Using Ontologies to Express Prior Knowledge for Genetic Programming

Stefan Prieschl[1(✉)], Dominic Girardi[1], and Gabriel Kronberger[2]

[1] RISC Software GmbH, Johannes Kepler University, Hagenberg, Austria
{stefan.prieschl,dominic.girardi}@risc-software.at
[2] Josef Ressel Centre for Symbolic Regression, University of Applied Sciences Upper Austria, Hagenberg, Austria
gabriel.kronberger@fh-hagenberg.at

Abstract. Ontologies are useful for modeling domains and can be used to capture expert knowledge about a system. Genetic programming can be used to identify statistical relationships or models from data. Combining expert knowledge as well as statistical rules identified solely from data is necessary in application domains where data is scarce and a large body of expert knowledge exists.

We therefore study if the performance of genetic programming can be improved by incorporating prior knowledge from an ontology. In particular, we include prior knowledge as additional features for genetic programming.

The approach is tested with six benchmark data sets where we compare the required computational effort that is necessary to find an acceptable model with and without additional features. The results show that additional features gathered from an ontology improve the performance of tree-based GP. The probability to find acceptable solutions with a fixed computational budget is increased. For noisy data sets we observed the same effect as for the data sets without noise.

Keywords: Supervised learning · Ontologies · Domain knowledge · Genetic programming · Symbolic regression

1 Motivation

In the recent years, research in many domains has developed into a mostly data-driven activity. This requires researchers with knowledge in their own research domain on the one hand and knowledge in data science on the other hand. It is a challenge to bring these two worlds together. In the field of medical informatics, activities around this problem are often labelled as doctor-in-the-loop approach [11]. The goal is to deeply integrate the domain experts into the knowledge discovery process and benefit from their expertise, while acknowledging the fact that these domain experts are neither IT experts nor data scientists. Thus, support from software tools is needed to utilize experts' prior domain knowledge for

Published by Springer Nature Switzerland AG 2019
A. Holzinger et al. (Eds.): CD-MAKE 2019, LNCS 11713, pp. 362–376, 2019.
https://doi.org/10.1007/978-3-030-29726-8_23

modeling. The challenge from a technical point of view is to formally describe their knowledge and to make algorithms that elicit and employ this knowledge. In the literature various examples of approaches to integrate humans in the process can be found. For example Holzinger et al. include humans by gamification into a machine learning process [16].

A concept that is potentially able to cope with this challenge is genetic programming (GP). GP is a method of evolutionary computing where concepts from natural evolution are simulated for the evolution of computer programs that are able to solve a given problem when executed [6,19]. A particular task of GP is symbolic regression. Here GP is used to evolve simple closed-form expressions that fit a given data set. Unlike black-box models such as support vector machines (SVM) or artificial neural networks (ANN), the aim of symbolic regression is to identify models which can be interpreted by humans. In this way, the loop from the expert knowledge to the algorithms back to the domain expert can be closed.

Expert knowledge can be formalized using a domain ontology, which contains structural information about the research data as well as prior domain knowledge about known correlations and causal dependencies between data attributes. A GP algorithm can be launched for a selected data set of interest incorporating prior knowledge as building blocks. The hypothesis of the present paper is that the performance of GP can be improved when it is provided with prior domain knowledge. This improvement can either yield a better regression model given a fixed computational budget or alternatively yield a model of equivalent quality with less computational effort.

2 Related Work

2.1 Genetic Programming

GP is a technique where a genetic algorithm (GA) is used to generate a computer program [19]. Usually, programs are encoded as an expression tree and are evolved over a number of iterations through evolutionary operations: selection, crossover and mutation. In the case of symbolic regression, the GP programs are formulas that replicate a specific relationship from a data set.

A GA is a heuristic method based on Charles Darwin's idea of natural selection [8]. When using a GA one encodes solutions as sets of individuals. Initially, these individuals are randomly generated. Each individual is evaluated by calculating a so-called fitness value. Individuals with higher fitness have a higher probability to be selected for reproduction. This mechanism implements the idea of *survival of the fittest*, which Darwin describes. New individuals are created based on their predecessor using a crossover operation. This leads to increasing result quality of the individuals over iterations. More detailed description of GAs can be found in the literature [1,12,15].

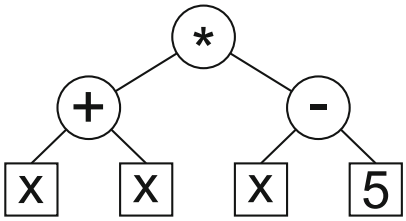

Fig. 1. Example of a symbolic tree representing (x + x) * (x − 5).

