Convergence of Incremental Distributed Symbolic Regression.

Ben Cardoen
Department of Computer Science
University of Antwerp
Antwerp, Belgium
Email: bcardoen@sfu.ca

Jan Broeckhove
Department of Computer Science
University of Antwerp
Antwerp, Belgium
Email: jan.broeckhove@uantwerpen.be

Lander Willem
Centre for Health Economics Research and
Modelling Infectious Diseases
University of Antwerp
Antwerp, Belgium
Email: lander.willem@uantwerpen.be

***Abstract*—**
Symbolic regression (SR) builds algebraic expressions for an underlying model using a set of input-output data. Amongst its advantages are the ability to interpret the resulting expressions, determine important model features based on the presence in the expressions, and gain insights into the behavior of the resulting model (e.g. extrema). A key algorithm in SR is genetic programming (GP) to optimize the search process, though convergence characteristics are still an open issue. In this work, we elaborate on a distributed GP-SR implementation and evaluate the effect of communication topologies on the convergence of the algorithm. We evaluate a grid, tree and random topology with the aim of finding a balance between diffusion and concentration. We use a variation of k-fold cross validation to estimate the accuracy to predict unknown data points. This validation executes in parallel with the algorithm, thus combining the advantages of the cross validation with the increase in search space coverage. We introduce an incremental approach where the SR tool can start on partial data. This saves time, as the tool can be used in parallel with the simulator, which generates the input-output data. It also enables a human expert in the modeling loop, where partial results of the SR tool can be used to tune the design of experiments. We validate our work on several test problems and a use case on epidemiological simulations for the spread of measles.

I. Introduction

Symbolic regression (SR) is a supervised learning algorithm that fits variable length mathematical expression on a set of input features within a certain error distance to an expected output. SR evolves form and weights of the expression in tandem, unlike other forms of regression that only focus on the parameters. Its goal is to find a symbolic expression that best explains the observed data while at the same time being interpretable by the user.

A. Algorithm

The algorithm requires input data in a matrix X = n x k, and a vector Y = 1 x k of expected data, with n the number of features, or parameters and k the number of instances, or datapoints. It will generate and evolve expressions to obtain a 1 x k vector Y' that approximates Y when evaluated on X. We do not know in advance if all features are equally important to predict the response, which increases the complexity. The goal of the algorithm is to find f' such that

$$d(f(X), f'(X)) = \epsilon \tag{1}$$

results in ϵ minimal and f is the process we wish to approximate with f'. Not all distance functions are equally well suited for this purpose. A simple root mean squared error (RMSE) function has the issue of scale, the range of this function is [0, +∞), which makes comparison problematic, especially if we want to combine it with other objective functions. A simple linear weighted sum requires that all terms use the same scale. Normalization of RMSE is an option, however there is no single recommended approach to obtain this NRMSE. In this work we use a distance function based on the Pearson Correlation Coefficient r. Specifically, we define

$$r(Y, Y') = \frac{\sum_{i=0}^{n}(y_i - E[Y]) * (y'_i - E[Y'])}{\sqrt{\sum_{j=0}^{n}(y_j - E[Y])^2 * \sum_{k=0}^{n}(y'_k - E[Y'])^2}} \tag{2}$$

$$d(Y, Y') = 1 - |r| \tag{3}$$

The correlation coefficient ranges from -1 to 1, indicating negative linear and linear correlation between Y and Y', respectively, and 0 indicates no correlation. The distance function d has a range [0,1], which facilitates comparison across domains and allow us to make combinations with other objective functions. We not only want to assign a good (i.e. minimal) fitness value to a model that has a minimal distance, we also want to consider linearity between Y an Y'. The use of the Pearson Correlation Coefficient as a fitness measure has been used in [1].

1) Genetic Programming Implementation: In the context of symbolic regression, the GP algorithm controls a population of expressions, represented as binary expression trees. After initialization, they are evolved using mutation and cross-over operators to mimic genetic evolution. The algorithm is based on different phases in which the population is initialized based on a tree-archive populated by the user or previous phases. One phase consists of multiple generations or runs, where the GP operations are applied on a subset of the population. If this leads to fitness improvement, the population is replaced by this new set. At the end of a phase, the best scoring expressions are stored in the archive to seed consecutive or parallel phases.

We use a vanilla GP implementation with a 'full' initialization method [2]. Expressions trees are generated with a specified minimal and maximal depth, which differs from most GP optimization algorithms. We use 2 operators: mutation and crossover. First, mutation replaces a randomly selected subtree with a randomly generated expression tree. Next, crossover selects 2 trees based on fitness and swaps randomly selected subtrees between them. A stochastic process decides whether crossover is applied pairwise (between fitness ordered expressions in the population) or at random. The combination of new expressions and recombinations enables the exploration of the search space.

During the initialization or recombination of expression trees, it is possible to end up with invalid expressions for the given domain. The probability of an invalid expression increases exponentially with the depth of the tree. A typical example of an invalid tree is division by zero. Some approaches alter the division semantics to return a 'safe' value when the argument is zero. Our implementation discards invalid expressions and replaces them with a valid expression. We implemented a bottom up approach to detect and replace invalid trees. In contrast to a top down approach, this results in early detection and avoids redundant evaluations of generated invalid subtrees. However, the initialization constitutes a significant additional computational cost in the initialization stage of a phase and in the mutation operator.

2) Software: We implemented our distributed SR-GP algorithm, CSRM, in Python. It offers portability, rich libraries and fast development cycles. The disadvantages compared with compiled languages (e.g. C++) or newer scripting languages (e.g Julia) are speed and memory footprint. Python's use of a global interpreter lock makes shared memory parallelism infeasible but distributed programming is possible using MPI. The source code is provided in an open source repository (https://bitbucket.org/bcardoen/csrm) which also holds benchmark scripts, analysis code and plots. The project dependencies are minimal making the CSRM tool portable across any system with Python3, pip as an installation manager and MPI.

