Implementing and Testing the Performance of Parameter Fitting algorithms for Systems Biology networks

Michiel Bakker
University of Twente
P.O. Box 217, 7500AE Enschede
The Netherlands
m.a.bakker@student.utwente.nl

ABSTRACT

Recently biological research has put more focus on the mathematical and computational modelling of systems. ANIMO was developed to help the researchers with this, by allowing them to make computational models using a simple user interface. ANIMO can run simulations of biological networks which produce the same results as the real biological processes. ANIMO has a feature called *parameter fitting*, which automatically sets the parameters of a network such that the outcome of the simulation matches experimental data. Parameter fitting has been implemented in various ways in different pieces of software, but little research has been performed that compared the performance of them. This research implemented four algorithms and tested their performance, and concluded that of those four algorithms the genetic algorithm produced the best results.

Keywords

ANIMO, systems biology, s systems, parameter fitting, parameter estimation

1. INTRODUCTION

Recently, biological research has shifted in the direction of systems biology, which focuses on the mathematical and computational modelling of biological systems. ANIMO (Analysis of Networks with Interactive MOdelling) [7] was developed as a tool to model and analyse biological signalling networks. The biological network of the user is transformed to a Timed Automata model, which is analysed using the model checking tool UPPAAL [3].

A model in ANIMO (see Figure 1) has a certain topology, i.e. the set of reactants and the interactions between these reactants. Every reactant has a certain activity level that changes over time, which represents the ratio between the quantity of active and inactive forms of a protein. For instance, the phosphorylated state of a kinase is considered active. The interactions all have a certain reaction speed, in this paper referred to as the k parameter.

Initially when designing an ANIMO model the user will often guess a value for k, using a qualitative scale from 'very slow' to 'very fast'. This is done because the reaction speeds are often not known, either because the reaction is hypothetical, or the speed cannot be measured by experiment. By tweaking the k parameters of every reactant, a model can be constructed whose output of the simulation accurately matches data gathered via an experiment. However, this process is slow and error prone, since the user constantly has to guess a parameter to change, and re-simulate the updated network to check whether the model fits the data better.

25th *Twente Student Conference on IT* July 1st, 2016, Enschede, The Netherlands.

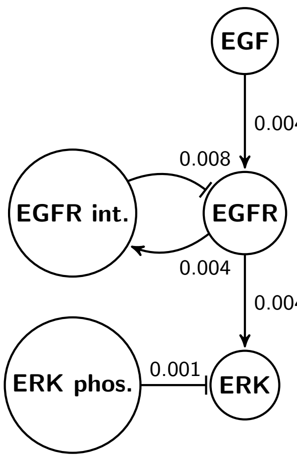

Figure 1. Example ANIMO model [5]. The numbers at the edges specify the k parameters.

In order to improve this process, ANIMO has the option to perform automated parameter fitting (PF). The user can use this when it is observed that the data of the simulation does not match experimental data, such as in Figure 2. To solve this, the user can use the parameter fitter to find k parameters that better fit the experimental data, such as in Figure 3. This feature automatically creates models with different parameters and simulates them to find the model that fits the data most accurately. Currently, two different algorithms can be used to perform the parameter fitting; either a brute-force approach, or using the Levenberg-Marquardt algorithm (LMA) [4]. The PF feature is currently only implemented in the desktop version of ANIMO, and not webANIMO [9]. This is a project which attempts to have the same functionality of ANIMO in a web site.

Section 2 and 3 will discuss necessary background information to read this paper and previously done work related to this study. The algorithms that were compared are described in Section 4. Section 5 describes the method that was used to acquire the results of this study, which are presented in section 6. The results and limitations of this research are discussed in section 7, and the research

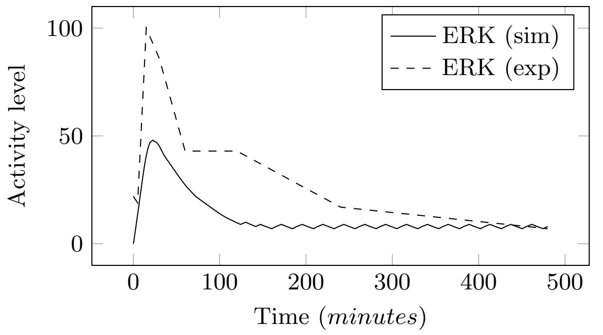

Figure 2. A mismatch between simulation and experimental data of an ANIMO model.

