Convergence of incremental distributed symbolic regression.

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

1 Abstract

Symbolic regression (SR) buils a symbolic expression for an underlying model using a set of input-output data. Amongst its advantages are the ability for a practitioner to interpret the resulting expression, determine important model features by their usage in the expression, and gain insights into the behavior of the resulting model (e.g. extrema). A key algorithms in SR is genetic programming (GP). The convergence characteristics of SR using GP are still an open issue. In this work use a distributed GP-SR implementation to evaluate the effect of communication topologies on the convergence of the algorithm. We introduce and evaluate a tree topology with the aim of finding a new balance between diffusion and concentration. We use a variation of k-fold cross validation to estimate how accurate a solution is in predicting 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 the practitioner time, as the tool can be used in parallel with the experiment generating the data. It also allows for a human expert in the loop, where the partial results of the SR tool can be used to further tune the experiment design. We validate our work on several test problems and a real world use case involving epidemiological simulation for the spread of measles.

2 Introduction and design

Our tool is implemented 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. Furthermore, Python's use of a global interpreter lock makes shared memory parallelism infeasible. Distributed programming is possible using MPI.

2.1 Algorithm

The algorithm accepts a matrix X = n x k of input data, and a vector Y = 1 x k of expected data. It will evolve expressions that result, when evaluated on X, in an 1 x k vector Y' that approximates Y. N is the number of features, or parameters, the expression can use. We do not know in advance if all features are needed, which makes the problem statement even harder. The goal of the algorithm is to find f' such that $d(f(X), f'(X)) = \epsilon$ results in ϵ minimal. 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

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

R has a range of [-1, 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 allows combining it with other objective functions. The function reflects the aim of the algorithm. 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 is not new, a variant of this approach is 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 trees. These are initialized, evolved using operators and selected to simulate evolution. The algorithm is subdivided into a set of phases. Each phase initializes the population with a seed drawn from an archive populated by previous phases or by the user. A phase is subdivided in runs, where each run selects a subset of the population and applies the GP operators. If this leads to fitness improvement, it replaces the expressions in the population. At the end of a phase the best expressions are stored in an archive to seed consecutive phases. At the end of a phases the best expressions are communicated to other processes executing the same algorithm with a differently seeded population. The next phase will then seed its population using the best of all available expressions.

We use a vanilla GP implementation, with 'full' initialization method [8]. Expressions trees are generated with a specified minimal depth. The depth of the expressions during evolution is limited by a second maximal parameter. GP differs from most optimization algorithms in this variable length representation. We use 2 operators in sequence: mutation and crossover. Mutation replaces a randomly selected subtree with a randomly generated tree. The introduction of new information leads to exploration of the search space. Crossover selects 2 trees based on fitness and swaps randomly selected subtrees between them. This exploits the search space. Selection for crossover is random, biased by fitness value. A stochastic process decides whether crossover is applied pairwise (between fitness ordered expressions in the population) or at random.

The initialization of expression trees can lead to 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. While some works opt to 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 implement an efficient bottom up approach to construct valid trees where valid subtrees are merged. In contrast to a top down approach this detects invalid expressions early on and avoids unnecessary evaluations of generated subtrees. Nevertheless, the initialization constitutes a significant computational cost in the initialization stage of each phase and in the mutation operator.

2.2 Distributed algorithm

GP allows for both fine grained and coarse grained parallelism. Parallel execution of the fitness function can lead to a speedup in runtime, but will not affect the search algorithm. Python's global interpreter lock and the resulting cost of copying expressions for evaluation makes this approach infeasible. With coarse grained parallelism one executes k instances of the algorithm in parallel. Each process has its own population of expression trees. Processes exchange their best expressions given a predefined process communication topology. This does affect the search algorithm. The topology is a key factor in determining the runtime and the convergence of the computation. Message exchange can introduce serialization and deadlock if the topology contains cycles. Our tool supports any user provided topology and must be able to deal with both issues.