B. Distributed Algorithm

GP allows for both fine and coarse grained parallelism. In the first case, parallel execution of the fitness function can lead to a speedup in runtime without interfering with the search algorithm. Unfortunately, python's global interpreter lock and the resulting cost of copying expressions for evaluation makes this approach infeasible. With coarse grained parallelism, one executes multiple instances of the algorithm in parallel, which alters the search algorithm. Each process has its own phase with an expression tree population. Processes exchange their best expressions given a predefined communication topology. The topology is a key factor for the runtime and the convergence of the search process. Message exchange can introduce serialization and deadlock if the topology contains cycles. Our tool supports any user-defined topology.

The communication is based on messages, which are expression trees, sent from a source to a target population. After each phase, a process sends its best k expressions to each target based on the communication topology. To avoid deadlock, a process sends its expressions asynchronously, not waiting for acknowledgement of receipt. As such, the sent expressions are stored in a buffer together with a callable. After the sending stage, the process collects all messages from its source buffers, marks the messages as "used" and executes the next phase of the algorithm. Before sending messages, the process will verify that all previous messages have been collected by invoking the callable object. Once this blocking call is complete, it can safely reuse the buffer and start the next sending stage. This introduces a delay tolerance between processes since the phase runtime between processes can vary based on different expression lengths and evaluation cost. Without a delay tolerance, processes would synchronize on each other, nullifying any runtime gains. The delay tolerance is specified as a number of phases, which enables a process to advance multiple phases ahead of a target process in the topology. For hierarchical, non-cyclic topologies this can lead to a perfect scaling, where synchronization decreases as the number of processes increases.

C. Approximated k-fold Cross Validation

We divide the data over k processes, each using a random sample of 4/5 of the full input-output data. Subsequently, each process divides its data by 4/5 between training and validation. As such, the distributed process approximates a k-fold cross validation. Irrespectively of the topology, each pair of communicating processes has the same probability of overlapping data. When this probability is too low, overfitting occurs and expressions from one process are likely to be invalid for other process' training data. When the overlap is too extensive, both processes will be searching the same subspace of the search space. The process is detailed in Figure 1a.

D. Communication Topology

The process communication topology affects the convergence characteristics algorithm, which can be expressed as concentration and diffusion. Concentration refers to the partitioning of the search space, as discussed above. Diffusion refers to the spread of information over communicating processes to accelerate the optimization process of the entire group. However, if diffusion happens instantly, a suboptimal solution can dominate other processes, leading to premature convergence. An edge case is a disconnected topology without diffusion of information where each process has to discover highly fit expressions independently. This might be an advantage when the risk of premature convergence due to local optima is high. An approach without communication is sometimes referred to as partitioning, as one divides the search space in k distinct partitions. The distance between processes and connectivity will determine the impact of diffusion.

a) Grid: The grid topology is a two-dimensional square of k processes, with k the square of a natural number. Each process connects to four neighboring processes. The grid allows for delayed diffusion, because to reach all processes an optimal expression needs to traverse $\sqrt{k}$ links, and all processes are interconnected.

b) Tree: A binary tree topology acts with a root as a source and leafs as targets with unidirectional communication. For k processes, there are k-1 communication links, reducing the messaging and synchronization overhead significantly compared to the grid topology. Diffusion is restricted, because the information flow is unidirectional. On the other hand, optimal expressions are spread over the outgoing links (which is a spreading distribution policy) a partitioning effect occurs counteracting premature convergence. As there are no cycles, synchronization overhead is minimal.

c) Random: In a random topology, the convergence of the distributed algorithm is hard to predict. Cycles and cliques are likely, thus diffusion is not guaranteed and runtime performance depends on the actual instance. As advantage, patterns that might interfere with deterministic topologies are avoided. The only constraint we enforce on this topology is that each process has both a source and target, which are selected uniformly at random.

II. Related Work

A. Symbolic regression compared to other approaches

Symbolic regression is one technique that fits a model, given a set of input values, to a known output. Other machine learning techniques such as Neural Networks, Support Vector Machines have the same functionality but where SR distinguishes itself is in the white box nature of the model. The convergence characteristics of SR are an active field of study [3]. New approaches in SR such as GPTIPS and FFX [4, 5] focus on multiple linear regression, where a linear combination of base functions is generated. While GPTIPS still uses GP, FFX is completely deterministic and eschews GP. In the recent work of [6] SR is compared with these approaches on a series of benchmark functions, some of which are used in this work. The authors concluded that while SR can have a slower convergence rate compared to conventional machine learning algorithms, the difference is not that large. SR distinguishes itself from other techniques by returning a model that allows for understanding and insight into the process we are approximating. Symbolic regression has been used to evolve implicit equations [7]. The difference with classical SR and other machine learning techniques is striking. As we have seen there are a near infinite number of equivalent equations that fit input to output data. Apart from issues such as representation, selecting a preferred solution is a hard problem, often problem domain dependent. An additional difficulty is providing the algorithm with negatives, values that the surrogate model generated by SR should not produce. These can be used to drastically reduce the number of equivalent solutions and thus increase accuracy. The authors resolved these issues by using a derivative based fitness function in order to find implicit equations that are nontrivial solutions for the datapoints.

B. Algorithms implementing symbolic regression

Even though SR is usually associated with GP, there exists a wide variety of alternative implementations. A non GP approach using elements from Grammatical Evolution (GE) [8], several genetic algorithm techniques and continuous optimizers (DE, PSO) has been presented in [9] with promising results in terms of convergence rate and accuracy. It uses a simple C-like expression grammar, including relational and conditional operators. The more recently introduced ABC algorithm has also been used for SR [10]. Ant Colony Optimization [11] has been used to generate programs [12], the same functionality that allows GP to be used for SR.

C. Genetic programming

A recent study offers a valuable overview of open issues in genetic programming [13]. This study lists issues we have covered such as problem hardness, fitness topology, problem representation, benchmarks, uncertainty regarding the optimal solution, and constant optimization. We restrict our work to a simple GP implementation as a baseline for future improvements. Advances such a semantically aware operators [14] and modularity are not applied. Modularity started with Koza's [2] Automatically defined functions (ADF), which allow reuse of partial solutions as base functions and thus signficantly increase the expressiveness of a solution without increasing the representation. This concept is further investigated in [15] where features such as structure modification and recursion are evaluated. In this work it is shown that with modularity the size of the representation in GP is no longer bound to the number of features. In our work we do not have modularity so we still have a correlation between tree depth and the number of features.

