Improving Parallel Efficiency of Sequential Approximate Bayesian Computation Schemes

Felipe Reck

Born 19th January 1997 in Starnberg, Germany

3rd May 2021

Master's Thesis Mathematics

Advisor: Prof. Dr. Jan Hasenauer

Second Advisor: Prof. Dr. Alexander Effland

Life & Medical Sciences Institute (LIMES)

Mathematisch-Naturwissenschaftliche Fakultät der

Rheinischen Friedrich-Wilhelms-Universität Bonn

Contents

Abstract

Approximate Bayesian Computation (ABC) is a tool used to analyze posterior distributions of parameters in mathematical models. It is based on Bayesian statistics and yet does not require detailed knowledge about the likelihood function, making ABC a commonly used method for complex stochastic models, which are frequently encountered in biological sciences and for which often no other approaches are feasible. As analyses are computationally expensive, parallelization on high-performance infrastructure is necessary, which existing parallelization strategies are however not yet able to fully exploit due to working with self-contained generations.

We present an unbiased, wall time minimizing parallelization strategy for sequential Monte Carlo samplers (ABC-SMC). It aims to fully employ all available computational resource at any given time by formulating tentative tasks for the next generation as soon as resources in the current generation become idle. We formulate an algorithm using the new parallelization strategy and prove its correctness. Additionally, we show that the algorithm can be combined with different enhancements previously developed to improve performance of ABC-SMC algorithms, such as self-tuning by adaptively selecting thresholds, population sizes and distance functions, which is essential for applications. The algorithm is then tested on various models, ranging from simple artificially constructed test problems to contemporary real life application examples. These tests demonstrate that we are able to achieve an average acceleration of around 10% to 20% in realistic settings when compared to established approaches, while obtaining equally accurate results.

1 Introduction

From the fields of biology over physics up to studies of social constructs, it usually is easy enough to come up with some model that can plausibly describe most observed phenomena [Bea10]. In most cases, simulating data for that model given its parameters is also fairly straightforward.

The difficulties that exist when dealing with such models mostly stem from its dependence on non-observable parameters, so parameters that can not be directly inferred based on the data. However, unlike the simulation of new data for given parameters, determining parameter values that could have generated some given observed data set is generally difficult to realize. Nonetheless, this inverse problem, the parameter inference, is often required to make statements about different properties. Thus, parameter inference is usually approached in a Bayesian setting, which allows formulation of a parameter posterior distribution combining prior information on the latent parameters with the likelihood of observing data. Using sampling methods, one can have a closer look at the posterior and therefore enable parameter estimation with global uncertainty analysis [RC04]. The pressing issue is however, that, as models are non-deterministic and become more complex, the likelihood function quickly becomes computationally infeasible to evaluate [Tav+97; Jag+17]. This problem is especially present for agent-based models, as in that setting the likelihoods are rarely tractable and there is insufficient common structure to transfer methods tailored to one application to others [ST10; Jag+17; Iml+19].

Approximate Bayesian Computation (ABC) is a likelihood-free method applicable in such cases [SFB18]. It circumvents evaluating the likelihood by simulating data from the model and comparing it to the observed data via a distance metric, generating a population of samples drawn from an approximation of the posterior. It has become popular especially in ecology and epidemiology [Ton+09; Bea10], due to its simplicity, however, it is also broadly applicable to any other field in which mathematical modeling is desirable.

ABC returns asymptotically correct results, but to reach the point where one can speak of asymptotic properties, ABC relies on repeating simulations thousands to millions of times. Therefore, the development has led to methods which can more efficiently explore the parameter space [SFB18]. Most prominent among these methods is ABC-SMC, in which ABC is combined with a sequential Monte-Carlo scheme (SMC), which draws importance samples from an increasingly refined posterior approximation over several generations. This allows to successively reduce acceptance thresholds while maintaining high acceptance rates [MDJ06; SFT07].

In recent years, various adjustments have already been made to improve ABC-SMC methods and to deal with different issues occurring when working with real life data [FBS11; Pra17; SH20]. Nonetheless, there are still

huge amounts of repeated simulations necessary. Since simulations within a generation are however independent of each other, ABC-SMC allows for the parallelization of tasks within a generation, enabling the use of high-performance clusters (HPC). This is also strictly necessary, since, for complex models, ABC-SMC methods are still far too time intensive to be run without parallelized execution.

To efficiently work with HPCs, strategies to evenly distribute the workload over all available workers are required. Most established ABC implementation use *static scheduling* [Dut+17; Kan+16], which already minimizes the active computation time and the consumed energy. However, as only as many tasks are defined as particles are required, it does lead to a very sub-optimal wall time. *Dynamic scheduling* [KRH18] mitigates this problem and reduces the overall wall time by continuing sampling on all workers until sufficiently many particles have been accepted. But, as in a standard SMC scheme the generations are self-contained, also in this approach workers become idle at the end of each generation. This is particularly disadvantageous on common HPC infrastructure, where computation time can only be allocated per node, consisting of typically a few dozen cores each. Thus, while waiting for the remaining workers to finish, a subset of cores is not used at all, inducing, besides a prolonged wall time, a sub-optimal use of computational and financial resources.

Here we present a novel ABC-SMC parallelization approach for multi-core systems, called *look-ahead scheduling*. It builds upon dynamic scheduling, but in addition preemptively at the end of each generation, as workers become idle, formulates preliminary sampling tasks for the next generation. These preliminary tasks do not require knowledge about the generations final population but are instead based only on currently available results. Look-ahead scheduling already distributes those tasks to idle workers before the current generation has finished, thus making use of all available workers at almost all times. We show that by proper weighting of the preliminarily sampled particles we obtain an unbiased sample. Using different application examples we demonstrate accuracy and compare efficiency to established approaches. We also provide an implementation applicable on HPC infrastructure.

2 Basics of Approximate Bayesian Computation

In this section we first introduce some mathematical concepts which are necessary for further understanding. We then present the ideas behind ABC and a standard ABC-SMC formulation.

2.1 Bayesian Statistics

The idea behind Bayesian statistics originated in a theorem from 1763 by Thomas Bayes [BP63] that later became known as Bayes' theorem. It describes the relation between the conditional probabilities of the two measurable sets (events) A and B:

$$p(A|B) = \frac{p(B|A)p(A)}{p(B)} \tag{1}$$

What we want to estimate is the probability of some parameter θ to have a specific value, given some observation in form of the observed data y_{obs}. So we want to compute the so called posterior $p(\theta|y_{obs})$ which, by Bayes' theorem, is given by

$$p(\theta|y_{obs}) = \frac{p(y_{obs}|\theta)p(\theta)}{p(y_{obs})}. \tag{2}$$

This formula consists of several parts:

- $p(\theta)$: the *prior*; some initial suggestion for the distribution of θ. This enables the integration of prior knowledge about the parameters into the estimation. Most commonly the prior is chosen to be some multivariate normal or uniform distribution.
- $p(y_{obs}|\theta)$: the *likelihood*; the probability of an observation to match our data y_{obs} given that the model has the fixed parameter θ.
- $p(\theta|y_{obs})$: the *posterior*; the probability distribution of the parameter θ given that data y_{obs} was observed. As mentioned above, this is the distribution we want to estimate as well as possible.
- $p(y_{obs}) = \int p(y_{obs}|\theta)p(\theta)$: the marginal probability of the data. This is basically just a normalization constant. Most common methods do not require knowledge about this constant, so usually one can assume that $p(y_{obs}) = 1$ without loss of generality.

As a simple example, one can take a look at some observed data y_{obs} describing the growth of a given population. One might assume that the population size N behaves for example according to the simplified Verhulst model of population dynamics [Ver38], i.e. our model is $\frac{d}{dt}N = rN(1 - \frac{N}{K})$.

In this case, the two unknown parameters that we would like to estimate are the maximum growth rate r and the carrying capacity K, so we want to find $p((r, K)|y_{obs})$. All parameters are usually notation-wise summarized by one multidimensional parameter $\theta \in \mathbb{R}^{n_\theta}$, i.e. here $\theta = (r, K) \in \mathbb{R}^2$.

In many cases it is computationally very demanding and even close to impossible to generally assess and calculate the likelihood $p(y_{obs}|\theta)$. However, it is often feasible to sample from it for a given fixed θ, which is why Approximate Bayesian Computation was developed.

For a more thorough introduction into Bayesian statistics see e.g. [Lee12].

