Convergence of incremental distributed symbolic regression.

Ben Cardoen
ben.cardoen@student.uantwerpen.be—bcardoen@sfu.ca

1 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.

2 Introduction and design

2.1 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 is the number of features, or parameters. It will generate and evolve expressions to obtain an 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 $d(f(X), f'(X)) = \epsilon$ 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, $+\infty$), 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

$$d(Y, Y') = 1 - |\frac{\sum_{i=0}^{n}(y_i - E[Y]) * (y'_i - E[Y'])}{\sqrt{\sum_{j=0}^{n}(y_j - E[Y])^2 * \sum_{k=0}^{n}(y'_k - E[Y'])^2}}|$$

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 [13].

2.1.1 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 an 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 [8]. 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 or new expressions and recombinations enable 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 alters 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 an early detection and avoids redundant evaluations of generated subtrees. However, the initialization constitutes a significant additional computational cost in the initialization stage of a phase and in the mutation operator.

2.1.2 Software

We implemented our distributed SR-GP algorithm, CRSE, 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 repository which also holds benchmark scripts, analysis code and plots. The project dependencies are minimal making the CRSE tool portable across any system with Python3, pip as an installation manager and MPI.

2.2 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 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.

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 boolean. 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. 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.

2.3 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.

2.4 Communication Topology

The process communication topology affects the convergence characteristics algorithm, which can be expressed as concentration and diffusion. Concentration refers to the partitioning the search space, as discussed above. Diffusion refers to the spread of information over communicating processes to accelerates 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.

Grid The grid topology is 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.

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.

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.

3 Related Work

The reference work of E. Alba [1] provides a broad overview of the challenges and advantages of parallel metaheuristics. In a random island model for a parallel GP-SR implementation [11], processes are allowed to ignore messages. The authors argue that this promotes niching, where 'contamination' of locally (per process) fit expressions could introduce premature convergence. The advantage of such a system is speed-up, since no process has to waits on other processes. This approach interleaves message exchange with computation during a phase, which allows for a heterogeneous set of processes.

Another approach is a master slave topology in combination with a load balancing algorithm between the slaves [10]. The slaves are not separate algorithms: they are assigned a subset of the population and only compute the fitness function. Selection and evolution are performed by the master process. This fine grained approach offers a speed-up compared to a sequential GP-SR, but it does not increase the coverage of the search space.

Ting and torus topologies are also used in the literature [9]. A two-way torus topology is similar to our grid topology. The study of Nigwa et al. [9] states that sharing of messages is essential to improve convergence but the communication pattern is largely defined by the problem

domain. They conclude that diffusion is more powerful compared to partitioning.

4 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.

Benchmark problems Recent work on the convergence of GP-based SR [6, 7] 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.

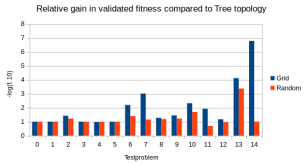
Experiment setup Each process has a population of 20 expression trees with an initial depth of 4, max depth of 8, executes 20 phases of 20 runs or generations with an archive size of 20 expressions. Per phase, the 4 best expressions are archived. The benchmark functions have at most 5 features with 20 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 2 cliques of 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. 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.

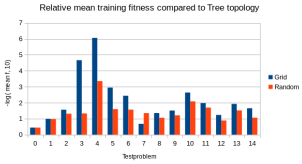
Convergence The fitness results on the training data, presented in Figure 1a, indicate that the Grid and Random topology have superior convergence characteristics compared to the Tree topology, with Grid outperforming Random on several benchmarks. When we look at the fitness values on the validation data in Figure 1b, 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.