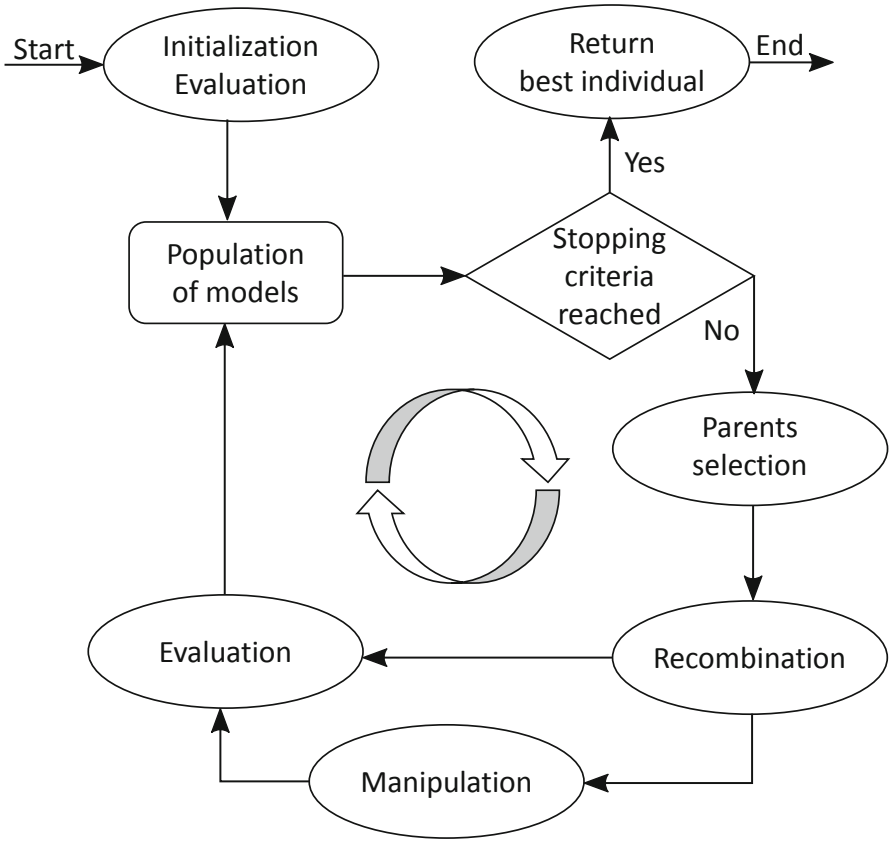

Fig. 2. The process cycle of GP.

Cramer has layed the foundations for GP in 1985 [6]. Koza later popularized and further developed GP [19]. In contrast to classic GAs, a variable-length encoding – most frequently expression trees – are used for GP. Expression trees represent formulas or computer program. Figure 1 shows an example of an expression tree.

Figure 2 shows the process cycle of GP. It starts with the initialization of the population of models by randomly generating a defined number of individuals. Then the main cycle is executed until the stopping criteria are met. The stopping criteria, similarly to classical optimization, can comprise a fixed number of iterations, a defined result quality, or a combination of both.

In every iteration the *selection of parents* takes place, based on the individual's fitness values. To that end, a fitness function is used to evaluate the fitness value of each individual.

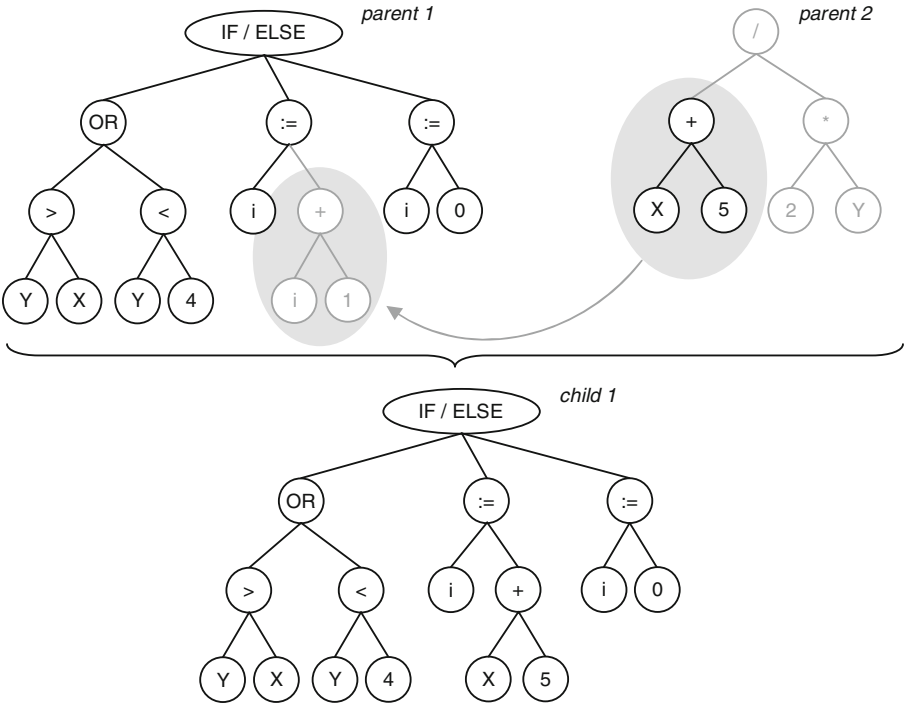

Fig. 3. Example of recombination in GP [1].

