSGP-LS algorithm proposed in [19] and described in the previous section;
3. An algorithm we refer to as HYBRID, that combines GSGP and GSGP-LS, by applying GSGP-LS for the first generations and then switches to GSGP, as suggested in [19];
4. HYBRID-LIN is the same as HYBRID, but the training data is preprocessed using the linear scaling approach of [35];
5. GP-LS is the GP algorithm with a LS operator integrated within the evolutionary process.

All GSGP variants are implemented using the GSGP-C++ library [34], while GP-LS is implemented using Matlab and the GPlab toolbox [36].

#### 4.1. Experimental setup

All algorithms were used to solve both the Flow and Temperature problems, performing 30 independent runs of each. In each case, the problem data was split into a training and testing set, where the former contains 70% of the data samples and the latter the remaining 30%. The datasets were randomly partitioned before each run and, while training performance will be reported, the critical result will be the prediction accuracy of the evolved models over the test data. Table 3 summarizes the parameters for all tested algorithms. For GSGP variants mutation and crossover rates have been chosen after a preliminary tuning phase that has been performed following the indication reported in [37]. GP-LS parameters were set based on the values reported in [17,18], with only minimal manual tuning. Values for other parameters were selected based on the results of the experiments. For the stopping criterion all GSGP variants were executed for a maximum of 300 generations. On the other hand, the stopping criterion for GP-LS is 100 generations. Notice also that the function sets in both cases are different, since we follow the best reported results for each algorithm. For HYBRID and HYBRID-LIN $l=10$ as in [19,11]. Finally, in GSGP variants, when the standard GSM mutation is used (instead of GSM-LS) the mutation step is randomly set at each mutation event in the range $[0, 1]$. It is important to mention that in all cases fitness was determined using the MAE between the real output and the estimated output of each problem when evaluated over the training set. Moreover, all subsequent comparisons performance will be measured using the MAE, computed on the training or testing data.

#### 4.2. Results and analysis

Fig. 6 presents a summary of all results as boxplots that show the median, first and second quartiles, as well as the maximum and minimum performance over all thirty runs for both problems.

---

\(^1\) In this work, the GNU Scientific Library is used [http://www.gnu.org/software/gsl/](http://www.gnu.org/software/gsl/).
Fig. 6. Experimental results showing the training (a and b) and test (c and d) performance of all algorithms on each problem, flow (a and c) and temperature (b and d).

Table 4: Summary of median performance by each algorithm on each problem.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Flow train</th>
<th>Flow test</th>
<th>Temperature train</th>
<th>Temperature test</th>
</tr>
</thead>
<tbody>
<tr>
<td>GSGP</td>
<td>0.0961</td>
<td>0.0952</td>
<td>293.75</td>
<td>279.36</td>
</tr>
<tr>
<td>GSGP-LS</td>
<td>0.0957</td>
<td>0.0952</td>
<td>85.18</td>
<td>108.23</td>
</tr>
<tr>
<td>HYBRID</td>
<td>0.0953</td>
<td>0.0945</td>
<td>94.41</td>
<td>113.41</td>
</tr>
<tr>
<td>HYBRID-LIN</td>
<td>0.0963</td>
<td>0.0980</td>
<td>62.72</td>
<td>77.97</td>
</tr>
<tr>
<td>GP-LS</td>
<td>0.0909</td>
<td>0.1006</td>
<td>12.94</td>
<td>12.92</td>
</tr>
</tbody>
</table>

Fig. 6(a) shows the training error on the Flow problem and Fig. 6(b) shows the same for the Temperature problem. Conversely, Fig. 6(c) shows the test error on the Flow problem while Fig. 6(d) shows the same for the Temperature problem.