D. Parallel symbolic regression

Enrique Alba's book [16] on parallel metaheuristics is the reference work for the field. It provides a broad overview of the challenges and advantages of parallel metaheuristics. SR can be implemented using a parallel metaheuristic such as GP. A Python toolbox for evolutionary algorithms has been developed [17], not specifically directed at symbolic regression but as a repository of evolutionary algorithms in a distributed context. An interesting parallel GP SR implementation [18] introduces a random islands model where processes are allowed to ignore messages, contrary to our approach. The authors argue that this promotes niching, where 'contamination' of locally (per process) fit individuals could otherwise introduce premature convergence. The clear advantage of such a system is speedup, since no process ever waits on other processes. Another difference is the message exchange protocol. Whereas our tool exchanges messages after each phase, their tool uses a probability to decide per process if messages are sent or received interleaved with the generations. Such a setup allows for a heterogeneous set of processes. Each process would execute a different configuration of an optimization algorithm or even a different algorithm altogether. This approach tries to mitigate the disadvantages of some algorithms or configuration with the advantages of others. It is an approximation exercise in avoiding the constraint posed by the NFL theorem. A heterogeneous approach can also serve as a self optimizing metaheuristic by testing parameter configurations in parallel or even responding to metadata generated from the convergence process and adjusting these parameters based on findings from other processes. It executes a two layer optimization process: it approximates the original problem and the optimal configuration for solving the problem. A superlinear speedup is reported for some problems. As we have argued before, ignoring messages introduces non determinism in the algorithm without there being a clear need for this. Niching, islands and prevention of premature convergence can be achieved by other methods. Adding constraints to the fitness function is one approach that promotes niching. Making the distributed algorithm non deterministic makes analysis and experimentation far more complex. A different approach is shown in [19] where a master slave topology is used in combination with a load balancing algorithm in order to resolve the imbalance between the different slaves executing uneven workloads. The slaves do not form separate processes, they are assigned a subset of the population and execute only the fitness function. The selection and evolution steps are performed by the master process. This a a fine grained approach, and while it offers a speedup in comparison with a sequential GP SR implementation it does not increase the coverage of the search space. The load balancing algorithm is an interesting approach to solve the unavoidable imbalance between the processes. We mitigate this issue with our delay tolerance, but as we have discussed this approach has limits. A master slave topology allows for a centralized approach at the cost of introducing synchronization or even serialization. A distributed variant would be interesting to apply, where the processes distribute the load or alter the communication topology based on their load. Distributed election algorithms could be of use here to elect a coordinator each round. A partitioning of the processes in subgroups based on their location in the topology and communication pattern are other alternatives. In Distributed Genetic Progamming (DGP) [20] a ring and torus topology are used. The two way torus topology is similar to our grid topology. The study finds that sharing of messages is essential to improve convergence but that the communication pattern is largely defined by the problem domain. It concludes that diffusion is a more powerful technique compared to partitioning. In partitioning no communication between subgroups is possible, which can protect against premature convergence.

E. Accuracy and convergence

The Accuracy and convergence characteristics of SR are an open issue [21, 3, 22]. A practitioner would like to have certainty regarding the convergence characteristics and accuracy of SR. Given a problem, she would like upper bounds on both when applying SR to the problem. Without these acceptance of SR as a tool in industry will remain difficult. Finding good benchmark problems for GP is an open issue, recent work [23] attempts at unifying existing benchmarks and defining standards for existing and new benchmark problems. In this work we use the benchmarks introduced by [21], hard problems with poor convergence characteristics for a simple GP SR implementation. A measure estimating problem hardness is introduced in [24]. This measure is able to predict the effectiveness of operators, or conversely, estimate the hardness of the problem. The measure has a theoretical foundation, it allows a practitioner robust insights into the problem. Problem hardness will affect the convergence characteristics of any metaheuristic, with this hardness measure we can analyze convergence and determine if the algorithm configuration or problem hardness is the cause for poor convergence.

III. Experiments

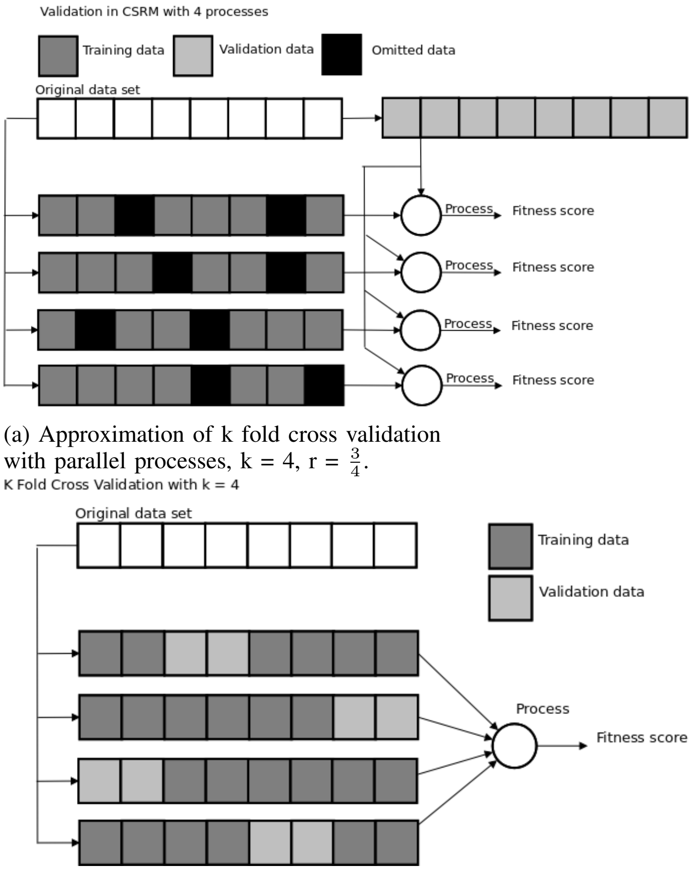
The project's source code, benchmark scripts, analysis code and plots are provided in an open source repository. Additional results, left out here due to space constraints are covered in work. The experiments were performed on an Intel Xeon E5 2697 processor with 64GB RAM, with Ubuntu 16.04 LTS. The experiments use a fixed seed in order to guarantee reproducibility. For the parallel experiments we use 25 processes.