2.2 Importance Sampling

Whenever it is difficult to sample from some density p, it can be beneficial to draw from an approximation q for p, with $q(x) \neq 0$ p-almost everywhere. By reweighting, we can still correct for the error of sampling from a different distribution afterwards. This is called importance sampling. It is based on the idea that for any test function $f : \mathbb{R}^{n_\theta} \to \mathbb{R}$ it holds that

$$\begin{aligned}\mathbb{E}_p[f(x)] &= \int f(x)p(x)\,dx = \int f(x)\frac{p(x)}{q(x)}q(x)\,dx \\ &= \mathbb{E}_q[f(x)\frac{p(x)}{q(x)}]. \end{aligned} \tag{3}$$

If N i.i.d. random variables $(X_i)_{i=1}^N$ with $X_i \sim q$ can be obtained, we can also approximate $\mathbb{E}_p[f(x)]$ following the idea behind Monte-Carlo methods [MU49; KTB11]

$$\frac{1}{N}\sum_{i=1}^{N} f(X_i)\frac{p(X_i)}{q(X_i)} \overset{N\to\infty}{\longrightarrow} \mathbb{E}_p[f(x)] \text{ a.s.} \tag{4}$$

The rescaling that is performed by multiplying with the weight function $\bar{w}(x) = p(x)/q(x)$ is necessary to correct for the fact that the samples are drawn from the "wrong" distribution q instead of the desired distribution p.

As p, q are two measures and $q \gg p$, $p(x)/q(x)$ corresponds to the Radon-Nikodym derivative of p with respect to q and is in this context also called the likelihood ratio / importance weight $\bar{w}$.

For more detailed information on importance sampling see e.g. [Sri02].

2.3 Approximate Bayesian Computation

Approximate Bayesian Computation is used to sample from the posterior distribution without evaluating the likelihood function.

Given some observed data y_{obs} and a prior $\pi(\theta)$, we want to obtain the posterior $p(\theta|y_{obs}) \propto p(y_{obs}|\theta)\pi(\theta)$ without requiring knowledge about the likelihood $p(y_{obs}|\theta)$.

The core idea is to use a proposal distribution $g(\theta)$ to create a set of parameters, with each of which the model can return data, that, if not equal, is at least sufficiently similar to the observation y_{obs}. This parameter population is then weighted, using the importance weights, turning it into a sample from an approximation of the posterior.

In the simplest ABC approach the proposal distribution is equal to the prior $g(\theta) = \pi(\theta)$. This approach consists of only three basic steps and works as follows:

1. Sample $\theta^i \sim \pi(\theta)$
2. Simulate $y \sim p(y|\theta^i)$
3. Accept θ^i if $d(y, y_{obs}) \leq \varepsilon$, else restart

By first sampling θ^i according to the prior, $\pi(\theta)$ a candidate is suggested. Then, some artificial data y is generated for the candidate θ^i according to the likelihood $p(y|\theta^i)$ at the specific point given by the candidate θ^i. In the third step, the candidate θ^i is accepted if the distance between the simulated and the observed data $d(y, y_{obs})$ is smaller than some value ε. Here, d is a distance metric and the acceptance threshold is given by some fixed $\varepsilon \in \mathbb{R}_+$.

For the second step, it is assumed that we can sample from the likelihood although it cannot be explicitly computed. While still not trivial, this form of sampling is normally possible without further complications [Ton+09]. However, this simulation step is the one which usually consumes the vast majority of the time, especially for more complex models.

In order to obtain a population consisting of several particles (i.e. parameters), the three steps of the ABC algorithm are repeatedly executed.

We now want to show that, if we assign each particle θ^i its importance weight $\pi(\theta^i)/g(\theta^i)$, the set of accepted particles $(\theta^i)_{i=1}^N$ constitutes a sample from an approximation of the posterior $\pi_{ABC,\varepsilon}(\theta|y_{obs})$. If the proposal is equal to the prior as in the above formulation, these weights are equal to 1, however, as we will use different proposal distributions in the following sections, we will prove the statement for a general proposal $g(\theta) \gg p(\theta|y_{obs})$ (see also [SFB18; Ton+09]).

Theorem 1. *The accepted particles θ^i constitute a sample from*

$$\pi_{ABC,\varepsilon}(\theta|y_{obs}) \propto \int_{\mathbb{R}^{n_y}} \mathbb{I}_{(d(y,y_{obs})\leq\varepsilon)} \pi(y|\theta) dy \cdot \pi(\theta) \tag{5}$$

and

$$\pi_{ABC,\varepsilon}(\theta|y_{obs}) \xrightarrow{\varepsilon\to 0} p(\theta|y_{obs}). \tag{6}$$

Proof. The procedure of first sampling from the proposal distribution $g(\theta)$, then generating data y from the likelihood $p(y|\theta)$ and at last accepting only if

$d(y, y_{obs}) \leq \varepsilon$, is equivalent to sampling a pair (θ, y) from a joint distribution proportional to

$$\mathbb{I}_{(d(y,y_{obs})\leq\varepsilon)} p(y|\theta) g(\theta). \tag{7}$$

If the sample is then assigned its importance weight $\pi(\theta)/g(\theta)$, the steps correspond to sampling from a joint distribution proportional to

$$\mathbb{I}_{(d(y,y_{obs})\leq\varepsilon)} p(y|\theta) g(\theta) \frac{\pi(\theta)}{g(\theta)} = \mathbb{I}_{(d(y,y_{obs})\leq\varepsilon)} p(y|\theta)\pi(\theta). \tag{8}$$

The fact that the distributions are always only proportional and not equal to the expression stems from the joint density being equal to zero whenever $d(y, y_{obs}) > \varepsilon$. So to turn the expression into a probability measure, the weights need to be scaled using an (unknown) normalization constant c in order to reach a total weight of 1.

As we are only interested in the distribution of θ, we take a look at the θ-marginal of our joint distribution, denoted by $\pi_{ABC,\varepsilon}(\theta|y_{obs})$, which is proportional to the marginal of Equation 8

$$\int_{\mathbb{R}^{n_y}} \mathbb{I}_{(d(y,y_{obs})\leq\varepsilon)} \pi(y|\theta)\pi(\theta) dy. \tag{9}$$

As $\varepsilon \to 0$, we then get that

$$\begin{aligned} \lim_{\varepsilon\to 0} \int_{\mathbb{R}^{n_y}} \mathbb{I}_{(d(y,y_{obs})\leq\varepsilon)} \pi(y|\theta) dy \cdot \pi(\theta) &\propto \lim_{\varepsilon\to 0} \pi_{ABC,\varepsilon}(\theta|y_{obs}) \\ &= \int_{\mathbb{R}^{n_y}} \delta_{y_{obs}} \pi(y|\theta) dy \cdot \pi(\theta) \\ &= p(y_{obs}|\theta)\pi(\theta), \end{aligned} \tag{10}$$

where $\delta_{y_{obs}}$ is the Dirac delta. At last, the final expression $p(y_{obs}|\theta)\pi(\theta)$ is equivalent to the posterior $p(\theta|y_{obs})$ by Bayes' theorem (Equation 1). □

2.4 Sequential Monte-Carlo

The previous simple formulation of an ABC algorithm, also called Rejection ABC, tends to have impossibly low acceptance rates if ε is small, resulting in a strong trade-off between run time and accuracy of the result. Thus, ABC is frequently combined with a Sequential Monte Carlo scheme (*ABC-SMC*) [Ton+09; Bea10; SFT07].

Generally speaking, a sequential Monte-Carlo (*SMC*) method is a tool to obtain knowledge about the state of a Markov process (i.e. the state of our population) which is affected by random perturbations and for which only partial observations are available. SMC starts with a set of particles which

is updated over several generations to obtain increasingly accurate representations of the posterior distribution. The method can be interpreted as a mean-field particle version of Feynman-Kac probability measures [MM00].

Applied to our ABC method, this means that in ABC-SMC we consider several generations $t = 1, \ldots, n_t$, with decreasing acceptance thresholds $\varepsilon_t > \varepsilon_{t+1}$. In each generation t we use the three steps seen in Section 2.3 multiple times to generate a weighted population $P_t = \{(\theta_t^i, w_t^i)\}_{i \leq N}$ consisting of N particles $(\theta_t^i)_{i \leq N}$ to which weights $(w_t^i)_{i \leq N}$ will be assigned, as explained below. These populations $(P_t)_{t=1}^{n_t}$ constitute samples of successively better approximations $\pi_{ABC,\varepsilon_t}(\theta|y_{obs})$ of the posterior.