Moreover, Table 4 summarizes these results numerically, reporting the median train and test performance for each algorithm on each problem, showing in bold the best (lowest) results. What immediately stands out from these results is that GP-LS outperforms all other algorithms in all cases except one, testing performance on the Flow problem. Also notice that while performance on the Flow problem is very similar for all algorithms, on the Temperature problem performance varies quite substantially, with GSGP performing the worst and GP-LS performing the best. Another relevant observation is related to the variance in performance of each method over all runs. First, on the Flow problem, at one extreme HYBRID-LIN exhibits very stable behavior in all runs, suggesting that the linear scaling approach increases the robustness of GP search. The other extreme is GP-LS, which shows the largest variance on this problem, with several runs achieving substantially smaller error values than the other methods. Second, on the Temperature problem, all methods exhibit comparatively less variance, even GP-LS behaves quite consistently over all runs.

Table 5: Summary p-values of the pairwise Friedman test with Benjamini–Hochberg correction for the test performance on the Flow problem; an asterisk (*) indicates that the null hypothesis can be rejected at the \( \alpha = 0.05 \) significance level.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>GSGP</th>
<th>GSGP-LS</th>
<th>HYBRID</th>
<th>HYBRID-LIN</th>
<th>GP-LS</th>
</tr>
</thead>
<tbody>
<tr>
<td>GSGP</td>
<td>–</td>
<td>0.7150</td>
<td>0.0000*</td>
<td>0.1358</td>
<td>0.715</td>
</tr>
<tr>
<td>GSGP-LS</td>
<td>–</td>
<td>–</td>
<td>0.0000*</td>
<td>0.1358</td>
<td>0.715</td>
</tr>
<tr>
<td>HYBRID</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>0.1358</td>
<td>0.715</td>
</tr>
<tr>
<td>HYBRID-LIN</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>0.715</td>
</tr>
<tr>
<td>GP-LS</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
</tbody>
</table>

Table 6: Summary p-values of the pairwise Friedman test with Benjamini–Hochberg correction for the test performance on the Temperature problem; an asterisk (*) indicates that the null hypothesis can be rejected at the \( \alpha = 0.05 \) significance level.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>GSGP</th>
<th>GSGP-LS</th>
<th>HYBRID</th>
<th>HYBRID-LIN</th>
<th>GP-LS</th>
</tr>
</thead>
<tbody>
<tr>
<td>GSGP</td>
<td>–</td>
<td>0.0000*</td>
<td>0.0000*</td>
<td>0.0000*</td>
<td>0.0000*</td>
</tr>
<tr>
<td>GSGP-LS</td>
<td>–</td>
<td>0.4652</td>
<td>0.0000*</td>
<td>0.0000*</td>
<td>0.0000*</td>
</tr>
<tr>
<td>HYBRID</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>0.0000*</td>
<td>–</td>
</tr>
<tr>
<td>HYBRID-LIN</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>GP-LS</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
</tbody>
</table>

To test the significance of these results, we use the Friedman test to perform all pairwise comparisons of the methods, focusing specifically on test performance for both problems. The null hypothesis in each pairwise comparison is that the groups have equal medians, and the p-values are corrected using the Benjamini–Hochberg algorithm. The resulting p-values are given in Tables 5 and 6, the former for the test performance on the Flow problem and the latter for the test performance on the Temperature problem. In both tables an asterisk (*) indicates that the null hypothesis can be rejected at the \( \alpha = 0.05 \) significance level.
This statistical comparison suggests the following. On the Flow problem, basically all methods exhibit statistically similar results, since the null hypothesis was not rejected in all pairwise comparisons except two. In particular, the test suggests that a statistically significant difference exists between HYBRID and GSGP, and HYBRID and GSGP-LS. While the boxplots of all three methods appear quite similar (see Fig. 6), under further inspection the distribution of GSGP and GSGP-LS are actually bi-modal, which explains the rejection of the null hypothesis. On the other hand, on the Temperature problem it is clear that GP-LS exhibits performance that is significantly different with respect to all other methods, confirming its superiority on this problem. Taken as a whole, we can say that GP-LS is the best method, since its median performance is not statistically different with respect to other methods on the Flow problem, while achieving the best performance on the Temperature problem.