Messages are expressions sent from a source process's to a target process's population. After each phase a process looks up its targets using the topology. It sends its best k expressions either by copying all expressions to each target or by spreading them over the target set. Upon completion of the sending stage, the process looks up its set of sources and waits until all source processes have sent their best expressions. To avoid deadlock a process sends its expressions asynchronously, not waiting for acknowledgement of receipt. The sent expressions are stored in a buffer for this purpose, together with an associated callable waiting object. After the sending stage the process synchronously collects messages from its sources, and executes the next phase of the algorithm. Before the next sending stage, the process will then check each callable to verify that all messages from the previous sending phase have been collected. Once this blocking call is complete, it can safely reuse the buffer and start the next sending phase. This also introduces a delay tolerance between processes. The phase runtime between processes will never be exactly identical, especially not given that the expressions have a variable length representation and differing evaluation cost. Without a delay tolerance processes would synchronize on each other, nullifying any runtime gains. With the delay tolerance a process may advance k phases ahead of a target process k steps distant 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 process using a random sample of 4/5 of the whole data. Each process effects a 4/5 split of its data between training and validation. The aggregate distributed process then approximates k-fold cross validation. Whatever the topology, each pair of communicating processes will have the same probability of overlapping data. When this probability is too low, overfitting occurs and highly fit expressions from one process will likely be invalid for another 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. In a disconnected topology, there is no diffusion of information. Each process must discover highly fit expressions independently. An edge case where this is an advantage, is when the risk of premature convergence due to local optima is high. We can try to avoid those optima by executing the algorithm in k instances, without communication. Such an approach is sometimes referred to as partitioning, as one divides the search space in k distinct partitions.

However, in general the goal is for the processes to share information in order to accelerate the search process. With a topology we introduce two concepts : concentration and diffusion. Concentration refers to processes that share little or no information and focus on their own subset of the search space. As with an overemphasis on search space exploitation, concentration can lead to overfitting and premature convergence. It is warranted when the process is nearing the global optimum. Diffusion, in this work, refers to the spread of information over communicating processes. It accelerates the optimization process of the entire group. It is not without risk, however. If diffusion happens instantly, a suboptimal solution can dominate other processes, leading to premature convergence. 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 with with root as a source and leafs as sinks with unidirectional communication, is an efficient hierarchical topology. For k processes there are k-1 communication links, reducing the messaging and synchronization overhead significantly compared to the grid topology. Diffusion can be hampered, because the information flow is unidirectional. On the other hand, with a spreading distribution policy (optimal expressions are spread over the outgoing links) a partitioning effect occurs that counteracts premature convergence. As there are no cycles, synchronization overhead is low.

Random In a random topology the convergence of the aggregated process is hard to predict. Cycles and cliques are likely, thus diffusion is not guaranteed and runtime performance differs based on the actual instance. The advantage of the random topology is that certain patterns that might occur deterministic topologies are avoided.

3 Related Work

The reference work [1] provides a broad overview of the challenges and advantages of parallel metaheuristics. An interesting parallel GP-SR implementation [11] 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 advantage of such a system is speedup, since no process ever waits on other processes. Another difference with our approach is the message exchange protocol. Whereas we exchange all messages after each generational phase, [11] interleaves message exchange with computation during a phase. Such a setup allows for a heterogeneous set of processes.

A different approach is shown in [10] 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 are not separate algorithms: they are assigned a subset of the population and only compute the fitness function. The selection and evolution are performed by the master process. This a a fine grained approach and offers a speedup compared to a sequential GP-SR implementation, but it does not increase the coverage of the search space.

In Distributed Genetic Progamming (DGP) [9] 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.

4 Experiments

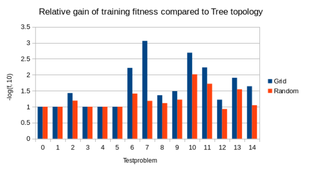
The experiments were performed on an Intel Xeon E5 2697 processor with 64GB RAM, with Ubuntu 16.04 LTS, kernel 4.4.0. CSRM is implemented using Python3, the test system uses Python 3.5. The experiments use a fixed seed in order to guarantee determinism. Where relevant the configuration is given. An open source repository holds the project's source code, benchmark scripts, analysis code and plots. Extra results left out here due to space concerns are covered in this work. The project dependencies are minimal making the project portable across any system that has a working Python3 implementation and pip as an installation manager. In order to run distributed the project requires an MPI implementation, which is available for most platforms.

Benchmark problems Recent work on the convergence of GP-based SR [6, 7] featured a set of benchmark problems that pose convergence problems for SR implementations. These 15 problems use at most five features, and are non linear arithmetic expressions using the standard set of base functions : (sin, cos, tan(h), log,a^x, /, *, modulo, abs, min, max, +, -). CSRM does not know which features are used, making the problem harder. In other words it assumes each problem is a function of 5 features which may or may not influence the expected outcome, testing the robustness of the algorithm.