a) Benchmark problems: Recent work on the convergence of GP-based SR [3, 22] featured a set of benchmarks that pose convergence problems for SR implementations. These 15 benchmarks use at most five features and are non-linear arithmetic expressions using standard base functions : (sin, cos, tan(h), log,a^x, /, *, modulo, abs, min, max, +, -). CSRM does not know in advance which features are used. It only assumes that each problem is a function of maximum 5 features, testing the robustness of the algorithm.

b) Experiment setup: Each process has a population of 20 expression trees with an initial depth of 4, max depth of 8, executes 20 phases of 20 runs or generations with an archive size of 20 expressions. Per phase, the 4 best expressions are archived. The benchmark functions have at most 5 features with 20 sample points in the range [1,5]. The Grid and Tree topology share their best expression per link, the Random topology shares 2 expressions per link. We used 25 processes, resulting in respectively 400, 15, 50 expressions being sent per phase. The random topology in this case contains is a disconnected set of 2 cycles.

At the end of an experiment, the best 20 expressions from all processes are collected and scored. We measure the fitness on the training data, and the fitness on the validation data. Finally, we record the mean fitness values of the best 5 expressions, both on the training and validation data, as convergence rate. The mean is restricted to the upper quarter of the population specifically to measure how the best expressions are distributed. This measure records the convergence more accurately as the fittest expressions drive the convergence rate, or put differently, it ignores the effect outliers would otherwise have on the mean. Fitness values fluctuate strongly across test problems and topologies. The results are presented in relative to the Tree topology to measure a relative gain or loss in orders of magnitude.

c) Convergence: For the best fitness, the first 5 benchmarks, with exception of the second, all have identical values on training and validation data. The processes converged to zero fitness values for these problems, hence the identical results. The fitness results on the training data, presented



(a) Approximation of k fold cross validation with parallel processes, k = 4, r = $\frac{3}{4}$.

(b) Visualization of k fold cross validation with k = 4.

Fig. 1: K fold cross validation approximation in CSRM.

in Figure 2a, indicate that the Grid and Random topology have superior convergence characteristics compared to the Tree topology, with Grid outperforming Random on more than half of the benchmarks. When we look at the fitness values on the validation data in Figure 2b, we observe more nuanced results. Overall, the Grid topology is the best choice regarding best fitness values and the Random topology sometimes performs inferior compared to the Tree topology (e.g. benchmark 11 and 12). If we look at the mean fitness values on training and validation data, the Tree topology outperforms the Grid and Random topology on 2 benchmarks. However, the Grid topology performs still the best for most problems, with the exception of benchmark 7 where the Random topology dominates. The similarity between the results of the training and validation data, both for the best and mean fitness, indicates a good predictive capability since overfitting would result in a reverse patterns for the training and validation data.

d) Measuring Overhead: Cycles in the topology lead to excessive synchronization and serialization. We measure the mean execution time for benchmark 6, which leads to different convergence characteristics based on the topology, making this a good testcase. The processes will communicate 25 times. The runtime of one phase depends on the number of generations, population size and depth ranges of the expressions. Ideally, we would like for a user to choose these parameters based on the problem at hand and not be constrained by synchronization considerations. To compare the three topologies, we use the disconnected or 'none' topology as a reference point, as it has zero synchronization overhead and has an ideal speed-up of n, where n is the processcount. From the synchronization overhead we can then derive the speed-up each of the topologies is able to offer. In practice even the 'none' topology will have some synchronization overhead, as the root process has to collect all results. Measuring communication overhead becomes hard if the runtime of a single phase is very long. If they are is too short, overhead dominates the entire runtime and it will unfairly penalize topologies with cycles forcing them to serialize.

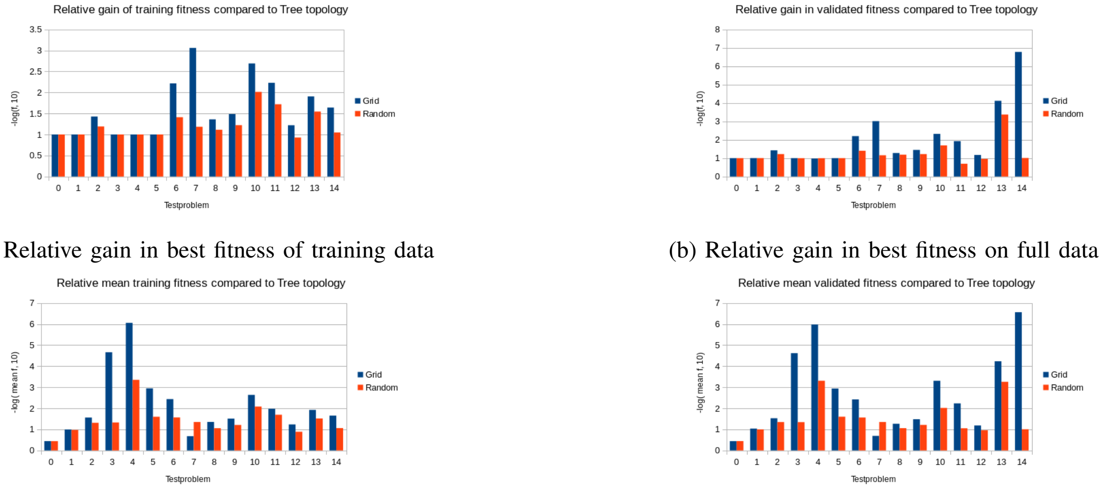
Figure 3 shows that the Tree topology has almost no delay caused by synchronization. This is due to the delay tolerance we have built in in our implementation. The Random topology has an average delay factor of 1.3 and the Grid topology has an average delay factor of nearly 2. This is easily translated in terms of speed-up. As such, the Tree topology will have near linear speedup, a Grid topology roughly half of that and a Random topology will have an intermediate speed-up. The standard deviation on the speed-up for the Tree topology is significantly smaller indicating a predictable speed-up. It should be noted that selecting a different benchmark will lead to differing convergence characteristics, which in turn can mask synchronization effects. However, covering all benchmarks with all possible configurations is outside the computation scope of this work.