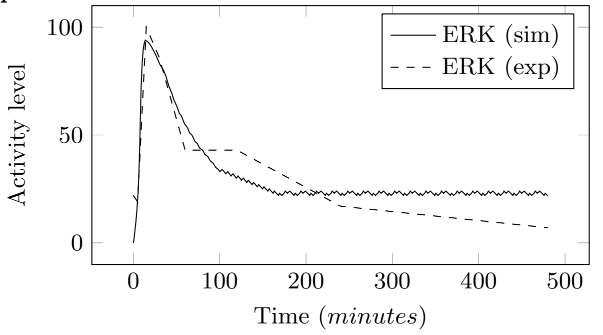

Figure 3. Simulation and experimental data of an ANIMO model after parameter fitting on all interactions.

questions are answered in section 8.

1.1 Research goal

Many algorithms can be used to perform parameter fitting [10], but it is challenging to choose which to use. Little research has been performed that compared the performance[1] of various PF algorithms with regard to biological networks. The goal of this research is to select and implement various algorithms and measure their performance. The implemented algorithms were extensively tested using benchmarks.

The results of the study will allow future developers of PF features to select the algorithm with the highest performance. Additionally, webANIMO was extended to include the PF feature to allow a greater group of users to employ the feature.

1.2 Research questions

The research was guided by the following research questions:

1. Which algorithm produces the highest quality solutions when performing parameter fitting on networks occurring in systems biology?
2. How can algorithms be combined to achieve better performance?
3. Will the algorithms converge to a local minimum, or always converge to the global minimum?
4. How is the parameter fitting feature implemented in an online environment?

[1]Note: in this paper *performance* is used to indicate the quality of solutions that an algorithm finds in the allocated time period.

2. BACKGROUND

In general, PF algorithms start with an initial solution, and iteratively attempt to find a solution that fits experimental data better. A solution is represented as array of real numbers, each representing the parameter k of an interaction in a network model. A solution that fits the data well has a high *fitness* or a low *cost*. These numbers describe how well the simulation and experimental data match. Generally, PF algorithms represent cost as the mean squared error, calculated as follows

$$C(P) = \begin{cases} \frac{1}{n*m} \sum_{i=1}^{n} \sum_{j=1}^{m} (y_{i,j} - f_{i,j}(P))^2, & \text{if } \forall p \in P, p \geq 0 \\ \infty, & \text{otherwise} \end{cases}$$

Where P is the array of parameters, n is the number of experimental time points, m the number of compared reactants, $y_{i,j}$ is experimental value of parameter j at time i, and $f_{i,j}(P)$ is the value of reactant j according to the simulation at time i with parameters P. Since negative parameters are not allowed in ANIMO, the cost is infinite if there exists any negative parameter.

Other metrics to estimate the fitness exists, but for consistency the algorithms implemented during this research will use the method above.

3. RELATED WORK

Sun et. al. reviewed various applications of metaheuristics to solve the parameter estimation problem [10]. It describes the algorithms used to perform parameter fitting in various studies, along with their advantages and disadvantages. However, the paper does not compare the performance of the algorithms that are listed. The paper was used as a guide to select algorithms to study in this research.

The parameter fitting used in ANIMO is described in [5]. The paper only describes the brute-force parameter fitter of ANIMO, and not the more advanced LMA implementation. Also, the paper mainly focuses on the usefulness and work flow of PF in ANIMO, instead of the technicalities behind it.

4. IMPLEMENTATION

Here the implementation of the four algorithms, and the implementation in webANIMO is discussed. Changes to the standard implementation of an algorithm are noted, such as how to mutate parameters, or when to stop iterating. Base knowledge of the workings of the algorithms is assumed in the description.