In the first generation ($t = 1$), particles are sampled directly from the prior, $\theta \sim g_1(\theta) = \pi(\theta)$ and all particles are assigned a uniform weight, so $w_i = 1 \ \forall i \in [N]$ (or $1/N$ for normalized weights).

In later generations ($t > 1$), candidates θ^* are sampled from a proposal distribution $g_t(\theta) \gg \pi(\theta)$ which is formed using the last generations accepted weighted population P_{t-1} and some perturbation kernel K_t (i.e. g_t is a kernel density estimate of the last generation). Conceptually, the idea is to select an accepted parameter from the last generation and perturb it, in which case g_t takes the form

$$g_t(\theta) = \frac{\sum_{i=1}^N w_{t-1}^i K_t(\theta|\theta_{t-1}^i)}{\sum_{i=1}^N w_{t-1}^i}, \tag{11}$$

where the kernel K_t is e.g. a normal distribution with mean θ_{t-1}^i and covariance matrix Σ_{t-1}, so $K(\theta|\theta_{t-1}^i) = \mathcal{N}(\theta|\theta_{t-1}^i, \Sigma_{t-1})$. However, in general the proposal distribution can take many forms.

For each candidate θ^*, data $y \sim p(y|\theta^*)$ is generated and if the distance between the simulated and the observed data $d(y, y_{obs})$ falls below the generations threshold ε_t, θ^* is accepted. The weights w_t^i assigned to each accepted particle θ_t^i are proportional to the importance weights, which correspond to the Radon-Nikodym derivatives $\bar{w}_t(\theta) = \pi(\theta)/g_t(\theta)$. As the importance weights are only correct up to some (unknown) normalization constant, as discussed in the proof of Theorem 1, they need to be scaled using a self-normalizing estimate (see Section 4.1 for details)

$$w_t^i = \frac{\bar{w}(\theta_t^i)}{\sum_{j=1}^N \bar{w}(\theta_t^j)} = \frac{1}{\sum_{j=1}^N \frac{\pi(\theta_t^j)}{g_t(\theta_t^j))}} \frac{\pi(\theta_t^i)}{g_t(\theta_t^i)}. \tag{12}$$

Following the same arguments as in Theorem 1, this is precisely such that the weighted parameters are samples from the distribution

$$\begin{aligned} &\int_{\mathbb{R}^{n_y}} w_t(\theta) \mathbb{I}_{(d(y,y_{obs}) \leq \varepsilon_t)} \pi(y|\theta)\, dy \cdot g_t(\theta) \\ &= \int_{\mathbb{R}^{n_y}} \mathbb{I}_{(d(y,y_{obs}) \leq \varepsilon_t)} \pi(y|\theta)\, dy \cdot \pi(\theta), \end{aligned} \tag{13}$$

i.e. the target distribution (5) for $\varepsilon = \varepsilon_t$.

A functional ABC-SMC algorithm is formulated in Algorithm 1.

Algorithm 1: Standard ABC-SMC

IN: prior π, data y_{obs}, particles per generation N,
    number of generations n_t, perturbation kernels $(K_t)_{t=1}^T$,
    thresholds $\varepsilon_1 > ... > \varepsilon_T$
Sample $(\theta_1^i)_{i=1}^N \sim \pi(\theta)$
Assign weights $w_1^i = 1/N \ \forall i \in [N]$
for $t = 2 : n_t$ **do**
    compute proposal $g_{t-1}(\theta) = \frac{\sum_{i=1}^N w_{t-1}^i K_t(\theta|\theta_{t-1}^i)}{\sum_{i=1}^N w_{t-1}^i}$
    $i = 0$
    while $i < N$ **do**
        Sample $\theta^* \sim g_{t-1}$
        Simulate $y \sim p(y|\theta^*)$
        if $d(y, D) < \varepsilon_t$ **then**
            $i++$
            $\theta_t^i = \theta^*$
            Assign weight $\bar{w}_t^i = \frac{\pi(\theta)}{g_{t-1}(\theta)}$
        end
    end
    Normalize weights $w_t^i = \frac{\bar{w}_t^i}{\sum_{j=1}^N \bar{w}_t^j}$
end
RETURN: $(\theta_{n_t}^i, w_{n_t}^i)_{i=1}^N$

The output of ABC-SMC is a population of weighted parameters, which constitutes a sample from an approximation of the posterior distribution, as seen in Theorem 1. The quality of this approximation improves, as the final acceptance threshold ε_{n_t} decreases.

$$P_{n_t} = \{(\theta_{n_t}^i, w_{n_t}^i)\}_{i \leq N} \sim \pi_{ABC,\varepsilon_{n_t}}(\theta|y_{obs}). \tag{14}$$

For any test function $f : \mathbb{R}^{n_\theta} \to \mathbb{R}$, the expected value under the posterior is then approximated via the *importance estimator*

$$\mathbb{E}_{\pi_{ABC,\varepsilon_{n_t}}(\theta|y_{obs})}[f] \approx \hat{f} = \frac{1}{N} \sum_{i=1}^N w_{n_t}^i f(\theta_{n_t}^i). \tag{15}$$

Components of an ABC-SMC algorithm (e.g. the population size N or the acceptance threshold ε_t for each generation) do not necessarily have to be fixed a-priori. Instead, it is possible to adapt these parts during

the algorithm based on the state of a population in order to e.g. improve acceptance rates. This is further discussed in Section 4.3.

2.5 Effective Sample Size

As the output of an ABC-SMC method is a set of weighted particles, it can happen that a few of these particles make up almost all of the weight. In such a case, most further candidates will be drawn based on these "heavy" particles, therefore they become the only effective ones. To quantify the strength of this effect, one can compute the effective sample size (ESS) [Owe13], which is defined as

$$ESS((w_i)_{i=1}^N) = \frac{(\sum_{i=1}^N w_i)^2}{\sum_{i=1}^N (w_i)^2}. \tag{16}$$

If the weight is perfectly distributed (i.e. uniform), the effective sample size of the population will be equal to

$$\frac{N \cdot (1/N)}{(N \cdot (1/N^2))} = N, \tag{17}$$

whereas a single particle carrying the full weight results in an ESS of $1/1 = 1$.

A reliable result should have a sufficiently large effective sample size. The order of magnitude of an acceptable value depends on uncertainties related to population size [KH17] and can thus vary strongly between different models. Whenever the ESS is too small, one should consider using a larger population size N.

3 Parallelization on High-Performance Clusters

While asymptotically exact, ABC relies on repeated simulation of data, performing the basic steps presented in Section 2.3 often thousands to millions of times. Therefore, applications are often only realistically possible on large scale infrastructure such as high performance clusters.

When assessing the performance of some strategy for parallel workload distribution over multiple workers, we distinguish between wall time and CPU time. The *wall time* is the real elapsed time between start and end of a task. In contrast, the *CPU time* measures the total time any core spent actively working on the task. If there are W workers available over which the workload can be distributed, the wall time should equal CPU time divided by W in an optimal scenario. In reality the two sides will differ significantly if only some fraction of the available cores is kept busy.

The average of the fraction of workers effectively working during the entire process is called the *parallel efficiency*.

$$\text{parallel efficiency} = \frac{\text{wall time on one worker}}{W \cdot \text{wall time on W workers}} \approx \frac{\text{CPU time}}{W \cdot \text{wall time}}. \quad (18)$$

When working on an HPC, it is crucial to employ strategies to properly distribute the workload over all available cores in order to maximize the parallel efficiency, i.e. minimize the wall time.

In this section we present the two most common approaches used to parallelize ABC schemes and explain some of their flaws. Motivated by these flaws, we present the new look-ahead scheduling, and formulate a working algorithm.

3.1 Established Parallelization Strategies

Static scheduling (STAT) is employed by most established ABC-SMC implementations [Kan+16; Dut+17]. In STAT, given a population size N, exactly N tasks are defined and distributed over the workers. Each task consists of sampling until one particle gets accepted (Figure 1A). The tasks are queued if N is larger than the amount of workers W. STAT aims to minimize the CPU time and is easy to implement, as it only requires basic pooling routines, which are available in many distributed computation frameworks. However, here even for $W > N$ only N workers are employed, although the number of required simulations is usually substantially larger than N. In addition, at the end of every generation the number of active workers decreases successively, with eventually most workers idly waiting for a few to finish their tasks.