Finally, it is important to comment on the size of the evolved models, which for GP-based systems is measured by the number of nodes of the best solution found. In particular, GSGP is known to produce extremely large models, the size of the models grows exponentially with the number of generation [15]. This means that GSGP solutions are basically black boxes. On the other hand, one of the goals of GP is to produce models that can be interpreted or easily implemented in real-world systems. In this case, GP-LS produces vastly more compact solutions. The median size of the evolved models are 33 and 36 nodes for the Flow and Temperature problems, respectively.

Fig. 7 presents a graphical depiction of the best models evolved by GSGP-LS on each problem, showing 3D point clouds of the real data compared with the predicted values of the GSGP-LS models over the entire dataset. The GP-LS models were chosen because they achieved the minimum error (see the boxplots of Fig. 6) and because the models are the most compact, which is desirable in a real-world deployment of the models. Notice that in both cases the models are able to closely approximate the real behavior of the GT unit, accurately predicting the behavior of both variables of interest: fuel flow and exhaust temperature.

The actual best symbolic model obtained by the GP-LS technique for the Flow problem is given by

$$K_F(tf, ts) = (\theta_1 \times (\theta_2(tf)))$$

$$+ (((\theta_3)(\log((\theta_4(\log((\theta_5((\sin((\theta_6(\sin((\theta_7(tf))))))))))))(\theta_11(tf))))))$$

$$/ (\theta_12((\log((\theta_13(ts)))))) + (\theta_14(\log((\theta_15(\log((\theta_16(ts))))))))$$

$$+ (\theta_17((\log((\theta_18(\log((\theta_19(\log((\theta_20(\log((\theta_21(ts))))))))))))))$$

$$+ (\theta_22((\log((\theta_23(\log((\theta_24(\log((\theta_25(ts)))))))))))/(\theta_26(t))))$$

$$+ (\theta_27((\exp((\theta_28(\log((\theta_29(ts)))) + (\theta_30(tf)))/(\theta_31(ts))))))))))))))))$$

$$- (\theta_32(tf))) + \theta_33)$$

(7)

where $K_F(tf, ts)$ is the approximate fuel flow model, $tf$ represents the reference fuel flow, $ts$ the turbine speed, and the model parameters are: $\theta_1 = 0.1377, \theta_2 = -2.7002, \theta_3 = 0.5654, \theta_4 = -17.8777, \theta_5 = 1.0151, \theta_6 = -2.0230, \theta_7 = -0.0188, \theta_8 = 2.1919, \theta_9 = 4.8047, \theta_{10} = 0.0550, \theta_11 = -8.2834, \theta_12 = -0.1644, \theta_13 = -0.255, \theta_14 = -7.4487, \theta_15 = 4.7472, \theta_16 = 0.4072, \theta_17 = -0.1766, \theta_18 = 4.8017, \theta_19 = 0.5157, \theta_20 = 21.6877, \theta_21 = 0.0480, \theta_22 = 14.5261, \theta_23 = 1.4830, \theta_24 = 3.1747, \theta_25 = 0.0689, \theta_26 = 0.0085, \theta_27 = 2.1211, \theta_28 = 758.9901, \theta_29 = -0.0101, \theta_30 = 1.6733, \theta_31 = 0.3230, \theta_32 = -9.9588, \theta_33 = -0.0007.$

On the other hand, the model corresponding to the Temperature variant is given by

$$K_T(ff, cp) = (\theta_1 + (\theta_2(\log(((\theta_3(\tan((\theta_4(\log((\theta_5(\log((\theta_6(ff))))))))))))))))$$

$$\times (\theta_14(\exp((\theta_15(ff)))))) \times (\theta_6(\exp((\theta_7(cp))))))$$

$$/ (\theta_12((\log((\theta_13(ts)))))) + (\theta_14(\log((\theta_15(\log((\theta_16(ts))))))))$$

$$+ (\theta_17((\log((\theta_18(\log((\theta_19(\log((\theta_20(\log((\theta_21(ts))))))))))))))$$