4.1 Levenberg-Marquadt Algorithm

A parameter fitter implementation using the LMA, a local optimisation technique, was implemented in ANIMO before the start of this research. The implementation computes the numerical Jacobian by perturbing each parameter slightly, performing a simulation, and seeing how the cost changes.

The damping parameter λ is first set to an initial value. When a solution with a lower cost is found after an iteration, λ is divided by 10. If a worse solution is found, it is increased by a factor 10, with an upper limit of 10^{15}. The algorithm stops iterating when either 50 iterations have been performed, or the difference of the new cost compared to the previous cost is lower than 10^{-6}.

4.2 Simulated Annealing

Simulated Annealing (SA) is a metaheuristic to find the global optimum for an optimization problem. SA has been used before to implement parameter estimation, among others by Gonzalez et al. [2]. For every iteration of the SA algorithm a neighbouring state is compared to the current state. The 'neighbour' of a state is chosen by randomly perturbing every parameter of the input parameters. The new parameter is calculated as follows:

$$p' = p * m^r$$

Where p' is the new parameter, p is the old parameter, m is the mutation factor, and r is an random number chosen uniformly from $[-1, 1]$. Using this method, the new parameter can become at most be m times larger or at least m times smaller, giving $p' \in [p/m, p * m]$. This is convenient because the parameters can differ by orders of magnitude, and adding or subtracting a set amount would affect small parameters more than high ones.

When the new parameters are chosen, the chance that these parameters are kept is:

$$P(c, c', T) = \begin{cases} e^{\frac{-(c'-c)}{T}}, & \text{if } c' > c \\ 1, & \text{otherwise} \end{cases}$$

Where c is the cost of the current parameters, c' is the new cost and T is the current temperature. The temperature is a value that decreases over time and affects the chance that a worse solution is accepted. The acceptance formula is chosen such that the acceptance probability for higher costs decreases as the temperature decreases. The temperature value is calculated as follows:

$$T = \left(1 - \frac{i}{n}\right) * \mathit{tempScale}$$

Where i is the current iteration index, n is the total number of allocated iterations, and *tempScale* is a parameter that describes the range of temperature values, and thus affects the acceptance chances. The algorithm continues until the total number of iterations has been reached.

With this implementation, the SA algorithm has three parameters: the mutation factor, the temperature scale, and the number of iterations. These parameters need to be tweaked in order to have the algorithm produce optimal results.

4.3 Hill Climbing

Hill Climbing (HC) is an optimization technique that is used to find a local optimum in the search space. It was chosen to include HC to study the effectiveness of an algorithm that does not necessarily converge to the global optimum. In the implementation used by this research every parameter is increased or decreased as long as the change produces a better solution. Like the SA algorithm, the parameters are multiplied or divided by the mutation factor to provide mutation that is independent on the scale of the parameters. The mutation factor decreases after an iteration of all parameters to avoid a potential 'overshoot' of the ideal parameter value. The full algorithm is described in Algorithm 1.

The algorithm has two parameters, the number of simulations, and the mutation factor, whose optimal value need to be selected.

```
$P$ is the mutable array $[P_1..P_n]$ of input parameters
while $true$ do
    $m \leftarrow 1 + (1 - \frac{i}{numSimulations}) * (mutFactor - 1)$ ;
    $currentCost \leftarrow cost(P)$;
    for $i \leftarrow 1$ to $n$ do
        $p \leftarrow P_i$ ;
        $bestValue \leftarrow p$;
        for $m' \in \{m, m^{-1}\}$ do
            for $j \leftarrow 1$ to $\infty$ do
                $p' \leftarrow p * (m')^j$;
                $P' \leftarrow P$;
                $P'_i \leftarrow p'$;
                $c \leftarrow cost(P')$;
                if max simulations reached then
                    stop algorithm;
                end
                if $c < currentCost$ then
                    $currentCost \leftarrow c$;
                    $bestValue \leftarrow p'$;
                else
                    break loop;
                end
            end
        end
        $P_i \leftarrow bestValue$;
    end
end
```

Algorithm 1: Hill Climb

4.4 Genetic Algorithm

Genetic algorithm (GA) is a metaheuristic to find the global optimum inspired by the process of natural selection. GA has been used for parameter fitting before in research done by Arisi et al. [1]. In the implementation used the initial population consist of mutations of the starting parameters. The parameters are mutated as defined by Equation 1, where $m = 8$. This gives a population whose parameters are in the same order of magnitude as the initial parameters.

$$p' = p * m^r \tag{1}$$

Where r is randomly selected from $[-1, 1]$.