Dynamic scheduling (DYN) [KRH18] already greatly improves wall time, as there parameter sampling is performed continuously on all available workers. This is achieved by splitting up tasks to only contain one single simulation instead of demanding an acceptance. Thus, the amount of tasks is

not fixed at the beginning but instead new tasks are continuously created until N particles have been accepted (Figure 1B). However, one issue is that we cannot simply use the first N accepted particles as the final population since that would bias the population towards parameters with short-running simulations. Instead, we have to keep track of the start time of each simulation and wait for the workers to finish all simulations for the candidates that began before the start time of the last accepted particle. Only then we can form the population of accepted particles for that generation by selecting the N particles with earliest starting times out of the $\tilde{N} \geq N$ total acceptances.

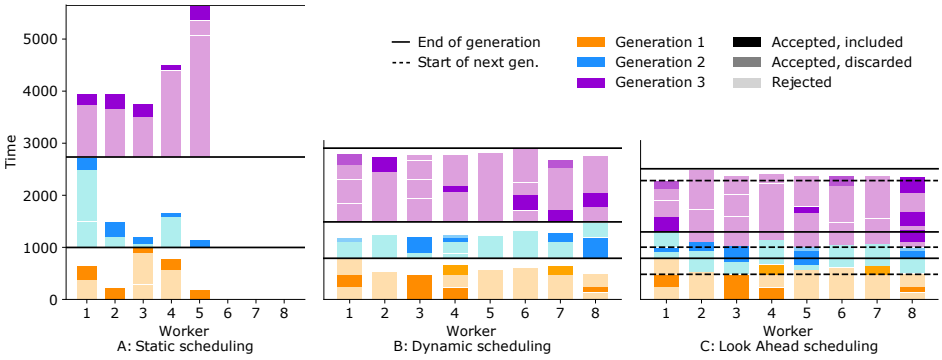


Figure 1: Different scheduling approaches to the same process

Note that as we possibly end up with some additional accepted particles and thus additional performed simulations when compared to STAT, this means that the CPU time of DYN is at least slightly higher than in STAT. This, however, is basically always well worth the wall time reduction, especially on common HPC infrastructure, where computation time can only be allocated per node, consisting of typically a few dozen cores each. So while in DYN some additional computation might be performed on some of the cores, those cores would simply remain unused and yet unavailable to other tasks in STAT.

3.2 Parallelization using Look-Ahead Scheduling

Dynamic scheduling already allows to exploit the available parallel infrastructure to a higher degree than static scheduling and therefore substantially decreases the wall time (see [KRH18] for an extended comparison). Nonetheless, there is still a considerable part of the workers idle as they wait for the remaining simulations to finish at the end of each generation. This fraction of time that the workers in average remain idle easily reaches 10% and more (see Section 5). It increases as the number of workers increases relatively to the population size, potentially even getting close 100%. Additionally, it may increase if the simulation times are heterogeneous, which

is often the case in a realistic setting, e.g. with estimated reaction rates determining the number of events that need to be simulated.

We propose to extend dynamic scheduling by using the free workers at the end of a generation to proactively sample for the next generation: As soon as N acceptances have been reached in generation $t-1$, and thus workers start to become idle, we define a preliminary population of accepted particles $P'_{t-1} = \{(\theta'^i_{t-1}, w'^i_{t-1})\}_{i \leq N}$ based on the first N acceptances. This preliminary population P'_{t-1} is used to construct a preliminary proposal distribution g'_t. New tasks can then be created and distributed to the free workers, which consist of generating candidates θ'^* based on the preliminary proposal distribution g'_t and executing the respective simulation.

The assessment of the acceptance of a candidate depends on the characteristics of the sampling procedure. If the acceptance criteria are chosen a-priori, all acceptances can be checked by the workers directly after the simulation. However, in order to enable adaptive components of the algorithm while maintaining common acceptance criteria across all particles of one generation, we will have to delay the evaluation for particles sampled from the preliminary (see Section 4.3).

As soon as all simulations for generation $t-1$ have finished and thus P_{t-1} is available, the final proposal distribution g_t is computed and all further tasks are again based on the actual proposal g_t.

By sorting the acceptances by their starting time and selecting the first N, the populations are corrected for run time bias as in DYN. At the end, the weights of particles based different proposals need to be normalized separately, as derived in Section 4.1.

The population for generation t is then given as

$$P_t = \{\{(\theta'^i_t, w'^i_t)\}_{i \leq N_1}, \{(\theta^i_t, w^i_t)\}_{N_1 < i \leq N}\}, \tag{19}$$

with $0 \leq N_1 \leq N$, where N_1 particles are based on the preliminary proposal density g'_t and $N - N_1$ on the final one g_t.

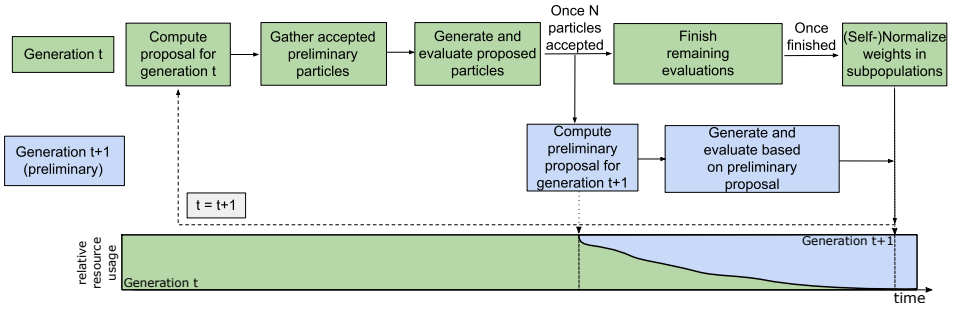


Figure 2: Conceptual visualization of the look-ahead approach and its resource assignment across generations

We call this parallelization strategy *look-ahead scheduling (LA)*, as it already looks ahead to generation t during generation $t-1$.

3.3 Algorithm

Given some function *create_proposal*, which takes the weighted population P_t and some kernel K_t as input and returns a proposal distribution, e.g. of the form

$$g_t(\theta) = \frac{\sum_{i=1}^{N} w_{t-1}^i K_t(\theta|\theta_{t-1}^i)}{\sum_{i=1}^{N} w_{t-1}^i}, \tag{20}$$

a non-adaptive look-ahead algorithm works as shown in Algorithm 2.

In order to understand the algorithm, it is important to note that one single worker always performs the sampling, the simulation and the distance calculations (function *worker_evaluate_one*, line 30-37) for one candidate consecutively, and that the simulation step (l.33) consumes a vast majority of the total run time. This means that, if we have W workers, W instances of the *while*-loop (l.10) will be running in parallel.

The proposal used for the calculations within one instance is passed to the worker at the function call (l.12), and therefore cannot change in the meantime. However, different concurrently active instances can be using different proposals (the preliminary or the final) or even be performing simulations meant for different generations. The latter happens when one worker returns the N-th acceptance for generation $t-1$; then, the other workers will finish their computations for generation $t-1$, but any new task assigned to a worker will be an instance of the while loop for generation t using the preliminary proposal.

Everything not in the *worker_evaluate_one* function, as for example the proposal creation and also anything else that requires knowledge about more than one particle, is not performed on a worker. Instead, there is one parent process that is connected to all workers which is in charge of these tasks. Furthermore, this parent process does itself not perform any of the time intensive simulations, but is continuously preparing and distributing the tasks, and retrieving their results [Jag+17].

If the algorithm has some of the previously mentioned adaptive components (for details see Section 4.3), the worker tasks based on the preliminary do not contain the acceptance check. Instead, the simulation results will be stored and the parent process will, as soon as the final proposal is available, start retrieving this information to go over the acceptance criteria for these particles (see Figure 2).