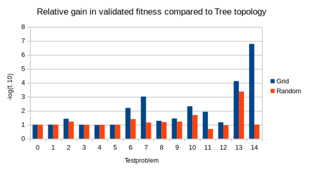
Experiment setup Each process has a population of 20, initial depth of 4, max depth of 8, executes 20 phases of 20 generations each, with an archivesize of 20. Per phase the 4 best expressions are archived, the spreading policy distributes the best expressions over the communication links. The benchmark functions have at most 5 features, each with 20 sample points in the range [1,5]. The Grid and Tree topology will have single expression per link, the Random topology will have 2 per link. 25 processes are used, resulting in respectively 400, 15 ,50 being sent per phase. The random topology in this case contains 2 cliques of cycles. When the experiment ends the best 20 expressions from all processes are collected and scored. We measure best fitness on the training data, and best fitness on the validated data. Finally we record the mean fitness values of the best 5 expressions, both on the training and validation data. 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. The fitness values fluctuate strongly between the test problems and even between topologies. We apply a negative logarithmic scale : $f_t = -\log_{10}(f)$ where f is either the best fitness value or the mean. Then we scale the results relative to the values obtained for the tree topology in order to measure relative gain or loss in orders of magnitude.

4.0.1 Results

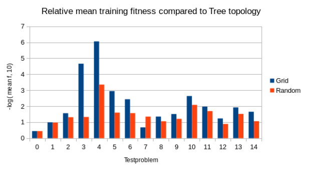
Convergence In Figure 1 we see how the topology affects convergence. The first observation we make is that the first 5 testproblems, with exception of the second, all have identical values for best fitness on training and validation data. The processes converged to zero fitness values for these problems, hence the identical results. The training fitness results 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 testproblems. When we look



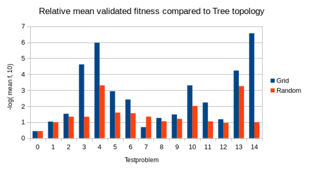
(a) Relative gain in best fitness of training data



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

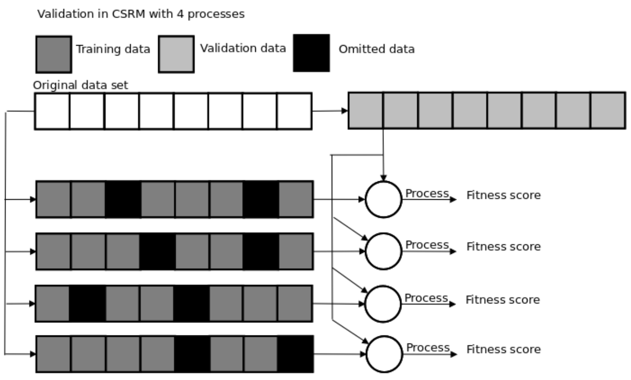


(c) Relative gain in mean fitness on training data.

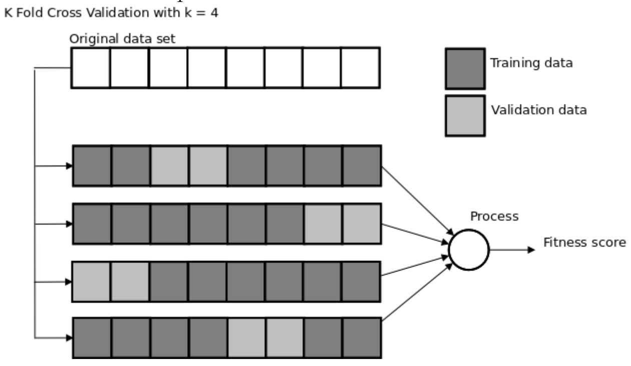


(d) Relative gain in mean fitness on full data.

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.

at the fitness values on the validation data in Figure 1b we see a more nuanced result. The grid topology is overall still the best choice, but the random topology has worse results for problems 11 and 12. Next we look at the mean fitness values on training and validation data. Interestingly enough for problem 0 both random and grid score far worse than the tree topology. Overall the grid topology is still better for most problems, with the exception of problem 7. Note the similarity in pattern in the results here between training and validation data indicating that the predictive capability of the results is still good. If overfitting would have taken place we would see a reverse pattern in the results for the validation data.