$$+ (\theta_22((\log((\theta_23(\log((\theta_24(\log((\theta_25(ts)))))))))))/(\theta_26(t))))$$

$$+ (\theta_27((\exp((\theta_28(\log((\theta_29(ts)))) + (\theta_30(tf)))/(\theta_31(ts))))))))))))$$

$$- (\theta_32(tf))) + \theta_33)$$

(8)

where $K_T(ff, cp)$ is the approximate temperature model, $cp$ represents the compressor discharge, $ff$ the fuel flow, and the model parameters are: $\theta_1 = 5.7509, \theta_2 = 5.2602, \theta_3 = 16.2779, \theta_4 = 9.2538, \theta_5 = 1.0549, \theta_6 = -4.4081, \theta_7 = -0.2369, \theta_8 = 0.6149, \theta_9 = -0.3637, \theta_{10} = -15.6936, \theta_{11} = 2.6504, \theta_{12} = 99.5816, \theta_{13} = -0.0022, \theta_{14} = 9.3046, \theta_{15} = 51.3792, \theta_{16} = 0.7228, \theta_{17} = -179.6383, \theta_{18} = -7.6820, \theta_{19} = 0.0430, \theta_{20} = 1.1433, \theta_{21} = 10.0145, \theta_{22} = -77.0285, \theta_{23} = 562.4549, \theta_{24} = -0.3169, \theta_{25} = 562.6132, \theta_{26} = -2.1891, \theta_{27} = -6.0626, \theta_{28} = 0.3125, \theta_{29} = 3.2894, \theta_{30} = 0.4170, \theta_{31} = 12.0751, \theta_{32} = 254.8610, \theta_{33} = 0.7083, \theta_{34} = -0.5231, \theta_{35} = -7.5917, \theta_{36} = 310.6381, \theta_{37} = 3.5085, \theta_{38} = -2.6837.
4.3. Comparison with other machine learning methods

In order to get a better contextualization of the previous results, we compare the best GP method (GP-LS) with other common techniques for data-driven modeling taken from machine learning literature. In particular, the following methods from the Weka open source library [38] have been considered: (1) linear regression (LR); (2) isotonic regression (IR); (3) square regression (SR); (4) multilayer perceptron (MLP); and (5) support vector machines (SVM) with polynomial kernel. In all cases, we use the default configurations for each method except for MLP, in which we performed a parameter sweep of the decay learning rate and the optimal number of hidden nodes. For both problems the learning rate was left in the default value, while the number of neurons in the hidden layer was set to 28 for the Flow problem and to 8 for the Temperature problem performing a greedy hill climber search. Moreover, we also performed a comparison with the Fast Function Extraction algorithm (FFX) [39] and Multivariate Adaptive Regression Splines (MARS) [40]. Generally speaking, FFX and MARS build a linear combination of basis functions to construct the model that best describes the training data of a supervised learning problem. On the one hand, FFX was developed as an alternative to the evolutionary search of standard GP but with the same general goals, while MARS is a well established method that continues to be widely used in difficult real-world domains [41–43]. Both of these techniques are similar to GP in the sense that their goal is to produce a new symbolic expression that best fits the data, so we will also compare these methods with GP-LS based on the size of the models each method produces. For both MARS and FFX we used the freely available toolboxes provided by the authors using their default configuration.

Fig. 8 shows a boxplot comparison between all methods based on the test performance on both problems. Table 7 compares the test performance of all of the above methods with GP-LS on both problems. Moreover, we apply the same statistical pairwise test as before and report the results in Table 8 for the Flow problem and in Table 9 for the Temperature problem. As before, there is little