Algorithm 2: ABC-SMC algorithm using LA scheduling

IN: prior π, data y_{obs}, number of generations n_t,
    particles per gen. N, perturbation kernels $(K_t)_{t=1}^{T}$,
    acceptance thresholds $\varepsilon_1 > ... > \varepsilon_{n_t}$
Sample $(\theta_1^i)_{i=1}^N \sim \pi(\theta)$
Set weights $w_1^i = 1 \; \forall i \in [N]$
$g_1 = \text{create\_proposal}((\theta_1^i, w_1^i)_{i=1}^N, K_1)$
for $t = 2 : n_t$ **do**
    $i = 0$ # accepted particle counter
    $j = 0$ # start time counter
    while $i < N$ **do**
        $j++$
        if worker_evaluate_one$(g_{t-1}, \varepsilon_t, y_{obs})$ **is** *True* **then**
            $i++$
            start_time$[i] = j$
            $\theta_t^i = \theta^*$
            Set weight $\bar{w}_t^i = \frac{\pi(\theta_t^i)}{g_{t-1}(\theta_t^i)}$
            if $i = N$ **then**
                g_t=create_proposal$((\theta_t^i, \bar{w}_t^i)_{i=1}^N, K_t)$
                # this is the **preliminary** proposal
                $\rightarrow$ **generation** $t+1$ **starts here**
            end
        end
    end
    Sort θ_t^i by start_time
    Self normalize $(\bar{w}_t^i)_{i=1}^N$ for each sub-population to obtain $(w_t^i)_{i=1}^N$
    # see Section 4.1
    $\rightarrow$**generation** t **ends here**
    $g_t = \text{create\_proposal}((\theta_t^i, w_t^i)_{i=1}^N, K_t)$
    # this is the **final** proposal
end
RETURN: $g_T = (\theta_T^i, w_T^i)_{i=1}^N$

function *worker_evaluate_one*
IN: proposal g, threshold ε, observed data y_{obs}
Sample $\theta^* \sim g(\theta)$
Simulate $y \sim p(y|\theta^*)$
if $d(y, y_{obs}) < \varepsilon$ **then**
    RETURN **True**
end
RETURN **False**

3.4 Implementation

We implemented the look-ahead scheduling strategy in the framework available through pyABC. pyABC is an open-source python package providing an ABC-SMC implementation, including various state-of-the-art subroutines. Through working implementations employing STAT and DYN scheduling, it was already capable of running an ABC-SMC scheme on an HPC. The version of pyABC used for this thesis can be found in the attachments, although the look-ahead scheduling is by now also available in the maintained version at `https://github.com/ICB-DCM/pyABC`.

To handle the task distribution, we employ a Redis low-latency server. Redis (Remote Dictionary Server) is an open source (BSD licensed), in-memory data structure store, used as a database, cache, and message broker (`https://redis.io/`).

Before the sampling is started, a dedicated (Redis-)server is set up which must not be shut down until the parameter inference is finished. To this server one can then connect the desired amount of workers, which will usually also run for the entire duration. However, it is also possible to add/remove workers during the process of parameter inference as outside circumstances require.

We set up a (Redis-)pipeline on the server which acts as a queue to which instances of the task at hand are added, so in our case one work packet consisting of sampling a candidate and simulating data. Whenever a worker frees up, it returns the results of its last computations to the server and then retrieves and starts work on a new task from the queue. This pipeline is governed by different counters, e.g. on the number of accepted particles and started simulations, which are managed by the parent process, i.e. the Redis-server. Based on these counters, the server prepares the new work packets (e.g. by computing a new proposal) and ensures the queue always contains enough tasks to keep the workers busy.

4 Theoretical Properties

While STAT and DYN both use only a single proposal distribution, LA contains an actual difference in how sampling is performed, as a part $N_1 \in [0, N]$ of the population is drawn from a preliminary proposal distribution. To distinguish particles and weights that are based on the preliminary from ones based on the final proposal, we denote the preliminary ones by $({\theta'}_t^i, {w'}_t^i)_{i \leq N_1}$ and the final ones by $(\theta_t^i, w_t^i)_{N_1 < i \leq N}$.

This chapter explains why an additional normalization step is required and how we deal with the occurring non-normalized weights. Afterwards the unbiasedness of our approach is proven, and how to enable the use of adaptive components in the algorithm is detailed.

4.1 Self-Normalizing Weights

As mentioned in Section 2.3 and 2.4, we are usually dealing with distribution which are not equal, but only proportional to the target distribution. This is mainly due to the fact that assigning each accepted particle θ_t^i its corresponding importance weight $\pi(\theta_t^i)/g_t(\theta_t^i)$ is only correct up to an unknown normalization constant. If there is only one proposal distribution, this does not pose any further problems as we can simply divide all weights by the same estimate of the normalization constant c, given by the sum over all weights $\sum_{j=1}^{N} \pi(\theta_t^j)/g_t(\theta_t^j)$, as also done in Algorithm 1.

However, if there are multiple different proposals, each one has its own normalization constant.

As discussed in Theorem 1, these constants originate in immediately rejecting any data y with $d(y, y_{obs}) > \varepsilon$, i.e. in setting the weight of a region of the joint space of (θ, y) to zero. This results in a total weight, which can not be explicitly determined, but which can be smaller than one, therefore requiring a normalization by an unknown constant c. The constant c depends on the original weight of the the region that is now rejected, which in turn depends on the proposal $g(\theta)$ as that is the distribution θ is sampled from during the ABC steps.

So, instead of the desired normalized weights for the particles sampled from the preliminary w' and the ones for particles from the final proposal w, we only know the unnormalized version $\bar{w}' = c'w'$ and $\bar{w} = cw$, where $c, c' > 0$ are two potentially different constants. Hence, it is necessary to apply the normalization to each sub-population individually, since we can not summarize particles from different proposals as long as their normalization does not align.

Nonetheless, using self-normalized importance sampling [Owe13], we can show that by proper handling of each subgroup of weights, we can still obtain a sample from $\pi_{ABC,\varepsilon}(\theta|y_{obs})$.

First, we define an estimate for a single proposal and prove its convergence.

Let $(w(\theta^i) = 1/c \cdot \bar{w}(\theta^i))_i$ be the weights of a population of particles $(\theta^i)_i$ sampled from the same proposal g_t. Here $\bar{w}(\theta^i)$ are the importance weights given by the Radon-Nikodym derivatives of the prior π w.r.t. the proposal g_t and $1/c$ is the respective (unknown) normalization constant with $c > 0$.

Theorem 2. *Let $\hat{f}_s$ denote the self-normalized importance sampling estimate of a test function $f : \mathbb{R}^{n_\theta} \to \mathbb{R}$, defined as*

$$\hat{f}_s = \frac{\sum_{i=1}^{N} w(\theta^i) f(\theta^i)}{\sum_{i=1}^{N} w(\theta^i)}. \tag{21}$$

Then

$$\hat{f}_s \overset{N\to\infty}{\longrightarrow} \int f(x)\pi(x)dx \ \textit{a.s.} \tag{22}$$

Proof. First note that

$$\begin{aligned}\hat{f}_s &= \frac{\sum_{i=1}^{N} w(\theta^i) f(\theta^i)}{\sum_{i=1}^{N} w(\theta^i)} = \frac{c}{c} \frac{\frac{1}{N}\sum_{i=1}^{N} \bar{w}(\theta^i) f(\theta^i)}{\frac{1}{N}\sum_{i=1}^{N} \bar{w}(\theta^i)} \\ &= \frac{\frac{1}{N}\sum_{i=1}^{N} \frac{\pi(\theta^i)}{g(\theta^i)} f(\theta^i)}{\frac{1}{N}\sum_{i=1}^{N} \frac{\pi(\theta^i)}{g(\theta^i)}}.\end{aligned} \tag{23}$$

Note that the normalization constant cancels itself out and thus we do not need to know its value. Nominator and denominator now have the shape of the standard importance sampling estimate $\frac{1}{N}\sum_{i=1}^{N} f(X_i)\frac{p(X_i)}{q(X_i)}$, for which the almost sure convergence to $\mathbb{E}_p[f(x)]$ is known, as seen in Equation (4). Thus, we know that

$$\frac{1}{N}\sum_{i=1}^{N} \bar{w}(\theta^i) f(\theta^i) \to \mathbb{E}_\pi[f(x)] = \int f(x)\pi(x)dx \text{ a.s.} \tag{24}$$

$$\frac{1}{N}\sum_{i=1}^{N} \bar{w}(\theta^i) \to \mathbb{E}_\pi[1] = 1 \text{ a.s.} \tag{25}$$

from which the claim immediately follows. □

As the normalization constant cancels itself out, the estimator $\hat{f}_s$ is independent of c. Therefore, we use this self-normalization on each sub-population individually to obtain the normalized weights

$$w'^i_t = \frac{\bar{w}'^i_t}{\sum_{j=1}^{N_1} \bar{w}'^j_t} \tag{26}$$

$$w^i_t = \frac{\bar{w}^i_t}{\sum_{j=N_1+1}^{N} \bar{w}^j_t}. \tag{27}$$

To enable the merging of particles into one population, even if they were sampled from different proposals, we now only need to make sure the total weight remains equal to one by multiplying the weights of one sub-population by some constant α and the weights of the other by its inverse $1-\alpha$. This will lead to an unbiased sample, as shown in the following section.