Each generation, the costs of every solution in the population is calculated. This is done in parallel to achieve speed up on multi core systems. The solutions are sorted based on their costs, and the n solutions with the lowest costs are used to form the next generation. Here $n = \lfloor \mathit{selectRatio} * \mathit{populationSize} \rfloor$, where $\mathit{selectRatio} \in [0, 1]$.

Every member of the new generation is formed by recombination of two random 'parents' from the selected population. The child's parameters are selected as follows:

$$c_i = \begin{cases} p_{1,i}, & \text{if } r = 0 \\ p_{2,i}, & \text{otherwise} \end{cases}$$

Where c are the child parameters, p_1 and p_2 are the parent parameters, and r is a random variable chosen from $\{0, 1\}$. After recombination the parameters are mutated according to Equation 1. Similarly to the HC implementation, the mutation factor decreases linearly over time to achieve higher precision as the algorithm converges to an optimum:

$$m = 1 + \left(1 - \frac{i}{numGenerations}\right) * (mutFactor - 1)$$

Where i is the current generation index. The selection and recombination process repeats itself until the allocated number of generations is reached. The algorithm has four parameters that affect the performance; the population size, the number of generations, the selection ratio and the mutation factor.

The parallel simulation of the solutions was done using a thread pool with four threads. This number was chosen because it resembled the 'typical' number of cores a normal user's processor would have. The reason the number of threads was not set to the actual number of cores available was to represent realistic usage circumstances. This is because the hardware that ran the benchmarks had more than typical (16) cores.

4.5 webANIMO implementation

The functionality of the webANIMO back end was extended to include parameter fitting. The design challenge was in allowing the front end to visualize intermediate results, similar to what is seen in the desktop version of ANIMO. When performing parameter fitting in ANIMO the simulations of attempted solutions are displayed, combined with their costs. The realised implementation saves the simulation data and the cost of an attempted solution server side, if its cost is lower than the lowest cost found so far. The front end is now able to periodically poll the server for the latest intermediate results and display them.

The benefit of the method used is that the client is free to poll and visualise the intermediate results at any interval. Another possible solution would be to use a streaming-based approach, where the client and server would keep open a stream. Every time a better solution is found there would be a response sent by the server to the client describing the solution. The advantage of this approach is that the client immediately receives the data that needs to be visualised, instead of waiting for the next poll moment. This method was not chosen because it imposed more requirements for the front end, and because it was more complicated than the conventional request-response method.

5. METHOD

In this research four algorithms were compared, whose implementation is described in section 4. In a test case an algorithm will optimize parameters until a certain budget of simulations has been reached. This budget was selected such that one run of all algorithms take an equal amount of time. This way there is a constant allocated time budget in which the algorithms attempt to find a solution with the least cost.

To simulate actual usage circumstances, models of real biological processes were used in the test cases. The model to be fitted is a mutated version of the starting model, where a subset of the parameters are scaled up or down by a factor up to 8. The 'experimental' data that the test model is fitted against are a selection of points from a simulation of the (non-mutated) starting model. This way there is a guarantee that an optimal solution with zero cost exists.

With this model and fitting data the algorithm is run. Every time the algorithm performs a simulation the cost of the best solution so far, and the time since the start of the run is stored. For each run there are two results that are of interest: the cost of the best solution, and the solution time. An assumption was made that most algorithms converge to an optimum after some time, and do not find a (significantly) better solution. This behaviour can be seen in Figure 4. The solution time is the time it took for the algorithm to find the best solution. It is defined as the time it took for an algorithm to find a solution with a cost that was at most 5% higher than the lowest cost found. This is to avoid having very small changes to the cost found affect the solution time.