The selected individuals are grouped as pairs for the *recombination* (also known as crossover) step. Figure 3 shows an example of such a recombination. For this example a sub-tree is selected for each of the two parent trees. These two sub-trees are swapped and thereby two child trees are created newly. The figure shows how the first of these two child trees is generated out of the two parent trees.

In the *manipulation* step, each new individual is additionally manipulated by mutation with a defined probability. For this mutation there are different possibilities. For example a sub-tree could be removed and replaced by a randomly generated one. Another possibility is to manipulate one of the nodes. This can be implemented by changing the type of a node or changing some of its parameters.

Next, newly generated individuals are evaluated for their fitness value.

The last step creates a new generation of individuals using the previous generation and the newly generated offspring. For example *generational replacement* can be used, where all individuals of the previous generation are discarded and the offspring individuals form the successor generation. Often also *elitism* is used, where one or more individuals with the best fitness values of the old generation are adopted in the new generation.

Nowadays GP gets more and more practicable thanks to the increasing computing power. In short GP offers a powerful tool for machine learning that can be applied on a lot of different problems. Moreover it creates a result, which is easy to understand since symbolic trees are human-readable.

2.2 Ontologies

The notion of ontology has its origin in the ancient Greek philosophy. It was Aristotle who defined ontology as the science of "being *qua* being", which dealt with the structure and nature of things, without going further or even requiring these things to actually exist. In the field of computer science, a number of definitions of the term ontology exist. According to Chandrasekaran et al. *"Ontologies are content theories about the sorts of objects, properties of objects, and relations between objects that are possible in a specified domain of knowledge."* [3] Gruber provide a more general definition: *"An ontology is a specification of a conceptualization."* [13].

Ontologies are used to describe and formally specify domain knowledge. The areas of application are manifold. The most prominent application is representation of complex domain knowledge. Traditionally, biomedical research is a typical application area for these kinds of ontologies, as it can be found in the literature [2,9,22]. Apart from this, ontologies are also used for data integration [7,10] and automated reasoning.

2.3 Genetic Programming in Combination with Ontologies

In the present study, a literature survey in the field of genetic programming in combination with ontologies was conducted, to illustrate the state-of-the-art. In fact, domain knowledge is being used for system identification, but with varying perspectives. For example Ratle and Sebag [24] or Schoenauer and Sebag [26] use domain knowledge for system identification, by using *G3P* (grammar guided GP). G3P [5,28,31] is used to define validity constrains for individuals as context free grammars. However, getting domain knowledge into the GP algorithm itself has not been investigated in this detail to date.

3 Methods

In this contribution we use a very limited set of the capabilities of ontology modelling. In particular, we neglect domain modelling and instead focus only on

the definition of suspected or known relationships between variables – knowledge which can be represented explicitly within an ontology.

To give a simple example, we might know that there is a direct linear relationship between two variables y and x which we could encode as a functional dependency $y \leftarrow \theta x$; whereby we use the parameter θ for the unknown scaling factor and $\leftarrow$ to encode the causal direction of the dependency. Many similar assumed or known functional dependencies can be encoded using the same approach. Further examples for commonly occurring bivariate functional dependencies are[1]:

- Exponential decay of y over x with an unknown rate: $y \leftarrow \exp(\theta x)$
- Logarithmic growth of y over x with an unknown rate: $y \leftarrow \log(\theta x)$
- Oscillation of y over x with an unknown frequency: $y \leftarrow \sin(\theta_1 x + \theta_2)$
- Logistic growth of y over x with an unknown rate and limit: $y \leftarrow \frac{\theta_1}{1+\exp(\theta_2 x)}$

Including knowledge in the GP process is possible in many ways (cf. [30]): extension of the feature set, seeding of the initial generation of GP, definition of syntactic building blocks for evolutionary operators, and extension of the function set.

A straightforward approach is to extend the set of variables and to add a pre-calculated feature for each defined functional dependency. This can be accomplished with minimal effort for any GP implementation without requiring adaptations to the GP implementation itself. A drawback of this approach is that for the calculation of the features it is necessary to know at least approximate values for the parameters θ. For example, for the case that the variable of interest y decays exponentially over x ($y \leftarrow \exp(\theta x)$) we would need to assume a value for the decay rate θ in the calculation of the feature values. Once these values have been calculated, the decay rate is fixed. GP uses the calculated feature exactly the same way as the original variable values. In particular, it is not possible to adjust the decay rate parameter when evolving models using the pre-calculated features. A similar argument can be given for the frequency parameter in a periodic function.