IV. Use Case

Infectious diseases have substantial impact on health care, economics and society. Observational data might not be fully predictive for future outbreaks in adapted populations or settings and experiments in the real world process can have practical, budgetary or ethical constraints. Therefore, simulation and surrogate modeling can be very useful to mimic real world process and to inform policy makers on the prevention of infectious diseases and to obtain insights in transmission dynamics.

Our aim is to apply our CSRM tool to the input-output data of a computationally intensive high-performance simulator, STRIDE [25], which models the transmission of infectious diseases. The simulator is calibrated for measles outbreaks in Antwerp, a city in Belgium, with a population size of 500000 individuals. Of vital importance here is the population immunity aspect and as such our research question is: how does the immunization fraction influence the outbreak of measles? In this paper, we focus on the convergence characteristics of the surrogate models rather than their domain specific implications.

A single simulation run with this individual-based model is computationally expensive hence surrogate models, approximating the detailed simulator, can be useful as an emulator to improve model exploration and facilitate rapid policy making



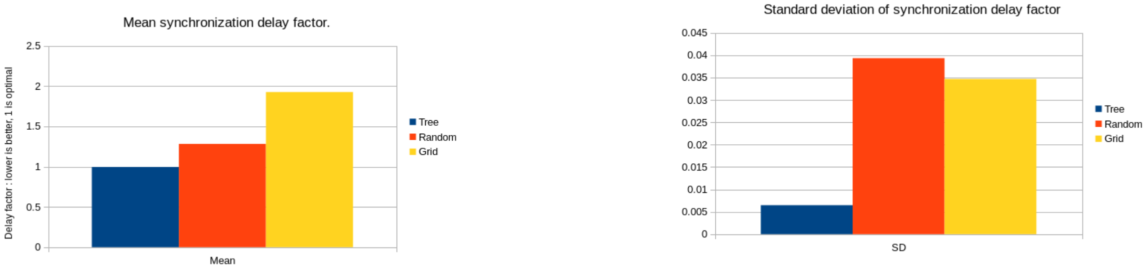
(a) Relative gain in best fitness of training data

(b) Relative gain in best fitness on full data.

(c) Relative gain in mean fitness on training data.

(d) Relative gain in mean fitness on full data.

Fig. 2: Convergence differences between topologies.



(a) Mean synchronization delay factor.

(b) Standard deviation of synchronization delay factor.

Fig. 3: Synchronization overhead introduced by topologies.

in various settings. Symbolic regression can be used to obtain surrogate models based on input-output simulation data. Generating all simulation output in sequence prior to building the surrogate model leads to significant downtime for the user. Therefore, we elaborate on the value of an incremental approach with surrogate models based on partial results of the simulation model with the aim to setup a trustworthy feedback loop between the user, simulator and regression tool. This incremental approach provides partial results to a domain expert to enhance the model building and design of experiments. Our CSRM tool is able to reuse partial results to seed subsequent runs.

a) Design of experiment: To run the disease transmission simulator, we used a space filling design to maximize the sampling of the parameter space and minimize the number of evaluation points. We applied a Latin Hypercube Design, specifically the Audze-Eglais [26, 27, 28], which uses the Euclidean distance measure but in addition obtains a more uniform distribution of the individual points. This design relies on the concept of minimizing a potential energy between design points, a measure based on the inverse of the euclidean distance.

$$E^{AE} = \sum_{i=0}^{p-1} \sum_{j=i+1}^{p-1} \frac{1}{d_{ij}}$$

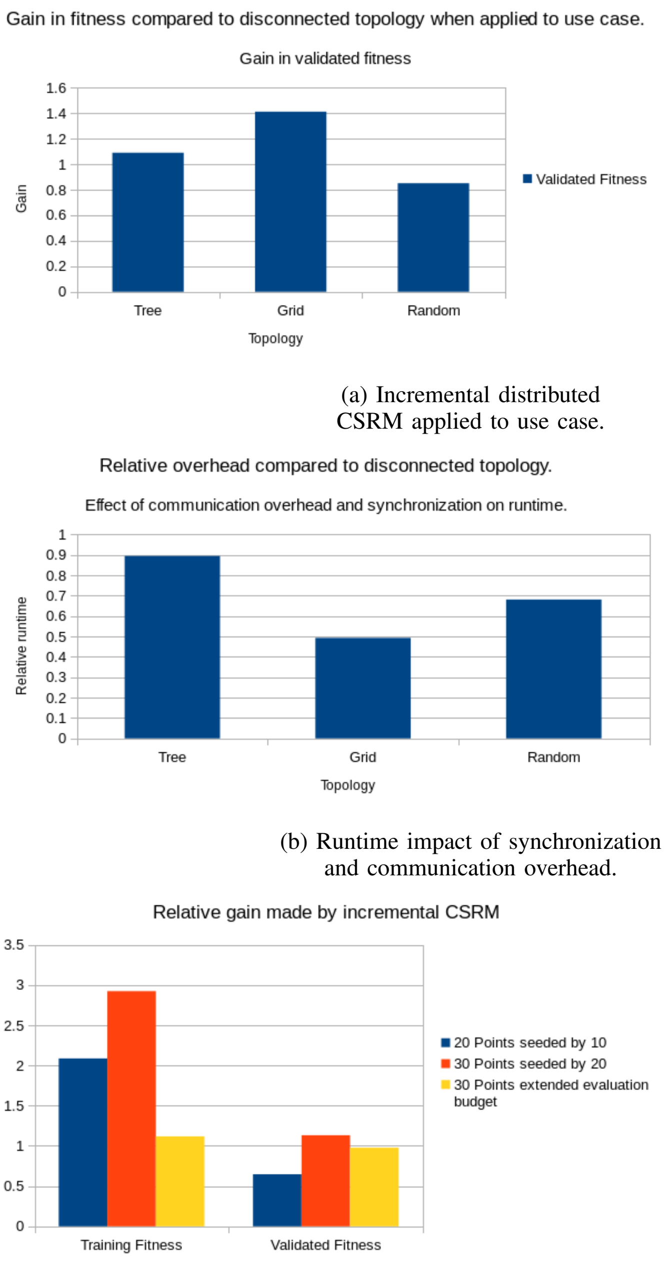
We constructed an experimental design with 3 dimensions and 30 points, using the tool introduced by Husslange et al. [29], for the following parameters:

- Basic reproduction number (R_0) : the expected number of secondary infections caused by a primary infection in a fully susceptible population, [12-20]
- Starting set of infected persons (S) : Number of persons in the population that is an infected person at the start of the simulation, [1-500]
- Immunity fraction (I) : Fraction of the population that is immune to the disease at the start of the simulation. [0.75, 0.95]

For each parameter we used 30 points uniformly chosen in their domain and each configuration is used to run the disease simulator once. We used the the attack rate as output parameter, which is the rate of the total number of cases in the population versus the size of the population.

b) Symbolic regression configuration: We run the GP-SR with 30 phases, each consisting of 60 generations of 20 expressions with an initial depth of 3 and maximum depth of 6. The 4 best expressions of each phase are archived. In our previous section we used 5 expressions to archive, with a maximum communication size of 4 per turn. This introduces selection pressure even in the first phase of communication. By using 4 we now test if the absence of this pressure in the first turn has some effect on convergence. We compare 3 approaches based on fitness and computational cost. Firstly, we run all 30 configurations and use the CSRM tool on the entire simulation dataset. This is the traditional approach where the GP-SR tool starts a blind search and serves as baseline in our comparisons. Secondly, we split the configurations into incremental sections. We start the CSRM tool with the results from 10 configurations. The best 4 results are saved to disk and used to seed the GP-SR with 20 simulated configurations. The results of the 20-point dataset are used to seed for the 30-point CSRM run. The computational cost of analyzing 10-, 20-, or 30-point simulation output is similar. Thirdly, we run the tool on the data from 30 configurations with a double amount of phases. As such, the same number of fitness evaluations are performed as with the 10-20-30 combination. Finally we run a distributed experiment seeded by a 10-20-30 configuration and observe the effects on convergence and speedup. We are particularly interested in the duplication of our empirical results on the benchmarks compared with the use case.

1) Fitness improvement: We compare the seeded and extended CSRM runs with the baseline 30-point run. Figure 4c shows that the fitness, accuracy to predict the response, improved for the training data for all methods. For the validation data, we observed that the ”20-point analysis seeded by the 10-point run” performs worse compared to the blind search on the full output. As such, 2/3rd of the simulations seems not enough, leading to overfitting. If we seed the best results from the 20-point CSRM run into a 30-point configuration, we observed that both the training and validated fitness values substantially improved. The 30-point CSRM run with 60 phases has the same computational cost as the combined 10-20-30 run, but has only little to no gain in convergence by the additional phases. We see that convergence is slowing, with training fitness improving by a factor of 10 %, at the cost of worsening validation fitness. This is a typical example of overfitting. The combined 10-20-30 run increases validation fitness with a factor of 13%



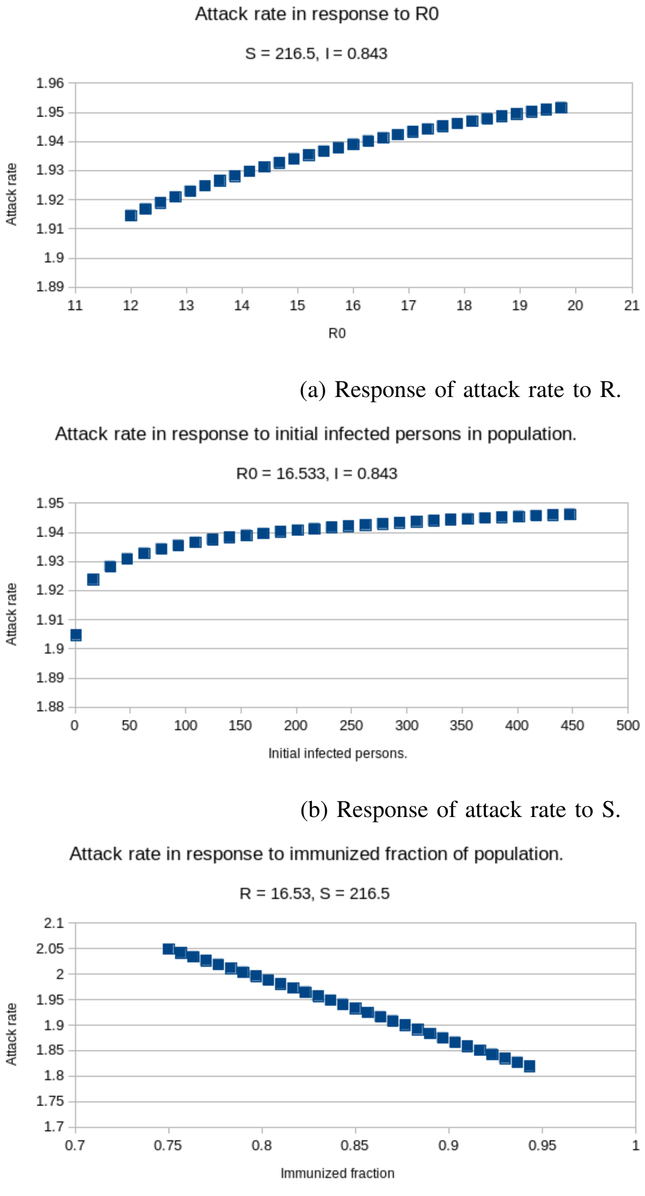
(a) Incremental distributed CSRM applied to use case.

(b) Runtime impact of synchronization and communication overhead.

(c) Incremental fitness gain in CSRM.

Fig. 4: Use case : Effect of topology when applied to use case.