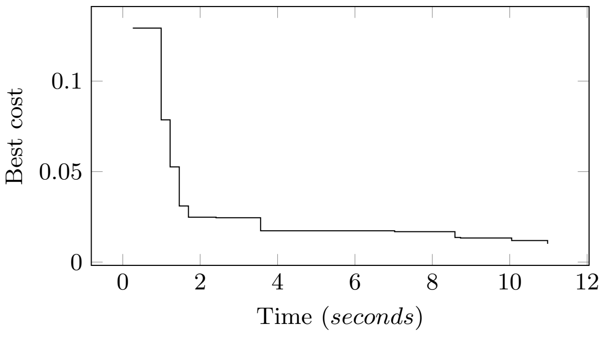

Figure 4. Best cost found over time by the HC algorithm optimizing 10 parameters of the chondrocyte model.

To facilitate performing the experiments, a test suite was developed, as specified in Algorithm 2. Per test case there is a set of test models (combined with the number of parameters that are optimized) and a set of algorithm configurations. An algorithm configuration specifies which algorithm should be used, and what parameters this algorithm should have. The test suite benchmarks an algorithm for every combination of test model and algorithm configuration.

```
for $m \in models$ do
    $expData = simulation(m)$;
    $params$ = random subset of model parameters;
    mutate $params$;
    for $a \in algorithms$ do
        $run(a, m, params, expData)$;
        log best solution costs and solution time
    end
end
```

Algorithm 2: Test suite

In order to make a fair comparison of all algorithms, the optimal configuration of algorithm parameters needs to be found. Therefore for each algorithm test cases were made with various configurations of parameters. The configurations per algorithm that were tested are specified in Section 6. The algorithm configuration with the best results will be used in the comparison across all algorithms.

Three biological models were used in this research:

- A simple 5 node network modelling part of the MAPK pathway [5] (seen in Figure 1)
- A 16 node network modelling a chondrocyte cell [8]
- A 52 node network modelling crosstalk between the TNF and EGF pathways [6]

Both the parameter selection and algorithm comparison tests use same set of models. Every model is used 20 times, giving 60 optimisation cases per algorithm configuration. The chondrocyte and TNF EGF model configurations require 1 to 20 parameters to be optimized. The MAPK model only has 5 interactions, so it cycles the number of optimized parameters from 1 to 5, in total 4 times.

All benchmarks were performed on a cluster of Dell PowerEdge M610 servers, each with 2 Intel Xeon E5520 processors, giving a total of 16 cores.

6. RESULTS

First, the results of the experiments attempting to find the best parameters per algorithm are presented. The test suite was run four times per algorithm, giving a total of $4 * 60 = 240$ runs per algorithm configuration. The values whose runs gave the lowest average cost over all runs were chosen as the best parameters. If that was not conclusive, the lowest average solution time was used for selection. Then, all algorithms are compared using the best parameters found based on those results.

For LMA, different values for the λ parameter were compared. The tested values were 10^{-3}, 10^{-4} and 10^{-5}. Based on the results found in Table 1, the best value for the λ parameter is 10^{-3}.

The SA implementation has two parameters, the mutation factor and the temperature scale. For the mutation factor the tested values were 1.1, 1.5 and 2.0, and for the temperature scale 0.1, 0.5 and 1.0. All combinations of these parameters were tested, given a total of 9 configurations, whose results are found in Table 2. The best values found were 2.0 for the mutation factor, and 0.1 for the temperature scale.

The HC implementation only has one parameter, the mutation factor. Tested values were 1.05, 1.1, 1.2, 1.5 and 2.0, giving the results found in Table 3. The runs where the mutation factor was 2.0 gave the lowest average cost, so this value was selected.

For GA, there are four parameters: the selection ratio, mutation factor, population size and number of generations. Because the amount of combinations rises exponentially with the number of parameters not all parameters could be tested thoroughly. The tested values for the selection ratio were 0.25 and 0.125, and the mutation factor was 1.25 for all tests. The values for the population size and number of generations were chosen such that their product was constant, making the total number of simulations constant across the tests. Their values were 100x5, 50x10, 25x20 and 10x50, giving a total of 8 configurations for this benchmark, of which the results can be found in Table 4. The configuration that gave the lowest average cost was a selection ratio of 0.25, population size of 50, and 10 generations.