---

Table 7
Summary of median performance by each algorithm on each problem, comparing GP-LS with other machine learning methods.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Flow test</th>
<th>Temperature test</th>
</tr>
</thead>
<tbody>
<tr>
<td>LR</td>
<td>0.1095</td>
<td>224.96</td>
</tr>
<tr>
<td>IR</td>
<td>0.1122</td>
<td>118.16</td>
</tr>
<tr>
<td>SR</td>
<td>0.0947</td>
<td>223.80</td>
</tr>
<tr>
<td>MLP</td>
<td>0.1493</td>
<td>15.23</td>
</tr>
<tr>
<td>SVM</td>
<td>0.0948</td>
<td>80.98</td>
</tr>
<tr>
<td>FFX</td>
<td>0.0992</td>
<td>15.03</td>
</tr>
<tr>
<td>MARS</td>
<td>0.1111</td>
<td>14.81</td>
</tr>
<tr>
<td>GP-LS</td>
<td>0.1006</td>
<td>12.92</td>
</tr>
</tbody>
</table>

---

3 http://trent.st/ffx/.
difference between the methods on the Flow problem, with the statistical tests suggesting that the best performance is achieved by SR, SVM, FFX and GP-LS. On the other hand, on the Temperature problem two methods stand out: MARS and GP-LS achieve the lowest median performance and the statistical test suggesting that there is no significant difference between their median performance. However, if we analyze the boxplots of Fig. 8 we can see that the lowest error was produced by GP-LS.

Finally, it is useful to compare the size of the models produced by FFX, MARS and GP-LS, since these methods are aimed at producing new symbolic expressions that describe the problem data. In this task, it is always preferable to generate the smallest or most parsimonious solutions, for ease of interpretation. Fig. 9 presents boxplots that compare the size of the solutions produced by each method over all runs. For a fair comparison, we express the models produced by MARS and FFX as syntax trees, and define the size of the models as the total number of nodes. On the Flow problem, we can see that the smallest solutions are produced by GP-LS and MARS, but it is important to remember that GP-LS outperforms MARS in terms of test error. Moreover, while FFX achieved similar performance to GP-LS in terms of test error, the models it generates are one order of magnitude larger. Similarly, on the Temperature problem the smallest solutions are generated by GP-LS. This is particularly important since it outperforms all other methods in terms of test MAE, and while MARS achieves similar performance it is evident that it does so at the cost of generating less parsimonious solutions.

5. Conclusions and future work

This paper addresses the problem of modeling key aspects of a GT power system, namely the fuel flow of the system and the temperature of the exhaust gas. Both variables are critical to the proper functioning and performance of the GT power generator. In particular, this paper describes a data-driven approach to generate the models using GP, a paradigm of evolutionary computation for automatic program induction. Recent state-of-the-art variants of GP are tested. These variants integrate a powerful local search mechanism (GP-LS) or are based on geometric semantic concepts (GSGP). To pose the supervised learning problem, real data taken from a power plant in Mexico is used, resulting in a difficult real-world task. Comprehensive experimental validation clearly shows that GP-LS outperforms all other GP variants on both problems, with clear statistical significance in the detected differences among the methods. In particular, we would emphasize that the largest improvement, relative to the other methods, was achieved by GP-LS on the temperature prediction problem. This is important to mention since the temperature can cause the most harm to a GT system if it fluctuates outside normal operating ranges, making accurate predictive models for this variable a desirable tool for real-world system monitoring.

While recent literature has emphasized the power of geometric semantic versions of GP, these results show that standard GP algorithms can still be used to address difficult real-world problems with strong results, while also evolving much smaller solutions. This is achievable by integrating local search strategies into the search process, allowing standard GP operators to focus on evolving the structure of the solution and allowing the fine-tuning of these models to be performed by standard numerical methods. These results are consistent with what we showed in previous works using community benchmarks.

Future work will be devoted towards integrating the evolved models into an online monitoring system, to be used by engineers and researchers working at the power plant. The goal is to develop an automatic fault detection system, that can allow for easy monitoring and control to avoid possible damage or losses in power service delivered by the plant.