2) Distributed: We performed the 10-20-30 analysis also in the distributed setting and compared the topologies in terms of fitness improvement and speed-up. We used 25 processes in a Tree, Grid and Random and Disconnected topology. Figure 4a presents the gain in fitness on the validation data for the Tree, Grid and Random topologies compared to the Disconnected topology. We observed that the diffusion in the Grid topology leads to the highest gain, followed by the Tree topology. Interestingly, the Random topology scored less than the Disconnected topology. This can occur when an early local optimum dominates the remainder of the process. The effect on runtime is showed in Figure 4b, with the Tree topology encountering minimal overhead. The Grid topology has a performance penalty of factor 2 and the Random topology performed intermediate. During the experiments, we observed that the progress of the processes in the Tree and Disconnected topologies could vary up to 4 phases. This can be explained by the distance between two processes in the Tree topology, which is at most 4 (depth of a 25-node binary tree). As such, if we increase the number of processes in the Tree topology, it will scale better.



(a) Response of attack rate to R.

(b) Response of attack rate to S.

(c) Response of attack rate to I.

Fig. 5: Use case : Response of attack rate to R, S, I.

Based on runtime and scaling, we obtained the best results with the distributed application of CSRM with a Tree topology. This resulted in an algebraic expression with a fitness value of 0.039 on the full data set. However, it is still 10 orders of magnitude removed from the optimal prediction of the response, which is an indication of the value that partial results can offer. The effect of each parameter on the response can be explored using response plots. As such, we vary each of the parameters while keeping the others fixed to the midpoint of the range, and calculate an estimate of the attack rate. We observed that our surrogate model predicts an attack rate outside of the valid range of [0,1]. This might be explained by the scaling factor of 10 between the prediction and the actual output from the simulator. The surrogate models from the CSRM tool are trained with 30 data points and not the full factorial design. Nonetheless, the response plots can be used to evaluate points that were not available in our training set. The trends in the response plots are in line with the literature [30]. R_0 increases logarithmic with the attack rate, which is in line with theoretical and empirical results. An increasing trend is also observed with the initial number of infected persons. An in crease in the immunization fraction shows a decease in the predicted attack rate. While the exact values of the predicted attack rate from our suboptimal surrogate model are not yet correctly, the expected trends are. This justifies our incremental approach since surrogate models will match the trend first, rather than fitting individual points. This can be partly explained by our usage of the Pearson R correlation coefficient as basis for the fitness function.

V. Conclusion

We have introduced a distributed SR tool with a delay tolerance mechanism that mitigates load imbalance between processes. We have compared three representative topologies in terms of convergence rate and speed-up. The tree topology can be used as an approximation for the grid topology with near linear speed-up and offers a process a delay tolerance equal to the distance between dependent processes. This feature improves the tree topology to scale when the process set is larger. The distributed processes approximate K fold cross validation in order to avoid overfitted solutions without its computational cost. Our modular architecture allows the tool to be extended with new topologies, policies, and optimization algorithms.

In a use case we have demonstrated the use our tool can be used to build a surrogate models for a simulator in an incremental approach, i.e. concurrent to the simulator running over all input data sets. For the use case, it lead to improvements in fitness and predictive value of the final model. A feedback loop between practitioner, simulator and regression tool offers savings in time while yielding insights that can be used by domain experts to enhance the model building. The use case demonstrated that intermediate solutions are able to approximate the final model's characteristics sufficiently to be of value, thus validating our incremental approach.

The distributed results of the use case were in line with the results of the benchmarks. The grid topology obtained the highest quality solution at the lowest speed-up. The tree topology achieved a near linear speed-up at the cost of a lower quality solution. The random topology demonstrated that the incremental approach can lead to overfitting in a distributed setting.

Future work can take any of three directions. First our distributed architecture allows for a each process to use a different metaheuristic, creating a heterogeneous set of communicating algorithms. A cooperative set of optimization algorithms could offer an optimal solution for all problem instances by balancing the disadvantages and advantages of each algorithm. Second, the tool can easily be extended with new topologies and spreading policies to further investigate their effects on convergence and accuracy. Lastly, we can approximate incremental design of experiment by making use of the collapsing property of Latin hypercubes. By reducing the number of parameters and fixing the other values, we could avoid the naive linear division approach we thus far have used in seeding the tool and possibly maintain the space filling characteristics of the LHD.

References

[1] L. Willem, S. Stijven, E. Vladislavleva, J. Broeckhove, P. Beutels, and N. Hens, "Active learning to understand infectious disease models and improve policy making," *PLOS Computational Biology*, vol. 10, no. 4, pp. 1–10, 04 2014. [Online]. Available: https://doi.org/10.1371/journal.pcbi.1003563

[2] J. R. Koza, *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. Cambridge, MA, USA: MIT Press, 1992.

[3] M. F. Korns, *Accuracy in Symbolic Regression*. New York: Springer, 2011, pp. 129–151. [Online]. Available: http://dx.doi.org/10.1007/978-1-4614-1770-5_8

[4] D. P. Searson, D. E. Leahy, and M. J. Willis, "Gptips:an open source genetic programming toolbox for multigene symbolic regression."

[5] T. McConaghy, "FFX: Fast, Scalable, Deterministic Symbolic Regression Technology," in *Genetic Programming Theory and Practice IX*, R. Riolo, E. Vladislavleva, and J. H. Moore, Eds. Springer New York, 2011, pp. 235–260. [Online]. Available: http://link.springer.com/chapter/10.1007/978-1-4614-1770-5_13

[6] J. Zegklitz and P. Posík, "Symbolic regression algorithms with built-in linear regression," *CoRR*, vol. abs/1701.03641, 2017. [Online]. Available: http://arxiv.org/abs/1701.03641

[7] M. Schmidt and H. Lipson, *Symbolic Regression of Implicit Equations*. Boston, MA: Springer US, 2010, pp. 73–85. [Online]. Available: http://dx.doi.org/10.1007/978-1-4419-1626-6_5

[8] M. O'Neil and C. Ryan, *Grammatical Evolution*. Boston, MA: Springer US, 2003, pp. 33–47. [Online]. Available: http://dx.doi.org/10.1007/978-1-4615-0447-4_4

[9] M. F. Korns, *Abstract Expression Grammar Symbolic Regression*. New York: Springer, 2011, pp. 109–128. [Online]. Available: http://dx.doi.org/10.1007/978-1-4419-7747-2_7

[10] D. Karaboga, C. Ozturk, N. Karaboga, and B. Gorkemli, "Artificial bee colony programming for symbolic regression," *Inf. Sci.*, vol. 209, pp. 1–15, Nov. 2012. [Online]. Available: http://dx.doi.org/10.1016/j.ins.2012.05.002

[11] M. Dorigo, M. Birattari, and T. Stutzle, "Ant colony optimization," *Comp. Intell. Mag.*, vol. 1, no. 4, pp. 28–39, Nov. 2006. [Online]. Available: http://dx.doi.org/10.1109/MCI.2006.329691

[12] O. Roux and C. Fonlupt, "Ant programming: or how to use ants for automatic programming," in *Proceedings of ANTS*, vol. 2000. Springer Berlin, 2000, pp. 121–129.