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

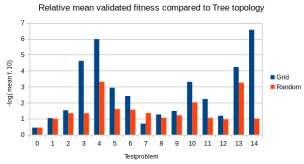
(a) Relative gain in best fitness of training data



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

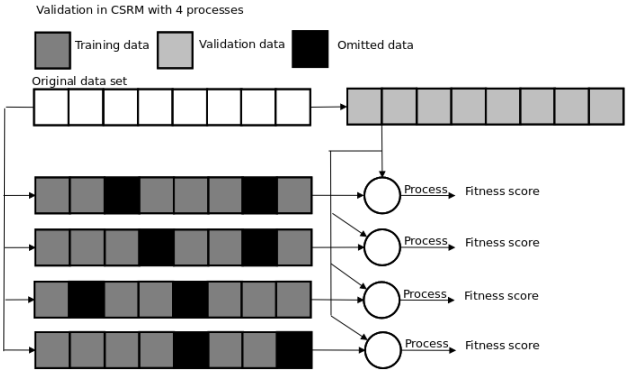


(c) Relative gain in mean fitness on training data.

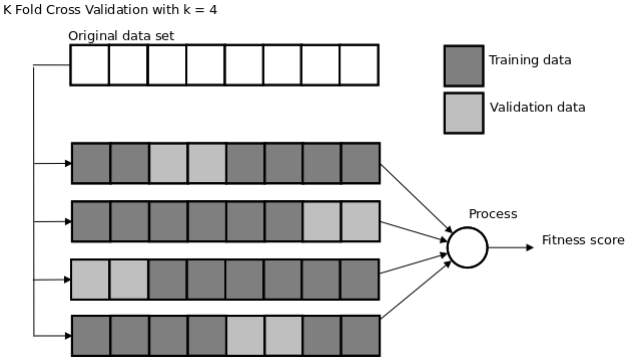


(d) Relative gain in mean fitness on full data.

Figure 1: Convergence differences between topologies.



(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.

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.

5 Use Case

Epidemiological simulation is a vital tool for policy makers since observed data is not fully predictive for new outbreaks in changing populations and experiments in the real world process can have practical, budgetary or ethical constraints. With a focus on the prevention and to obtain insights in transmission dynamics, simulations and surrogate modelling can be very useful using a configurable set of parameters mimicking real world process.

Our aim is to apply SR to the input-output mapping of a computationally intensive high performance simulator, STRIDE [12], that models the transmission of infectious diseases. The simulator is calibrated to model measles

(a) Mean synchronization delay factor.

(b) Standard deviation of synchronization delay factor.

Figure 3: Synchronization overhead introduced by topologies.

outbreaks in the city of Antwerp, Belgium, with a population size of 500000 individuals. Of vital importance here is the measles immunization aspect and as such, our research question is: how does the immunization fraction influence the outbreak of measles? We will 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 a simulator, can be useful as an emulator to improve model exploration and facilitate rapid policy making 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 are also interested in the value of an incremental approach with surrogate models based on partial results of the simulation model. This is vital in order 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 CRSE tool is able to reuse partial results as seeds for new runs. We elaborate in this section whether this approach can lead to improved models.

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 apply a Latin Hypercube Design, specifically the Audze-Eglais [4, 3, 2], which uses the Euclidean distance measure but in addition obtains a 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, 30 points in total using the tool introduced in the work of Husslange et al. [5]. We used 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]

The output parameter represents the attack rate, measured as the rate of new cases in the population at risk versus the size of the population at risk. For each parameter we obtain 30 points uniformly chosen in their range. These are then combined in the DOE. The simulator is run once for each configuration.

Symbolic regression configuration We run the experiment with a population size of 20, 30 phases with 60 generations per phase, (10,20,30) data points for 3 features, an initial depth of 3 and maximum depth of 6. The 4 best expressions of each phase are archived. We compare three approaches. First we run all simulations and then run the CSRM tool on the entire simulation dataset. This is the

traditional approach, the tool is not seeded and so starts a blind search. In the second approach we split the data into incremental sections. After 10 configurations have been simulated, we start the tool on this section. The best 4 results are saved to disk, then we run the tool with the result of 20 simulated configurations and use the results from the previous run as a seed. Finally we use the results of the 20-point dataset as a seed for the 30 point run. The cost of running the 10 and 20 point runs to use as seed for the 30 point run is similar to the cost of the 30 point run. Our last approach consists in runnning the tool on the data from 30 configurations with double the amount of phases. This means that it has approximately the same number of fitness evaluations as the 10-20-30 combination. We compare all three to see which gains are made and at what cost. We run the experiment distributed to observe the change in convergence characteristics using the seeds of the 10-20-30 combination to combine the distributed and incremental features of our tool.