Functional relations expressed in the ontology can be used for the initialization of the GP population in the first generation (seeding). Usually, individuals in the first GP generation are generated randomly. However, when prior knowledge is available in the ontology we can include these expressions as sub-expressions within randomly initialized individuals. The potential benefit is that GP already starts with relevant functional expressions in the genome. As a consequence, these sub-expressions do not have to be discovered through the evolutionary process, theoretically improving the performance of GP. An important

[1] It should be noted that it would be rather easy to provide a graphical interface which is easy to understand for users which are not very familiar with the mathematical notation. For instance it would be possible to provide a graphical representation of the suspected functional dependency between variables where the users only need to choose the function type from a menu. The so-defined functional dependency can then be added to the ontology. This would not affect the underlying implementation of the learning algorithm.

difference to the first approach is that GP is still able to break up, modify and improve sub-expressions which have been included by seeding. This would not be possible with the feature extension. Seeding would require a change of the procedure for initialization of the population of the GP algorithm.

An alternative to seeding of the initial population would be to include prior knowledge as pre-defined syntactical building blocks for expressions produced by GP (cf. [24,26]). This approach requires a so-called grammar-guided GP system [21] which allows the definition of syntactical constraints. Such GP systems produce expressions which conform to a syntax defined via a formal grammar. This facilitates integration of prior knowledge such as known transformations of input variables as well as the definition of the structure of the expression with slots for sub-expressions that can be evolved. For example, such systems would allow to express that

$$y = \exp(g(x_1, x_2)x_3) + f(x_1, x_2) + \epsilon$$

where $g(x_1, x_2)$ and $f(x1, x2)$ are sub-expressions evolved by the GP system. The approach via syntactical building blocks is arguably the most general. Many forms of prior knowledge can be expressed via syntactical constraints. Notably, the simple extension of the feature set as well as seeding of the population are special cases of this approach. However, the flexibility also introduces higher complexity of the GP implementation as well as issues with GP performance which can be hampered by intricate syntactical constraints. Syntactical constraints might even increase the problem of premature convergence as a consequence of the reduced diversity of solutions.

Finally, instead of extending the feature set we could extend the function set of the GP system to include expressions from the ontology. The GP system is allowed to include these functions within evolved expressions using terminals or sub-expressions as arguments. The motivation for this approach is that we could include functions with unknown numeric parameters and allow GP to identify optimal parameters via evolution. For example this would allow us to add a parametric function such as:

$$\text{ExpDecay}_\theta(x) = \exp(-\exp(\theta)x), x \in \{x_1, x_2, x_3\}$$

which only allows features x_1, x_2, x_3 as arguments and has θ initialized randomly and evolved via GP.

For this study, we have chosen to use pre-calculated features because it can be implemented with minimal effort. As discussed above, this approach has the important drawback that parameters have to be approximated by users. However, for our experiments with synthetic benchmark problems we assume that these parameters are known. We leave experiments with syntactical building blocks or parametric functions for future work.

3.1 Experimental Setup

We test the hypothesis that GP performance can be improved trough expert knowledge using computational experiments with simulated data sets. For our

experiments we omit the use of an ontology for simplicity and we directly use the knowledge that could be provided by an ontology. We consider two scenarios where we use the same simulated data sets with and without noise. For the experiments we use tree-based genetic programming which can be considered more a less a de-facto standard variant (SGP). We use PTC2 [20] to initialize random trees with a uniform length distribution between 1 and *max. tree length* nodes. Terminals and function symbols are selected with uniform probabilities from the terminal set and the function set. The algorithm uses generational replacement where the best individual is kept for the next generation (elitism). For crossover events we use sub-tree crossover with 90% probability to select an internal node. For mutation events we conduct one of the following mutation operations with uniform distribution:

- change the function symbol of an internal node
- change the parameter or variable reference of a terminal node
- change the parameters of all terminal nodes
- delete a randomly selected sub-tree
- replace a randomly selected sub-tree with a newly initialized random tree

The mutation rate parameter defines the probability that a newly created individual is manipulated with one the operators above. The fitness of individuals is determined by calculating the coefficient of determination (squared Pearson correlation) of the function output values and the target variable values. The parents for recombination are chosen via tournament selection where the individual with highest R^2 in the group is selected. The full configuration for our GP experiments is shown in Table 1.

We run 60 independent repetitions of the same GP configuration for each of the problem instances described below. In each GP run we invest the same effort of 500,000 evaluated solutions, where we record the quality (R^2) of the best solution in each generation step. This data allow us to answer whether:

1. The probability to solve a given problem is increased when prior knowledge from an ontology is available (for fixed effort).
2. The computational effort to solve a given problem can be reduced (for a given success probability).

For the comparison of algorithm configurations we visualize the empirical distribution of run length (evaluated solutions) until the problem is solved. This method is used for instance for the comparison of multiple optimization algorithms on a large set of benchmark problems in [14]. For the problem instances without noise we set 0.99 as the R^2 threshold for success; for the noisy problems we use the threshold value 0.95. All experiments have been performed with HeuristicLab[2] which provides a grammar guided tree-based GP system [17].

[2] https://dev.heuristiclab.com.

Table 1. Genetic programming parameter settings.