Acknowledgements

This research was partially supported by CONACYT Basic Science Research Project No. 178323, TecnNM (México) Research Project 5621.15-P, as well as by FP7-Marie Curie-IRSES 2013 European Commission program with project ACoBSEC with contract No. 612689. Third and fourth authors are supported by CONACYT doctoral scholarships, Nos. 294213 and 302526, respectively. The authors also thank Ing. Sergio Ruiz González (Superintendente General) and Ing. Genero Mena Ramírez (Supervisor de Instrumentación y Control) from the Central Chankanaab, CFE, for their support in the development of this work.

Table 8
Summary p-values of the pairwise Friedman test with Benjamini–Hochberg correction for the test performance on the Flow problem; an asterisk (*) indicates that the null hypothesis can be rejected at the α = 0.05 significance level.

<table>
<thead>
<tr>
<th></th>
<th>LR</th>
<th>IR</th>
<th>SR</th>
<th>MLP</th>
<th>SVM</th>
<th>FFX</th>
<th>MARS</th>
<th>GP-LS</th>
</tr>
</thead>
<tbody>
<tr>
<td>LR</td>
<td></td>
<td>0.0068*</td>
<td>0.0000*</td>
<td>0.0000*</td>
<td>0.0000*</td>
<td>0.0000*</td>
<td>0.0129</td>
<td>0.0035*</td>
</tr>
<tr>
<td>IR</td>
<td></td>
<td></td>
<td>0.0000*</td>
<td>0.0000*</td>
<td>0.0000*</td>
<td>0.0000*</td>
<td>0.5010</td>
<td>0.0001*</td>
</tr>
<tr>
<td>SR</td>
<td></td>
<td></td>
<td></td>
<td>0.0000*</td>
<td>0.1104</td>
<td>0.00646</td>
<td>0.0000*</td>
<td>1.0000</td>
</tr>
<tr>
<td>MLP</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0.0000*</td>
<td>0.0000*</td>
<td>0.0004*</td>
<td>0.0000*</td>
</tr>
<tr>
<td>SVM</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0.0015*</td>
<td>0.0000*</td>
<td>1.0000</td>
</tr>
<tr>
<td>FFX</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0.0000*</td>
<td>0.3061</td>
</tr>
<tr>
<td>MARS</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0.0000*</td>
</tr>
<tr>
<td>GP-LS</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 9
Summary p-values of the pairwise Friedman test with Benjamini–Hochberg correction for the test performance on the Flow problem; an asterisk (*) indicates that the null hypothesis can be rejected at the α = 0.05 significance level.

<table>
<thead>
<tr>
<th></th>
<th>LR</th>
<th>IR</th>
<th>SR</th>
<th>MLP</th>
<th>SVM</th>
<th>FFX</th>
<th>MARS</th>
<th>GP-LS</th>
</tr>
</thead>
<tbody>
<tr>
<td>LR</td>
<td></td>
<td>0.0000*</td>
<td>0.7700</td>
<td>0.0000*</td>
<td>0.0000*</td>
<td>0.0000*</td>
<td>0.0000*</td>
<td>0.0000*</td>
</tr>
<tr>
<td>IR</td>
<td></td>
<td></td>
<td>0.0000*</td>
<td>0.0000*</td>
<td>0.0000*</td>
<td>0.0000*</td>
<td>0.0000*</td>
<td>0.0000*</td>
</tr>
<tr>
<td>SR</td>
<td></td>
<td></td>
<td></td>
<td>0.0000*</td>
<td>0.0000*</td>
<td>0.0000*</td>
<td>0.0000*</td>
<td>0.0000*</td>
</tr>
<tr>
<td>MLP</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0.0000*</td>
<td>1.0000</td>
<td>1.0000</td>
<td>0.0346*</td>
</tr>
<tr>
<td>SVM</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0.0000*</td>
<td>0.0000*</td>
<td>0.0000*</td>
</tr>
<tr>
<td>FFX</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0.5210</td>
<td>0.0346*</td>
</tr>
<tr>
<td>MARS</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0.0792*</td>
</tr>
<tr>
<td>GP-LS</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
References