Using the parameter configurations found above, the benchmarks comparing all algorithms were performed. The test suite was performed 15 times, giving $15 * 60 = 900$ runs per algorithm. The results of these runs, combined with whether the algorithm converges to a global or local minimum, can be found in Table 5. For the TNF EGF model the number of parameters optimized is plotted against the average solution costs in Figure 5.

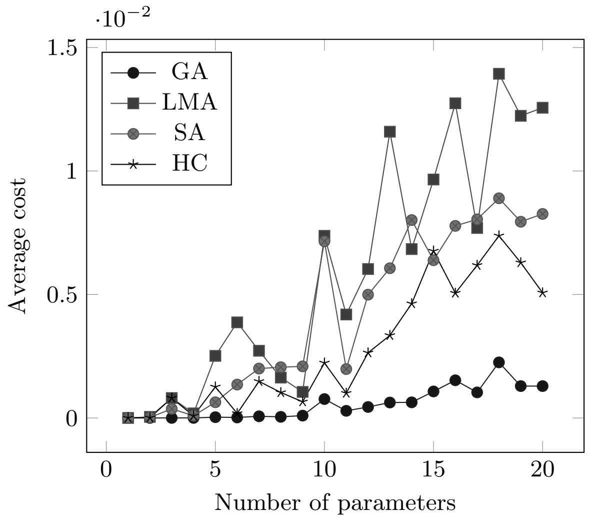

Figure 5. Comparison between the number of parameters to optimize, and the average cost found, using the TNF EGF model.

7. DISCUSSION

It can be seen in Table 5 that the GA implementation found the best solutions on average. The better performance of GA compared to the other algorithms can probably be attributed to the fact that GA allows for parallelism, and that it converges to the global minimum. The average solution time of GA is higher than SA or HC. This means that in a use case where fast convergence is more important than the quality of the solution SA or HC may be preferred.

Figure 5 plots the number of parameters that were optimized against the average cost found. This data also shows that GA produces the best solutions, having near optimal (cost ≈ 0) solutions when the number of parameters is below 10. It can be seen that all algorithms produce worse solutions as the number of parameters rises. This can be explained by the fact that the search space grows as the number of parameters grows, and that therefore more solutions need to be tried.

It is believed that the executed method is a valid way of assessing the performance of the selected algorithms. However, because of the limited number of tested algorithms, it cannot be said that GA is the best possible approach. Further research can be performed using a similar method but implementing additional algorithms, in an effort to find an even better PF solution. For instance, more metaheuristics, hybrid strategies or local optimisation algorithms may be of interest.

Also, the number of models used for the benchmarks in this research is rather low. It could be the case that the algorithms compared are only optimized for the models used here, and that they perform significantly worse on other models. Therefore further research could be performed that use a larger set of models in their benchmarks. This could also discover correlations between the performance of certain algorithms and the size or topology of the model.

With the method used in this study research question 2 cannot be answered. Future research could attempt to answer this. This could be achieved by running one algorithm for half of the allocated time, and continuing with another. Another possibility could be to include some pre-processing algorithm, for instance an algorithm that prunes a part of the network that is not needed for calculating cost.

8. CONCLUSION

From the results found in this research it may be concluded

Table 1. LMA benchmark results

λ parameter	10^{-5}	10^{-4}	10^{-3}
Average best cost	0.106	0.065	0.049
Average solution time	31.554	32.015	43.855

Table 2. SA benchmark results

Mutation factor	1.1	1.5	2.0	1.1	1.5	2.0	1.1	1.5	2.0
Temperature scale	0.1			0.5			1.0		
Average best cost	0.0863	0.0300	0.0288	0.0987	0.0451	0.0346	0.1029	0.0522	0.0446
Average solution time	16.0141	15.8935	14.1066	11.3849	13.8098	12.2283	10.1661	12.7718	10.2504