Parameter	Value
Population size	10000
Maximum generations	50
Elites	1
Mutation (one of)	Change function symbol Change terminal symbol Change all parameters Delete a random sub-tree Replace a random sub-tree
Mutation rate	15%
Selector	Tournament with group size 7
Crossover	Sub-tree crossover
Function set	$+, -, *, /, \sin(x), \cos(x), \log(x), \exp(x), x^2, \sqrt{x}$
Max tree depth	17
Max tree length	100
Fitness evaluation	Pearson R^2

3.2 Selection of Benchmarking Data

We selected the following benchmark problem instances, that are shown in Table 2. We generate data for input variables using the ranges defined in Table 3 and calculate the target variable using the expressions.

Some of the instances are recommended in [29]. The *flow psi* function is a function occurring in modelling fluid dynamics and has been taken from [4].[3] All problem instances are possible to solve with our GP settings.

Table 2. The problem instances of our experiments.

Instance	Function
Salustowicz [25,29]	$f(x) = x^3 \exp(-x) \cos(x) \sin(x)(\sin(x)^2 \cos(x) - 1)$
Vladislavleva-3 [27,29]	$f(x_1, x_2) = \exp(-x_1)x_1^3 \cos(x_1) \sin(x_1)(\cos(x_1) \sin(x_1)^2 - 1)(x_2 - 5)$
Vladislavleva-4 [27,29]	$f(x_1, x_2, x_3, x_4, x_5) = \frac{10}{5+\sum_{i=1}^{5}(x_i-3)^2}$
Pagie [23,29]	$f(x, y) = \frac{1}{1+x^{-4}} + \frac{1}{1+y^{-4}}$
Flow psi [4]	$f(x1, x2, x3, x4, x5) = x_1 x_3 \sin(\frac{\pi x_2}{180})(1-(\frac{x_4}{x_3})^2)+x_5 \log(\frac{x_3}{x_4})$
Korns-12 [18,29]	$f(x_0, x_1, x_2, x_3, x_4) = 2 - 2.1 \cos(9.8x_0) \sin(1.3x_4)$

[3] We also ran preliminary experiments with the *Keijzer-6* and *Nguyen-7* functions recommended in [29]. However, these two functions are trivial to solve with the function set used in our experiments. We have therefore not reported the results for these two functions.

Table 3. The input data ranges used for generating the data sets. $E[l, u, s]$ is a grid of evenly spaced points between l and u inclusive with step size s. $U[l, u]$ means uniformly distributed points between l and u (exclusive).

Instance	Input distribution
Salustowicz	$x : E[0, 3.2, 0.05]$
Vladislavleva-3	$x_1, x_2 : E[-0.5, 10.5, 0.1]$
Vladislavleva-4	$x_1, x_2, x_3, x_4, x_5 : U[-0.25, 6.35]$, 4000 samples
Pagie	$x_1, x_2 : E[-5, 5, 0.4]$
Flow psi	$x_1 : U[60, 65], x_2 : U[30, 40], x_3 : U[0.2, 0.5]$, $x_4 : U[0.5, 0.8], x_5 : U[5, 10]$, 132 samples
Korns-12	$x_0, x_1, x_2, x_3, x_4 : U[-50, 50]$, 1320 samples

For the noisy problem instances we modified the data sets by adding a randomly distributed noise term to the target variable y.

$$y_{noise} = y + N(0, 0.2\sqrt{\text{Var}(y)})$$

This means that the maximally achievable R^2 value is limited by the noise level. Table 4 shows the best possible R^2 value for each of the noisy problem instances. We define that a GP run is successful if it reaches at least $0.95R^2$ for the noisy problem instances.

Table 4. The highest possible fitness value (R^2) for each of the noisy problem instances.

Instance	Highest fitness
Salustowicz (noisy)	0.959
Vladislavleva-3 (noisy)	0.961
Vladislavleva-4 (noisy)	0.962
Pagie (noisy)	0.963
Flow psi (noisy)	0.964
Korns-12 (noisy)	0.961

3.3 Selection of Predefined Features

For each problem instance we defined a small set of pre-calculated features. This was done manually and based on the known expression for the problem instance. The used features are shown in Table 5.

Table 5. The pre-calculated features for each problem instance. For Korns-12 we tried two configurations where we only add one of the necessary factors as feature in the first case and both in the second case.

Instance	Features
Salustowicz	$x^3, \exp(-x), \cos(x), \sin(x), \sin(x)^2 \cos(x)$
Vladislavleva-3	$\exp(-x_1), x_1^3, \cos(x_1), \sin(x_1), \sin(x_1)^2 \cos(x_1)$
Vladislavleva-4	$(x_1-3)^2, (x_2-3)^2, (x_3-3)^2, (x_4-3)^2, (x_5-3)^2$
Pagie	x^{-4}, y^{-4}
Flow psi	$\frac{x_4^2}{x_3^2}, \log(\frac{x_3}{x_4}), \sin(\frac{\pi x_2}{180})$
Korns-12 (1)	$\cos(9.8x_0), \sin(1.3x_4)$
Korns-12 (2)	$\cos(9.8x_0)$

4 Results

Figure 4 depicts the empirical run length distribution results for GP on the instances without noise. For each of the six instances we show the performance of GP with and without pre-calculated features. The graphs show the empirical success probability (target) for the configuration for the 60 runs. A run is successful if a solution with the defined level of quality is found. The evaluations are displayed relative to their total number.