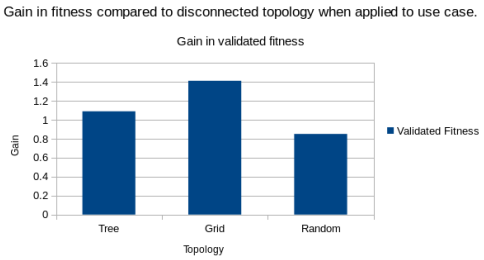
5.0.1 Fitness improvement

We compare both seeded runs and the extended run with th 30-point run. In Figure 4c we see that the fitness is improved by using the best results of the previous run on a partial data set. We have deliberately split our data set in such a way as to expose a risk here. If we run the tool with 20 datapoints seeded by a run of 10 datapoints, we see that the validated fitness actually decreases compared to a non seeded run. The ratio between new and known data is too large, leading to overfitting. If we seed the best results from the 20-point run into a 30 point configuration we see that both the training and validated fitness values significantly improve. The 30 point run with 60 phases has the same computational cost as the 10-20-30 runs combined, but gains little to nothing in convergence. We see that convergence is slowing, with training fitness improving by a factor of 10 %, but validation fitness worsens. This is a typical example of overfitting. The combined 10-20-30 run increases validation fitness with a factor of 13%

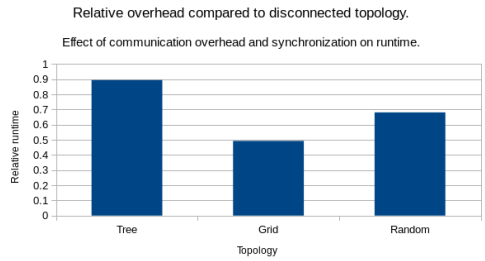
5.0.2 Distributed

Next we seed a distributed run with the results of the 10/20/30 run and compare the topologies in terms of fitness improvement and speedup. We run the same configuration as before with 25 processes, we use as seeds the 4 best expressions from the 10/20/30 run, and use Tree, Grid, Random and Disconnected topologies. In Figure 4a we compare the gain in fitness on the validation data for the tree, grid and randomstatic topologies compared to the disconnected topology. We can clearly see that the diffusion in the grid topology leads to the highest gain, followed by the tree topology. Interestingly enough, the random topology scored worse than the disconnected topology. This can occur when a local optimum is communicated early to the other processes which then dominates the remainder of the process. The effect on the runtime is measured in Figure 4b. We see that the tree topology has minimal overhead and runs nearly as fast as the disconnected topology where no synchronization or communication overhead is present. The grid topology suffers a 2x performance penalty and the random topology finds the middle ground between the two. During the experiment we observed that the processes in the tree and disconnected topologies varied as much as 4 phases. This is what we expected, in this tree topology the distance between two processes is at most 4 (depth of a 25-node binary tree). This is an important observation, if we increase the number of processes the tree topology will actually scale better. The delay tolerance allows the tree topology this scaling effect.

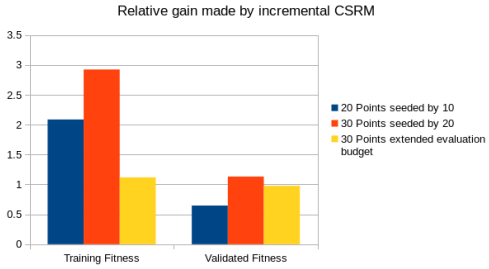
We select the best expression returned by the distributed application of CSRM with a tree topology, given its benefits in runtime and scaling. The resulting expression has a fitness value of 0.039 on the full data set. While this value is low, it is still 10 orders of magnitude removed from the optimal. This expression therefore represents an intermediate result and gives us an indication of the value partial results can offer. We use response plots for each parameter in order to isolate the effect each parameter has. We vary each of the parameters while keeping the other two constant. For the constant value we select the midpoint of the range. We then observe the effect on the attack rate. It is important to note that the range of the attack rate is [0,1]. We observe 2 important effects. First, our model produces an attack rate outside of the valid range of [0,1]. There is a scaling factor of 10 between the output of the model and the actual output data from the simulator. This is simply due to the fact that convergence is still in an intermediary phase. An important observation here is that our tool evolves the model based on 30 data points



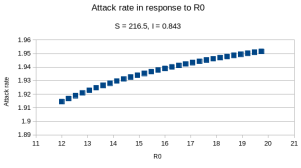
(a) Incremental distributed CSRM applied to use case.