Table 3. HC benchmark results

Mutation factor	2.0	1.5	1.2	1.1	1.05
Average best cost	0.0260	0.0336	0.0494	0.0676	0.0875
Average solution time	17.6804	17.5548	17.0401	16.1080	14.4709

Table 4. GA benchmark results

Selection ratio	0.25	0.125	0.25	0.125	0.25	0.125	0.25	0.125
Population size	100		50		25		10	
Number of generations	5		10		20		50	
Average best cost	0.00323	0.00255	0.00237	0.00239	0.00360	0.00336	0.00543	0.01038
Average solution time	39.0758	39.9916	39.9655	39.6461	36.2205	36.1320	35.6279	32.7343

Table 5. All algorithms benchmark results

Algorithm	GA	LMA	SA	HC
Average best cost	0.0028	0.0372	0.0258	0.0222
Average solution time	40.2459	46.1136	14.4923	18.3721
Global minimum	Yes	No	Yes	No

that the answer to research question 1 is the Genetic Algorithm. The average cost found in the experiments was over a factor 7.5 lower than the next best average cost. This makes it by far the most suitable algorithm for the circumstances that were used in this research.

The answer to research question 3 can be found in Table 5. GA converges to the global optimum and is the highest performing algorithm tested. However, it cannot be concluded that convergence to the global optimum always leads to a higher performance, since the SA implementation performed worse than the HC, which always converges to a local optimum.

An answer to research question 4 was found by implementing the parameter fitting functionality into the back end of webANIMO. This allows for the inclusion of PF in the webANIMO front end in future work. To allow for the same user experience in webANIMO as in ANIMO for the desktop intermediate results can be collected by the front end. The solution used for this was a polling-based approach, keeping the requirements on the client side low.

The work done and results found in this research will be of use to others performing research related to parameter fitting. By performing extensive benchmarks under various circumstances this research was able to thoroughly compare the performance of the tested algorithms. The work done should improve the experience of ANIMO users by allowing them to use more powerful PF algorithms, both in the web and desktop versions of ANIMO.

9. REFERENCES

[1] I. Arisi, A. Cattaneo, and V. Rosato. Parameter estimate of signal transduction pathways. *BMC neuroscience*, 7(1):1, 2006.

[2] O. R. Gonzalez, C. Küper, K. Jung, P. C. Naval, and E. Mendoza. Parameter estimation using simulated annealing for s-system models of biochemical networks. *Bioinformatics*, 23(4):480–486, 2007.

[3] K. G. Larsen, P. Pettersson, and W. Yi. Uppaal in a nutshell. *International Journal on Software Tools for Technology Transfer (STTT)*, 1(1):134–152, 1997.

[4] K. Levenberg. A method for the solution of certain non–linear problems in least squares. 1944.

[5] S. Schivo, J. Scholma, M. Karperien, J. N. Post, J. van de Pol, and R. Langerak. Setting parameters for biological models with animo. *arXiv preprint arXiv:1404.0444*, 2014.

[6] S. Schivo, J. Scholma, P. E. van der Vet, M. Karperien, J. N. Post, J. C. van de Pol, and R. Langerak. Modelling with animo: between fuzzy logic and differential equations. *BMC Systems Biology*, 2016.

[7] S. Schivo, J. Scholma, B. Wanders, R. A. Urquidi Camacho, P. E. van der Vet, H. B. J. Karperien, R. Langerak, J. C. van de Pol, and J. N. Post. Modelling biological pathway dynamics with timed automata. *IEEE Journal of Biomedical and Health Informatics*, 18(3):832–839, May 2014.

[8] J. Scholma, S. Schivo, R. A. Urquidi Camacho, J. C. van de Pol, H. B. J. Karperien, and J. N. Post. Biological networks 101: computational modeling for molecular biologists. *Gene*, 533(1):379–384, January 2014.

[9] W. Siers, M. Bakker, B. Rubbens, R. Haasjes, J. Brandt, and S. Schivo. webanimo: Improving the accessibility of animo. *submitted to F1000Research*, 2016.

[10] J. Sun, J. M. Garibaldi, and C. Hodgman. Parameter estimation using metaheuristics in systems biology: A comprehensive review. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 9(1):185–202, Jan 2012.