Generally for all problem instances the number of successful runs is higher with extra features. The results show that for a given budget of evaluations the success probability is higher when extra features are available. Alternatively, for a given success probability the computational effort is lower when extra features are available. The results therefore support our hypothesis and we can give a positive answer to the research questions.

In practical applications we often need to accept inaccurate data or noisy measurements. The results for the noisy problem instances are shown in Fig. 5. We observe similar results as for the instances without noise. Only for the *Pagie* problem the extra features did not affect the performance.

5 Discussion

The results of our experiments show that the success rate of GP can be increased by providing pre-calculated features based on expert knowledge. For all tested problem instances without noise the probability of success for a given computational effort increased significantly. Some of the problem instances became almost trivial to solve when prior knowledge was available.

We found that the positive effect is apparent even for noisy problem instances.

A limitation of our contribution is that we used only synthetic benchmark problems where the underlying function is known. This makes it easy to come up with features which are necessary to express the functional relationship

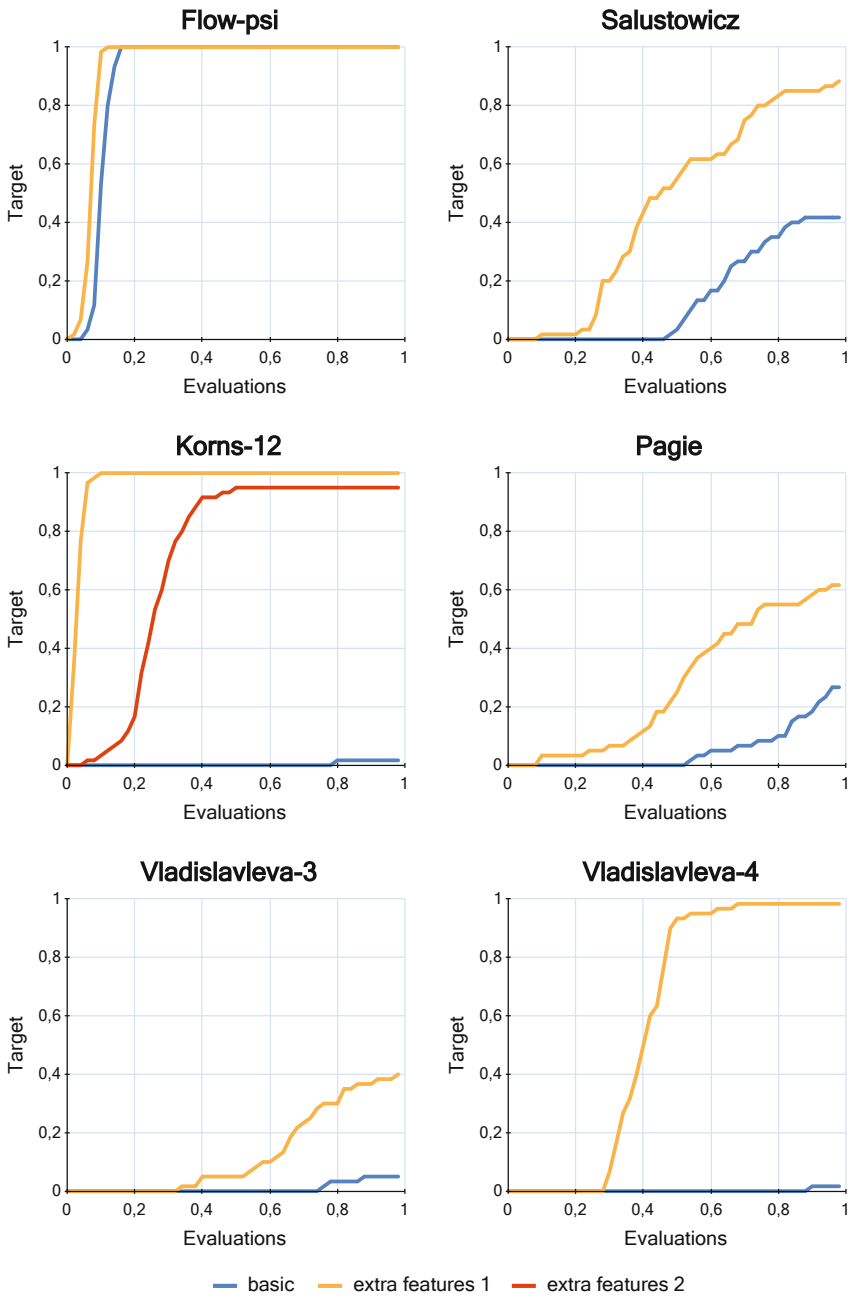

Fig. 4. Experiments without noise

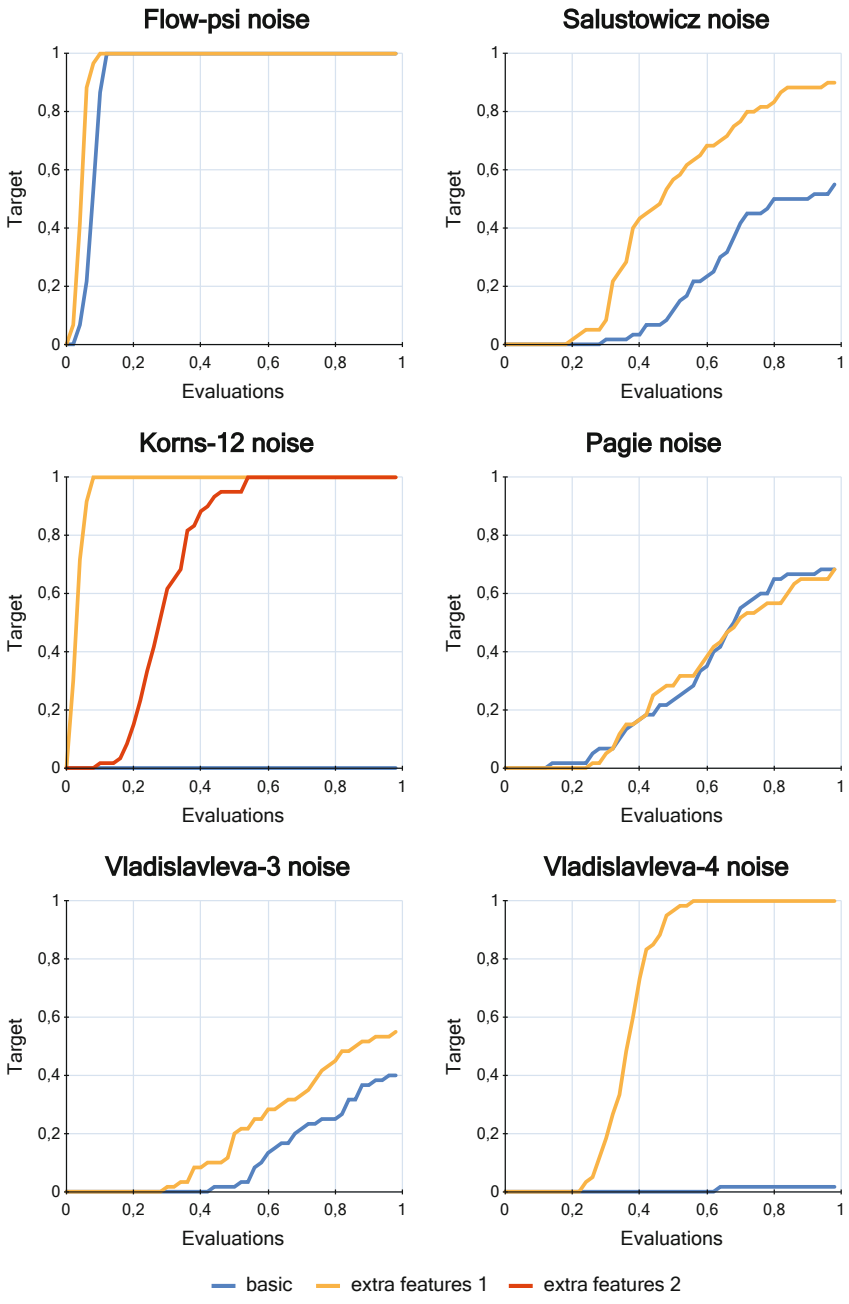

Fig. 5. Experiments with noise

that should be identified. In real-world applications where the underlying data-generating function is unknown it is harder to define such features. One particular limitation when using pre-calculated features is that non-linear parameters in the feature expressions must be approximated because they are not subject to evolutionary optimization by GP. This could be overcome by either providing parametric functions in the function set or defining syntactical building blocks which are used for pre-seeding of in crossover and mutation operators. However, such mechanisms would necessitate adaptations to the GP implementation which require more effort.