[13] M. O'Neill, L. Vanneschi, S. Gustafson, and W. Banzhaf, "Open issues in genetic programming," *Genetic Programming and Evolvable Machines*, vol. 11, no. 3, pp. 339–363, 2010. [Online]. Available: http://dx.doi.org/10.1007/s10710-010-9113-2

[14] N. Q. Uy, N. X. Hoai, M. ONeill, R. I. McKay, and E. Galván-López, "Semantically-based crossover in genetic programming: application to real-valued symbolic regression," *Genetic Programming and Evolvable Machines*, vol. 12, no. 2, pp. 91–119, 2011.

[15] M. Dostál, *Modularity in Genetic Programming*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 365–393. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-30504-7_15

[16] E. Alba, *Parallel Metaheuristics: A New Class of Algorithms*. Wiley-Interscience, 2005.

[17] F.-A. Fortin, F.-M. De Rainville, M.-A. G. Gardner, M. Parizeau, and C. Gagné, "Deap: Evolutionary algorithms made easy," *J. Mach. Learn. Res.*, vol. 13, no. 1, pp. 2171–2175, Jul. 2012. [Online]. Available: http://dl.acm.org/citation.cfm?id=2503308.2503311

[18] A. Salhi, H. Glaser, and D. De Roure, "Parallel implementation of a genetic-programming based tool for symbolic regression," *Inf. Process. Lett.*, vol. 66, no. 6, pp. 299–307, Jun. 1998. [Online]. Available: http://dx.doi.org/10.1016/S0020-0190(98)00056-8

[19] M. Oussaidène, B. Chopard, O. V. Pictet, and M. Tomassini, "Parallel genetic programming: An application to trading models evolution," in *Proceedings of the 1st Annual Conference on Genetic Programming*. Cambridge, MA, USA: MIT Press, 1996, pp. 357–362. [Online]. Available: http://dl.acm.org/citation.cfm?id=1595536.1595586

[20] T. Niwa and H. Iba, "Distributed genetic programming: Empirical study and analysis," in *Proceedings of the 1st Annual Conference on Genetic Programming*. Cambridge, MA, USA: MIT Press, 1996, pp. 339–344. [Online]. Available: http://dl.acm.org/citation.cfm?id=1595536.1595583

[21] M. F. Korns, *Accuracy in Symbolic Regression*. New York: Springer, 2011, pp. 129–151. [Online]. Available: http://dx.doi.org/10.1007/978-1-4614-1770-5_8

[22] ——, *A Baseline Symbolic Regression Algorithm*. New York: Springer, 2013, pp. 117–137. [Online]. Available: http://dx.doi.org/10.1007/978-1-4614-6846-2_9

[23] J. McDermott, D. R. White, S. Luke, L. Manzoni, M. Castelli, L. Vanneschi, W. Jaskowski, K. Krawiec, R. Harper, K. De Jong, and U.-M. O'Reilly, "Genetic programming needs better benchmarks," in *Proceedings of the 14th Annual Conference on Genetic and Evolutionary Computation*, ser. GECCO '12. New York, NY, USA: ACM, 2012, pp. 791–798. [Online]. Available: http://doi.acm.org/10.1145/2330163.2330273

[24] R. Poli and L. Vanneschi, “Fitness-proportional negative slope coefficient as a hardness measure for genetic algorithms,” in *Proceedings of the 9th Annual Conference on Genetic and Evolutionary Computation*, ser. GECCO ’07. New York, NY, USA: ACM, 2007, pp. 1335–1342. [Online]. Available: http://doi.acm.org/10.1145/1276958.1277209

[25] L. Willem, S. Stijven, E. Tijskens, P. Beutels, N. Hens, and J. Broeckhove, “Optimizing agent-based transmission models for infectious diseases,” *BMC bioinformatics*, vol. 16, no. 1, p. 183, 2015.

[26] S. J. Bates, J. Sienz, and D. S. Langley, “Formulation of the audze–eglais uniform latin hypercube design of experiments,” *Adv. Eng. Softw.*, vol. 34, no. 8, pp. 493–506, Jun. 2003. [Online]. Available: http://dx.doi.org/10.1016/S0965-9978(03)00042-5

[27] S. Bates, J. Sienz, and V. Toropov, “Formulation of the optimal latin hypercube design of experiments using a permutation genetic algorithm,” in *45th AIAA/ASME/ASCE/AHS/ASC Structures, Structural Dynamics & Materials Conference*, 2004, p. 2011.

[28] P. Audze and V. Eglais, “New approach for planning out of experiments,” *Problems of dynamics and strengths*, vol. 35, pp. 104–107, 1977.

[29] B. G. M. Husslage, G. Rennen, E. R. van Dam, and D. den Hertog, “Space-filling latin hypercube designs for computer experiments,” *Optimization and Engineering*, vol. 12, no. 4, pp. 611–630, 2011. [Online]. Available: http://dx.doi.org/10.1007/s11081-010-9129-8

[30] J. J. Grefenstette, S. T. Brown, R. Rosenfeld, J. DePasse, N. T. Stone, P. C. Cooley, W. D. Wheaton, A. Fyshe, D. D. Galloway, A. Sriram, H. Guclu, T. Abraham, and D. S. Burke, “Fred (a framework for reconstructing epidemic dynamics): an open-source software system for modeling infectious diseases and control strategies using census-based populations,” *BMC Public Health*, vol. 13, no. 1, p. 940, 2013. [Online]. Available: http://dx.doi.org/10.1186/1471-2458-13-940