As α only determines how much weight we want to place on the respective sub-populations, it can be chosen as any value $\alpha \in [0,1]$. A natural choice is for example $\alpha = N_1/N$, so weighting the contribution of each sub-population by the number of samples generated from the respective proposal. However, since the normalization only happens at the very end of a generation, it is for example also possible to choose α in a way that maximizes the effective sample size of the population. This is further discussed in Section 6.

4.2 Unbiasedness

A key characteristic of ABC-SMC methods is that they provide an asymptotically unbiased Monte-Carlo sample from $\pi_{\mathrm{ABC},\varepsilon_{n_t}}(\theta|y_{obs})$ as $N \to \infty$, which converges to the target distribution $\pi(\theta|y_{obs})$ for $\varepsilon \to 0$ (see Theorem 1). Here, we verify that the sample $P_t = \{(\theta'^i_t, w'^i_t)_{i\leq N_1}, (\theta^i_t, w^i_t)_{N_1<i\leq N}\}$ obtained through an algorithm employing LA scheduling conserves this property.

Theorem 3. *Assume that the weights $(w'^i_t)_{i=1}^{N_1}$, $(w^i_t)_{i=N_1+1}^{N}$ are self-normalized within their sub-population. Then, for any test function $f : \mathbb{R}^{n_\theta} \to \mathbb{R}$ it holds that*

$$\hat{f} = \frac{1}{N}\left[\sum_{i=1}^{N_1} w'^i_{n_t} f(\theta'^i_{n_t}) + \sum_{i=N_1+1}^{N} w^i_{n_t} f(\theta^i_{n_t})\right] \overset{N\to\infty}{\longrightarrow} \mathbb{E}_{\pi_{\mathrm{ABC},\varepsilon_{n_t}}(\theta|y_{obs})}[f]. \tag{28}$$

Proof. The main idea is that each sub-population gives an asymptotically unbiased estimator on its own, even if the proposal is based on a preliminary population that potentially consists of particles with shorter

simulation times. This is because the weights before the normalization $c'w'_t(\theta') = \bar{w}'_t(\theta') = \pi(\theta')/g'_t(\theta')$, $cw_t(\theta) = \bar{w}_t(\theta) = \pi(\theta)/g_t(\theta)$ are exactly the Radon-Nikodym derivatives w.r.t. the respective proposal distributions. So, after normalization within the respective sub-population, we individually obtain the target distribution in Equation 5 for both proposals.

So, for the importance estimator $\hat{f}$ it holds that

$$\begin{aligned}
\hat{f} &= \left[\sum_{i=1}^{N_1} \alpha {w'}^i_{n_t} f({\theta'}^i_{n_t}) + \sum_{i=N_1+1}^{N} (1-\alpha) w^i_{n_t} f(\theta^i_{n_t}) \right] \\
&= \alpha \frac{c'}{c'} \frac{\sum_{i=1}^{N_1} \bar{w}'(\theta^i) f(\theta^i)}{\sum_{i=1}^{N_1} \bar{w}'(\theta^i)} + (1-\alpha) \frac{c}{c} \frac{\sum_{i=N_1+1}^{N} \bar{w}(\theta^i) f(\theta^i)}{\sum_{i=N_1+1}^{N} \bar{w}(\theta_i)} \\
&= \alpha \frac{\frac{1}{N_1} \sum_{i=1}^{N_1} \bar{w}'(\theta^i) f(\theta^i)}{\frac{1}{N_1} \sum_{i=1}^{N_1} \bar{w}'(\theta^i)} + (1-\alpha) \frac{\frac{1}{N-N_1} \sum_{i=N_1+1}^{N} \bar{w}(\theta^i) f(\theta^i)}{\frac{1}{N-N_1} \sum_{i=N_1+1}^{N} \bar{w}(\theta^i)} \\
&\overset{N\to\infty}{\longrightarrow} \alpha \frac{\mathbb{E}_{g'(\theta)}[\bar{w}'(\theta) f(\theta)]}{\mathbb{E}_{g'(\theta)}[\bar{w}'(\theta)]} + (1-\alpha) \frac{\mathbb{E}_{g(\theta)}[\bar{w}(\theta) f(\theta)]}{\mathbb{E}_{g(\theta)}[\bar{w}(\theta)]} \\
&= \alpha \mathbb{E}_{g'(\theta)}[\bar{w}'(\theta) f(\theta)] + (1-\alpha) \mathbb{E}_{g(\theta)}[\bar{w}(\theta) f(\theta)] \\
&= \alpha \mathbb{E}_{g'(\theta)}[\frac{\pi(\theta)}{g'_t(\theta)} f(\theta)] + (1-\alpha) \mathbb{E}_{g(\theta)}[\frac{\pi(\theta)}{g_t(\theta)} f(\theta)] \\
&= \mathbb{E}_{\pi_{\mathrm{ABC},\varepsilon}(\theta|y_{obs})}[f]. \qquad (29)
\end{aligned}$$

The potentially biased preliminary population is only used to construct a proposal distribution $g'_t(\theta)$, but not to form the final Monte-Carlo sample from the target distribution $\pi_{\mathrm{ABC},\varepsilon_{n_t}(\theta|y_{obs})}$. For $N \to \infty$ a proposal density can be arbitrary, subject to $g'_t(\theta) \gg \pi(\theta)$. □

To assure that all particles target the same distribution,

$$\pi_{\mathrm{ABC},\varepsilon_{n_t}}(\theta|y_{obs}) \propto \int \mathbb{I}_{(d(y,y_{obs})\leq\varepsilon_{n_t})} \pi(y|\theta)\, dy \cdot \pi(\theta), \qquad (30)$$

it is necessary that the acceptance of particles is always based on the same final acceptance criteria, regardless of whether they were sampled from g'_t or g_t.

One problem that can arise is that, if the proposal designates less weight to certain regions in parameter space, this can lead to an imbalance for a finite population size N. In theory, the weighting of preliminary particles does account for this, while practically the sample may be more likely to be deteriorated, s.t. in general a sufficiently large population size should be employed. This phenomenon is further analyzed in test model (T3).

Note, that the above statements and the concept of LA can without loss of generality be applied to multiple preliminary proposal distributions. This might occur for example when the proposal is updated whenever a

new particle is accepted in hopes of improving the acceptance rate, which motivates one of the enhancements discussed in Section 6.

4.3 Adaptivity

It is crucial for any real life application that an implementation can choose and update some parts of the ABC-SMC algorithm by itself, adapting to its current situation.

For example, in applications it is often hard to predict what acceptance threshold one should choose for each generation, in order to maintain a decent acceptance rate while still making as much progress towards a small error as possible. Thus, the acceptance threshold is commonly chosen as a q-quantile of the previous accepted distances

$$\varepsilon_t = \min \left\{ \varepsilon > 0 \,\middle|\, \frac{1}{N} \sum_{i=1}^{N} \mathbb{I}_{\{d(y_i^{t-1}, y_{obs}) < \varepsilon\}} \geq q \right\}, \; q \in (0,1). \tag{31}$$

ABC-SMC can in general be integrated with certain adaptive algorithms developed in ABC, e.g. to learn distance functions accounting for heterogeneity of data types [Pra17], to adjust population sizes to the problem [KH17], to reformulate ABC as an exact inference method customizing the acceptance step [SH20], and to work on a lower-dimensional representation of the data via essential information extracting summary statistics [FP12]. These additions are practically relevant, as they make the analysis more robust, reduce the required number of simulations, and replace time-consuming manual tuning.

In pyABC several such improvements are already implemented, however, as we start sampling for the next generation before the previous one finishes when using LA scheduling, adaptive components are still not known when they are usually needed. As the acceptance criteria have to be equal across all particles to target the same distribution, we can also not generate a preliminary version of the adaptive parts.

To enable our concept to still take advantage of these strategies, we do not determine acceptance of a candidate right away, if the desired information is not yet available. Instead, the evaluation must be delayed until the final criteria are fixed, i.e. until after the full population P_{t-1} is available. This requires that the workers do not simply return their acceptances to the parent process, but instead store the results of the simulation at least temporarily until generation $t-1$ is finished. While previously not necessary, this was usually already done at least partially, in order to be able to create different statistics and diagnostics for individual inference runs.