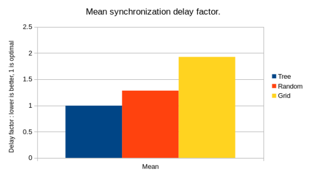
Measuring Overhead Cycles in the topology will lead to excessive synchronization and even serialization. We measure the mean execution time for testproblem 6. The convergence characteristics using the three topologies differed significantly making this a good testcase. The processes will communicate 25 times. If the runtime of a single phase is too long, the overhead of communication will become hard to measure. If it is too short the overhead dominates the entire runtime. This second case is one we should try to avoid, it will unfairly penalize topologies with cycles forcing them to serialize. The runtime of a phase is dependent on the generations, population and depth ranges of the expressions. Ideally we would like for a practitioner to choose these parameters based on the problem at hand and not be constrained by synchronization considerations. We compare the three topologies and use the disconnected or 'none' topology as a reference point, which has zero synchronization overhead. This last topology has an ideal speedup of n, where n is the processcount, compared to a sequential process. From the synchronization overhead we can then derive the speedup 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 from the other processes. In Figure 3 we see that the tree topology has a near zero delay introduced by the synchronization. This is due to the delay tolerance we have built in in our implementation. The random topology has a mean delay factor of 1.3, the grid topology scores worst with a mean delay factor of nearly 2. This is easily translated in terms of speedup. A tree topology will have near linear speedup, a grid will have a speedup roughly half of that value and a random topology will have a speedup bracketed between those two. The standard deviation for the tree topology is significantly smaller indicating that a tree topology will have a far more predictable speedup.

5 Use Case

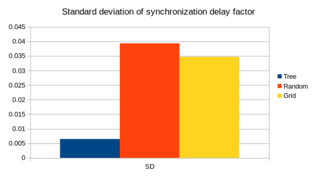
5.1 Problem statement

Our aim is to apply symbolic regression on the output of a simulator. The simulator we use for our experiment is a computationally intensive high performance tool [12] that models the spread of infectious diseases. Epidemiological simulation is a vital tool for policy makers. Simply observing the real world process is a measure of last resort at unacceptable cost in human suffering, and the resulting data is not strictly predictive for new outbreaks. A single outbreak is a sample of a very large configuration space. The focus in policy making lies on prevention and insight, and for these aims simulation and surrogate modelling are essential. A theoretical model cannot approximate within a reasonable error margin the complexity of a real world process, while simulation, with a configurable set of parameters mimicking the real world process, can. The simulator is configured to model a measles outbreak in a population of 5e5 in the city of Antwerp, Belgium. With immunization for this disease a worldwide concern we would like to obtain a surrogate model that can offer policy makers insights leading to preventative measures. Of vital importance here is the immunization aspect. Our research question for this case is : How does the immunization fraction influence the outbreak of measles? We investigate this use case within the context of this work, that is, we focus on the convergence characteristics of the process evolving the model rather than the domain specific implications of the surrogate model itself. We are interested in the value of the surrogate model at an intermediary stage in the process. How closely does this model exhibit the same trends as the underlying process? This relation is vital in order to justify our usage of partial results in the feedback loop between practitioner, simulator and regression tool. A single simulation instance is computationally expensive. We would like to construct a surrogate model that approximates the simulator. A surrogate model can offer insights into the underlying process that the resulting data cannot. Symbolic regression offers a white box model in addition to this advantage. We can use symbolic regression to obtain such a model, but in order to do so we need to obtain input and output data. Generating all simulation output in sequence leads to significant downtime for the practitioner. Using our incremental support detailed we offer the practitioner partial results during this downtime. These results can be used by a domain expert to modify the design of experiment instead of having to wait until the entire process has completed. Our tool is able to reuse the partial results as seeds for new runs. We will investigate if this approach can lead to an improved model. The value for the practitioner in this approach is twofold : incremental informative results are offered during an otherwise inactive time allowing for a feedback loop with the simulator, and a possible improvement in the final model can be obtained by seeding our regression tool.

Design of experiment We would like to have a space filling design that maximizes the sampling of the parameter space while minimizing the number of evaluation points. In this experiment we apply a Latin Hypercube Design (LHD), specifically the Audze-Eglais [4, 3, 2] (AE) LHD which uses the Euclidean distance measure but in addition obtains a uniform distribution of the individual points. The AE LHD is based on the concept of mini-



(a) Mean synchronization delay factor.



(b) Standard deviation of synchronization delay factor.

Figure 3: Synchronization overhead introduced by topologies.

mizing potential energy between design points, a measure based on the inverse of the euclidean distance.