References

1. Affenzeller, M., Winkler, S., Wagner, S., Beham, A.: Genetic Algorithms and Genetic Programming - Modern Concepts and Practical Applications. CRC Press, Boca Raton (2009)
2. Ashburner, M., et al.: Gene ontology: tool for the unification of biology. Nat. Genet. **25**(1), 25 (2000)
3. Chandrasekaran, B., Josephson, J.R., Benjamins, V.R.: What are ontologies, and why do we need them? IEEE Intell. Syst. **14**(1), 20–26 (1999)
4. Chen, C., Luo, C., Jiang, Z.: A multilevel block building algorithm for fast modeling generalized separable systems. Expert Syst. Appl. **109**, 25–34 (2018). https://doi.org/10.1016/j.eswa.2018.05.021
5. Couchet, J., Manrique, D., Ríos, J., Rodríguez-Patón, A.: Crossover and mutation operators for grammar-guided genetic programming. Soft. Comput. **11**(10), 943–955 (2007)
6. Cramer, N.L.: A representation for the adaptive generation of simple sequential programs. In: Proceedings of the First International Conference on Genetic Algorithms, pp. 183–187 (1985)
7. Cruz, I.F., Xiao, H.: The role of ontologies in data integration. Eng. Intell. Syst. Electr. Eng. Commun. **13**(4), 245 (2005)
8. Darwin, C.: The Origin of Species: By Means of Natural Selection, or the Preservation of Favoured Races in the Struggle for Life. Cambridge Library Collection - Life Sciences, 6th edn. Cambridge University Press, Cambridge (2009)
9. Eilbeck, K., et al.: The sequence ontology: a tool for the unification of genome annotations. Genome Biol. **6**(5), R44 (2005)
10. Gardner, S.P.: Ontologies and semantic data integration. Drug Discovery Today **10**(14), 1001–1007 (2005)
11. Girardi, D., et al.: Interactive knowledge discovery with the doctor-in-the-loop: a practical example of cerebral aneurysms research. Brain Inf. **3**(3), 133–143 (2016)
12. Goldberg, D.E.: Genetic Algorithms in Search, Optimization, and Machine Learning. Addison-Wesley Professional, Boston (1989)
13. Gruber, T.R.: A translation approach to portable ontology specifications. Knowl. Acquisition **5**(2), 199–220 (1993)
14. Hansen, N., Auger, A., Ros, R., Finck, S., Pošík, P.: Comparing results of 31 algorithms from the black-box optimization benchmarking BBOB-2009. In: Proceedings of the 12th Annual Conference Companion on Genetic and Evolutionary Computation, pp. 1689–1696. ACM (2010)

15. Holland, J.H.: Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence. MIT Press, Cambridge (1992)
16. Holzinger, A., et al.: Interactive machine learning: experimental evidence for the human in the algorithmic loop. Appl. Intell. **49**(7), 2401–2414 (2019)
17. Kommenda, M., Kronberger, G., Wagner, S., Winkler, S., Affenzeller, M.: On the architecture and implementation of tree-based genetic programming in HeuristicLab. In: Companion Proceedings of the 14th Annual Conference on Genetic and Evolutionary Computation (GECCO 2012), pp. 101–108. ACM (2012)
18. Korns, M.F.: Accuracy in symbolic regression. In: Riolo, R., Vladislavleva, E., Moore, J. (eds.) Genetic Programming Theory and Practice IX, pp. 129–151. Springer, New York (2011). https://doi.org/10.1007/978-1-4614-1770-5_8
19. Koza, J.R.: Genetic Programming: On the Programming of Computers by Means of Natural Selection. MIT Press, Cambridge (1992)
20. Luke, S.: Two fast tree-creation algorithms for genetic programming. IEEE Trans. Evol. Comput. **4**(3), 274–283 (2000)
21. McKay, R.I., Hoai, N.X., Whigham, P.A., Shan, Y., O'Neill, M.: Grammar-based genetic programming: a survey. Genet. Program Evolvable Mach. **11**(3–4), 365–396 (2010)
22. Osborne, J.D., et al.: Annotating the human genome with disease ontology. BMC Genom. **10**(1), S6 (2009)
23. Pagie, L., Hogeweg, P.: Evolutionary consequences of coevolving targets. Evol. Comput. **5**(4), 401–418 (1997)
24. Ratle, A., Sebag, M.: Genetic programming and domain knowledge: beyond the limitations of grammar-guided machine discovery. In: Schoenauer, M., et al. (eds.) PPSN 2000. LNCS, vol. 1917, pp. 211–220. Springer, Heidelberg (2000). https://doi.org/10.1007/3-540-45356-3_21
25. Salustowicz, R., Schmidhuber, J.: Probabilistic incremental program evolution. Evol. Comput. **5**(2), 123–141 (1997)
26. Schoenauer, M., Sebag, M.: Using domain knowledge in evolutionary system identification. CoRR abs/cs/0602021 (2006). http://arxiv.org/abs/cs/0602021
27. Vladislavleva, E.J., Smits, G.F., Den Hertog, D.: Order of nonlinearity as a complexity measure for models generated by symbolic regression via pareto genetic programming. IEEE Trans. Evol. Comput. **13**(2), 333–349 (2009)
28. Whigham, P.A., et al.: Grammatically-based genetic programming. In: Proceedings of the Workshop on Genetic Programming: From Theory to Real-world Applications, vol. 16, pp. 33–41 (1995)
29. White, D.R., et al.: Better GP benchmarks: community survey results and proposals. Genet. Program Evolvable Mach. **14**(1), 3–29 (2013)
30. Winkler, S.M.: Evolutionary system identification: modern concepts and practical applications. Ph.D. thesis, Johannes Kepler University, Altenbergerstr. 69, 4040 Linz (2008)
31. Wong, M.L., Leung, K.S.: Data Mining Using Grammar Based Genetic Programming and Applications, vol. 3. Springer Science & Business Media, New York (2006). https://doi.org/10.1007/b116131