Only once the final P_{t-1} is available, those results are retrieved and evaluated using the adapted components, thus requiring some additional computations by the parent process. Since this evaluation only requires a

simple step of calculating the distance using the updated distance metric and comparing that value with the new epsilon, the added workload is trivial, even when compared to a single simulation.

5 Test Models

To test the new concept, it was used to perform parameter inference on various different models. First, we used LA on some basic toy models (T1-T3) to examine the general behavior and some specific properties. Later, we also ran the tests for some fairly complex, contemporary examples (M1-M2) to test the performance in realistic settings.

As for software, Python 3.8 was used as base python version with packages *pyABC* 0.10.15, *redis* 3.5.3, *tumor2d* 1.0.0 (for model (M1)) and *morpheus* 2.2.0 (for model (M2)). Tests were run on the HPC structure provided by the standard nodes of Jülich Supercomputing Centres *Juwels* cluster (48 cores per node) and the University of Bonns *Bonna* cluster (32 cores per node).

5.1 (T1) ODE Model

While ODE based models are clearly not the main application area of ABC, it is fairly simple to create an example for which we can easily verify the correct result. Parameter inference for such a model is also not as time intensive as real life application examples. Therefore, we can perform it several hundred times with the new concept and compare the results to the ones of established approaches, turning an ODE-model into a handy tool to perform a sanity check on the new algorithm.

Our basic ODE-model with two parameters describes the inter-conversion between two species x_1, x_2, with rates θ_1, θ_2. It is given by

$$\frac{d}{dt}x_1 = -\theta_1 x_1 + \theta_2 x_2 \tag{32}$$

$$\frac{d}{dt}x_2 = \theta_1 x_1 - \theta_2 x_2. \tag{33}$$

Using this ODE, we can immediately verify that the results converge towards the same value as the original version for which the asymptotical correctness is known. In order to not have a fully deterministic system and to consider that in reality measurements are not perfectly accurate, we added multiplicative normal noise $\sim \mathbb{N}(0, 0.05)$ to each model evaluation.

To analyse the run times and the effect of heterogeneous simulation times, we used the same ODE and added some idle time to each model call, since otherwise the duration of a single simulation is almost trivial. This idle time was chosen in a way that imitates some single simulations that take vastly longer than the expected value. As distribution for t_{idle} we employed a log-normal one $t_{idle} \sim LogNormal(\mu_n, \sigma_n^2)$, where μ_n, σ_n^2 are the mean and variance of the underlying normal distribution. μ_n, σ_n^2 were chosen such that the real variance σ^2 takes values between 0.25 and 4 and the real mean

μ is constantly equal to 1 for the different values of σ^2, i.e.

$$\sigma_n^2 = 2\log\left(\frac{\sqrt{1^2+\sigma^2}}{1}\right), \quad \mu_n = \log(1) - \frac{\sigma_n^2}{2}. \tag{34}$$

As prior we used a uniform distribution on the unit square $\pi \sim U([0,1]^2)$ and as true parameters for example $\theta_1 = e^{-2.5} \approx 0.0821$, $\theta_2 = e^{-2} \approx 0.135$, so here a point close to one corner of the prior distribution. The observation we get from models is not a single value but instead some trajectory. So as observed data y_{obs} we also use the trajectory of the ODE-solution with the true parameters, evaluated at some fixed time points t_i. This results in a data format which consists of a value-filled array, so the distance function can be chosen as a metric on the real numbers, e.g. in our case we simply use the sum over the distances for each time point, so the metric induced by the 1-norm $d(y, y_{obs}) = \sum_i |y^{t_i} - y_{obs}^{t_i}|$. Candidates are accepted if the distance between their simulated data and the observation falls below a fixed epsilon for each generation, given by $\varepsilon_1, ..., \varepsilon_8 = 8, 4, 2, 1, 0.75, 0.5, 0.33, 0.25$.

This means the acceptance threshold assumes the value of 0.25 in the last generation, still allowing for a small discrepancy between accepted parameters in the final population and the true parameter.

Results – Correctness

Using (T1) we were able to generate dozens of runs for both the LA and the DYN scheduling version in a reasonable amount of time (i.e. a few days), which we can compare to check that the results coincide (see Figure 4).

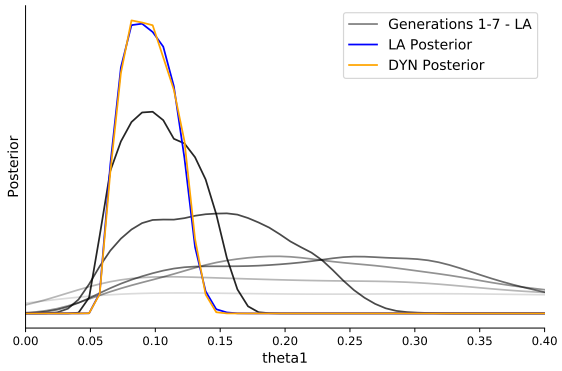


Figure 3: LA and DYN posterior and LA proposal development over the generations for one randomly selected run.

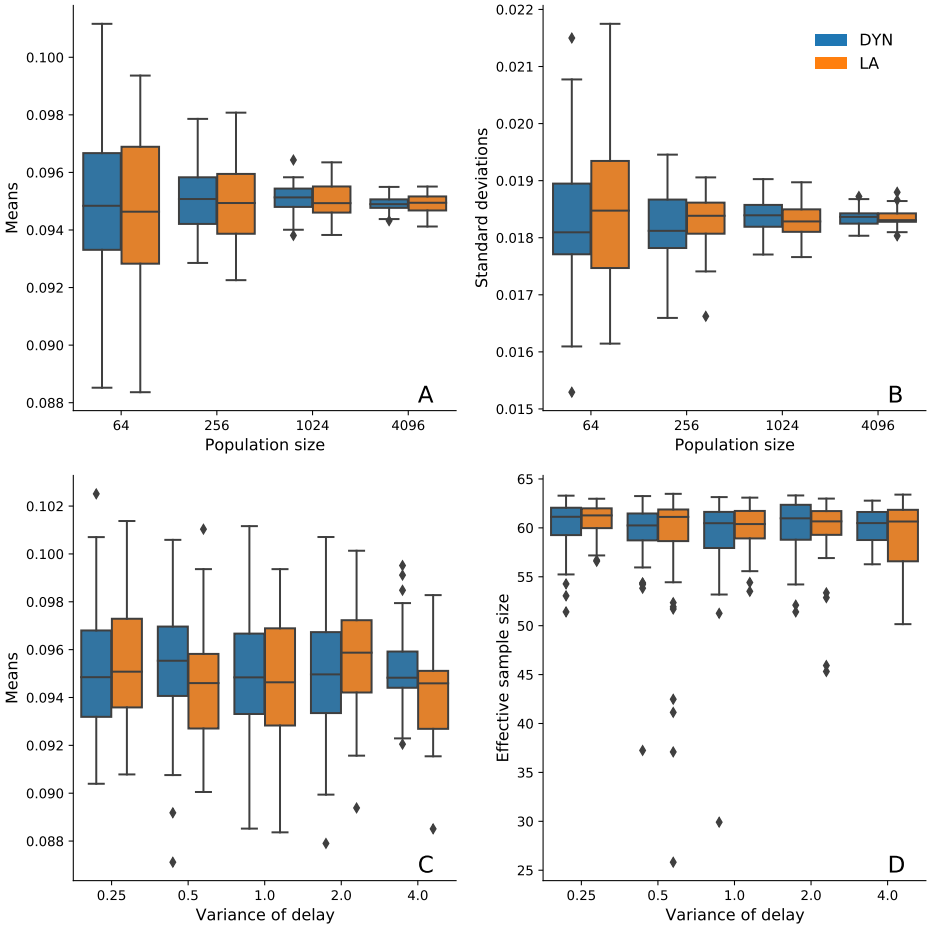


Figure 4: (T1) Results for different population sizes and idle time variances. Run on 384 workers (8 nodes with 48 workers). (A) and (B) show the results for t_{idle} variance of $\sigma^2 = 1$. (C) and (D) show results for a fixed population size of $N = 4096$ and $N = 64$ respectively.

Both scheduling approaches seem to return equal results (see Figure 3, 4) when taking into account that any two runs will always slightly differ due to the innate random nature of the sampling, the noise and the run time.