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

Simulator configuration We construct a DOE with 3 dimensions, 30 points in total using the tool introduced in the work of [5]. The following are the parameters used:

- Basic reproduction number (R0) : the number of persons an infected person will infect, [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. [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) datapoints for 3 features, an initial depth of 3 and maximum depth of 6. The 4 best expressions of each phase are archived. We compare 3 approaches. First we run the CSRM tool on the entire dataset. This is the classical approach, the tool is not seeded and so starts a blind search. In a real world setting this would mean waiting until the simulator has run all 30 configurations. In our second approach we split the data into incremental sections. After 10 configurations have completed we start the tool on this dataset. The best 4 results are saved to disk, then we run the tool with the result of 20 configurations and use the results from the previous run as a seed. The overlap between the two datasets will influence the effects of the initialization problem. 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. To ground the comparison our last approach runs 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.

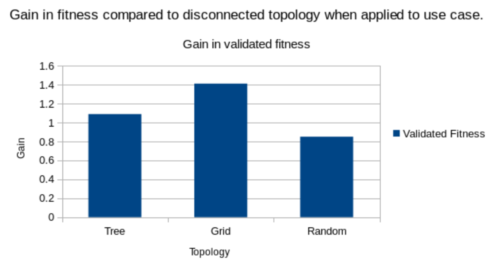
5.2 Results

5.2.1 Fitness improvement

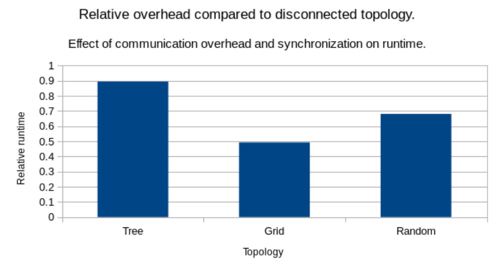
We compare both seeded runs and the extended run with the normal 30-point run. In Figure 4c we see that the fitness is improved by using the best results of the previous run on a partial data set. We have deliberately split our data set in such a was as to expose a risk here. If we run the tool with 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%

5.2.2 Distributed

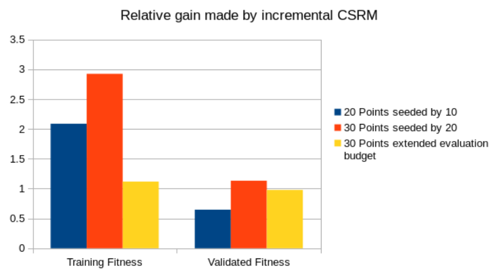
Next we seed a distributed run with the results of the 10/20/30 run and compare the topologies in terms of fitness improvement and speedup. We run the same configuration as before with 25 processes, we use as seeds the 4 best expressions from the 10/20/30 run, and use Tree, Grid, Random and Disconnected 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



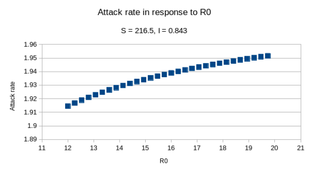
(a) Incremental distributed CSRM applied to use case.



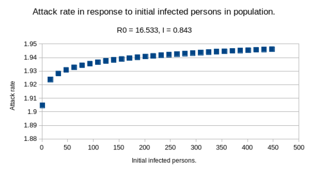
(b) Runtime impact of synchronization and communication overhead.



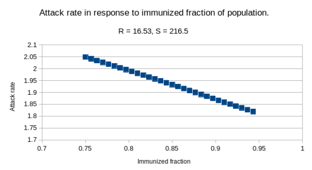
(c) Incremental fitness gain in CSRM.



(d) Response of attack rate to R.



(e) Response of attack rate to S.



(f) Response of attack rate to I.

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 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 that with a delay tolerance mechanism that mitigates load imbalance between processes. We compared three representative topologies in terms of convergence rate and speedup. The tree topology can be used as an approximation for the grid topology with near linear speedup and offers a process a delay tolerance equal to the distance between dependent processes. This feature allows the tree topology to scale better when the process set is larger. The distributed processes approximate K fold cross validation (KCV) in order to avoid overfitted solutions without the high computational cost of KCV. Our modular architecture allows it to be extended with new topologies, policies, and optimization algorithms. In the use case we demonstrated how our tool can be used to derive a surrogate model for a simulator in parallel without waiting for the simulator to complete all results. In particular we looked at the interaction between simulator and regression tool and showed that our incremental support allowed for improvements in fitness and predictive value of the final model. A feedback loop between practitioner, simulator and regression tool offers savings in time that increase with the computational cost of the simulator while yielding valuable insights that can be used during the experiment to adapt the design. The use case demonstrated that intermediate solutions are able to approximate the final model's characteristics even at a significant distance from the actual solution, 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 speedup. The tree topology achieved a near linear speedup 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.