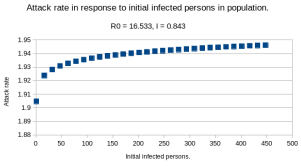
(b) Runtime impact of synchronization and communication overhead.



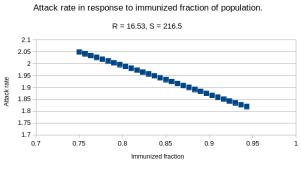
(c) Incremental fitness gain in CSRM.



(d) Response of attack rate to R.



(e) Response of attack rate to S.



(f) Response of attack rate to I.

and not the full factorial design. This means that the response plots will use the model to evaluate points that are not necessarily available to our tool to train on. Second, the trend in the response plots is in line with what we expect to see in such a surrogate model. When R0 increases the attack rate increases, which is in line with theoretical and empirical results. A similar trend is visible with the initial number of infected persons, where R0 shows a logarithmic response. Finally, as the immunization fraction increases we see a negative linear response in the attack rate. We have chosen this suboptimal surrogate model to demonstrate that while the exact values of the attack rate are not yet correctly modelled, the expected trends are. This conclusion is vital to justify our incremental approach. We can see that surrogate models will focus on matching the trend first, rather than matching individual points. This is in part due to our usage of the Pearson R correlation coefficent as a basis for the fitness function.

6 Conclusion and future work

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 (KCV) 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 optimizations 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 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] ALBA, E. *Parallel Metaheuristics: A New Class of Algorithms*. Wiley-Interscience, 2005.

[2] AUDZE, P., AND EGLAIS, V. New approach for planning out of experiments. *Problems of dynamics and strengths 35* (1977), 104–107.

[3] BATES, S., SIENZ, J., AND TOROPOV, V. 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.

[4] BATES, S. J., SIENZ, J., AND LANGLEY, D. S. Formulation of the audze–eglais uniform latin hypercube design of experiments. *Adv. Eng. Softw. 34*, 8 (June 2003), 493–506.

[5] HUSSLAGE, B. G. M., RENNEN, G., VAN DAM, E. R., AND DEN HERTOG, D. Space-filling latin hypercube designs for computer experiments. *Optimization and Engineering 12*, 4 (2011), 611–630.

[6] KORNS, M. F. *Accuracy in Symbolic Regression*. Springer, New York, 2011, pp. 129–151.

[7] KORNS, M. F. *A Baseline Symbolic Regression Algorithm*. Springer, New York, 2013, pp. 117–137.

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

[9] NIWA, T., AND IBA, H. Distributed genetic programming: Empirical study and analysis. In *Proceedings of the 1st Annual Conference on Genetic Programming* (Cambridge, MA, USA, 1996), MIT Press, pp. 339–344.

[10] OUSSAIDÈNE, M., CHOPARD, B., PICTET, O. V., AND TOMASSINI, M. Parallel genetic programming: An application to trading models evolution. In *Proceedings of the 1st Annual Conference on Genetic Programming* (Cambridge, MA, USA, 1996), MIT Press, pp. 357–362.

[11] SALHI, A., GLASER, H., AND DE ROURE, D. Parallel implementation of a genetic-programming based tool for symbolic regression. *Inf. Process. Lett. 66*, 6 (June 1998), 299–307.

[12] WILLEM, L., STIJVEN, S., TIJSKENS, E., BEUTELS, P., HENS, N., AND BROECKHOVE, J. Optimizing agent-based transmission models for infectious diseases. *BMC bioinformatics 16*, 1 (2015), 183.

[13] WILLEM, L., STIJVEN, S., VLADISLAVLEVA, E., BROECKHOVE, J., BEUTELS, P., AND HENS, N. Active learning to understand infectious disease models and improve policy making. *PLOS Computational Biology 10*, 4 (04 2014), 1–10.