Especially in Figure 4, we can observe various key properties. (A) demonstrates how the mean for both DYN and LA converges to the same value as we increase the population size. At the same time, (B) shows that also the standard deviation within individual distributions decreases, meaning that, as N increases, both posteriors peak more sharply at the same value. (C) makes it clear that a stronger variances for t_{idle} does not affect the mean, even though there is a larger fraction of preliminary particles expected. In (D), one can see that LA does not seem to significantly affect

the effective sample size either even though we here used $N = 64$, which, on 384 workers, makes it likely that several generations are sampled completely from the preliminary.

By showing that the preliminary and the final proposal both considerably contribute to the posterior distribution, we get a strong indication that everything works as intended. And indeed, in many of the relevant scenarios the amount of particles that were based on the preliminary in the last generation ends up making up roughly half of the population, which is returned as output by the ABC-SMC algorithm (see Figure 5).

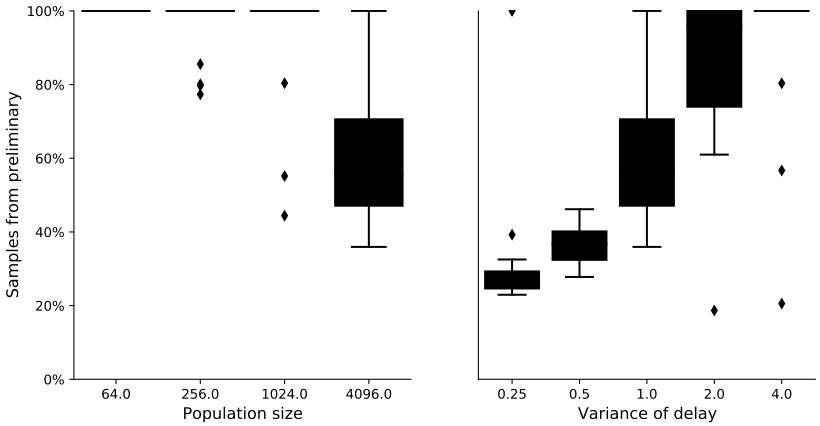


Figure 5: Fraction of particles sampled from the preliminary proposal in the last generation (for $\sigma^2 = 1$ (left) and $N = 4096$ (right))

.

In Figure 5, one can also observe how the fraction of particles sampled from the preliminary decreases as the population size increases. Which makes sense, since keeping the number of workers constant results in a more or less constant amount of resource used for the preliminary sampling between the N-th acceptance in a generation and the last worker to finish working on that generation. So, there should be a roughly constant number of preliminary acceptances opposite to an increasing size of the total population.

Similarly, a higher run time variance increases the fraction of preliminary particles, as this results in more time between the N-th acceptance and the last simulation of a generation.

Results – Run Time

To analyze how much effect the new scheduling can have on the wall time, we ran the same model on different population size to worker ratios. First,

we used an idle time with variance $\sigma^2 = 1$, the same population sizes as above, i.e. between 64 and 4096 and ran each of those on 1 to 16 nodes with 48 workers each, so on 48 to 768 workers. Each scenario was repeated between 10 and 100 times for reliable results.

One additional run was performed using only a single worker to calculate the parallel efficiency. As no parallelization is possible on a single worker anyway, that one run was used as reference for both LA and DYN, which also means that the acceleration, i.e. the fraction between wall times, is the same as the fraction between the parallel efficiencies.

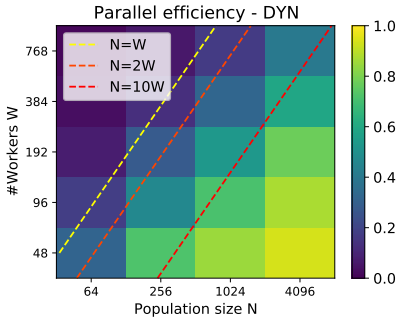


Figure 6: DYN parallel efficiency

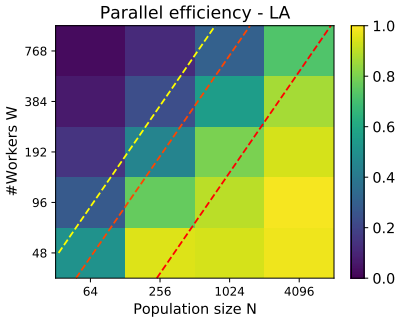


Figure 7: LA parallel efficiency

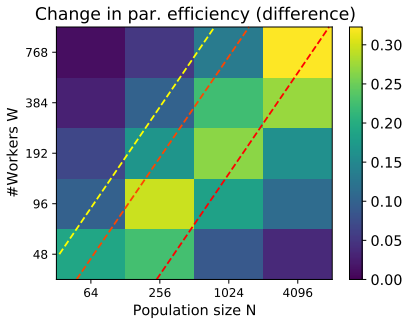


Figure 8: LA parallel efficiency - DYN parallel efficiency

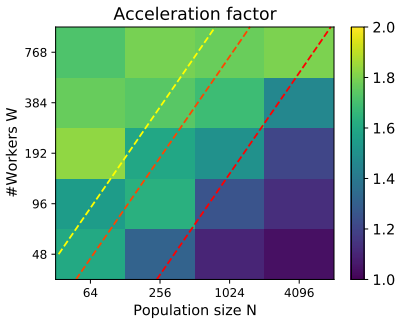


Figure 9: LA parallel efficiency / DYN parallel efficiency

We can observe a significant acceleration when using LA instead of DYN scheduling (see Figure 6-9), whenever the population size is about as large or slightly larger than the amount of workers. In the best case, the LA approach decreased the wall time by nearly a factor of two when compared to the dynamic scheduling runs. On the other hand, the difference in efficiency is much less apparent, or even almost trivial, in case the two factors, population and worker size, are vastly different.

In the case where the amount of workers is significantly larger than our population size, only few simulations remain for each worker. This can go as far as even having enough preliminary acceptances to complete the next generation even before the previous one is finished leaving all further simulations retrospectively useless, which results in LA scheduling also having a poor parallel efficiency. Nonetheless, in that case there is still a significant acceleration when compared to established approaches as LA can almost complete two generations in the time it takes to complete one using DYN.

When the population size is multiple times larger than the amount of available workers, dynamic scheduling already performed very well, leaving little room for improvement. However, even in that scenario we do not have any changes to the negative, so any additional computation necessary to enable the sampling from the preliminary population is always worth the effort.

Further, using the same setup, we examined the effect of the run time variance of the single evaluations on the acceleration. For that we ran the same tests as before with an idle time variance of $\sigma^2 = 2$; and indeed results seem to indicate that the achieved acceleration is even slightly higher than for the less varying run time with $\sigma^2 = 1$ (see Figure 10 in comparison to Figure 9).

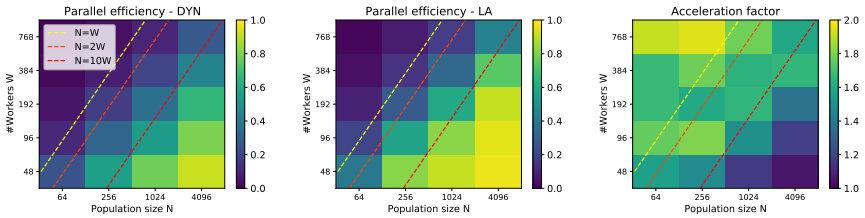


Figure 10: Parallel efficiencies and acceleration for runs with $\sigma^2 = 2$

5.2 (T2) Gillespie Algorithm based Model

Once the parameters are fixed in model (T1) the trajectory of the solution of the ODE is fully deterministic. This is in ABC relevant applications hardly the case, so as a second test model we considered one based on a Markov Jump Process (MJP).

For this we observe a population consisting of two species X, Y which develops according to a simplistic reaction network with one equation and parameter k.

$$X + Y \xrightarrow{k} 2Y \tag{35}$$

For a given parameter we use the Gillespie algorithm [Gil77] to compute a stochastically correct trajectory, i.e. to simulate the data.

Results

Using a uniform prior $\pi \sim U([0, 100])$, $k = 2.3$ as true value for our parameter, initial conditions of $X = 40$, $Y = 3$, and one possible trajectory of the true value generated using the Gillespie algorithm as observed data y_{obs}, the model was run repeatedly with different population sizes N and on varying amount of workers W. The acceptance thresholds were again chosen as a fixed schedule, with maximum accepted distance in the last generation $\varepsilon_{n_t} = 0.7$.

The results (see Figure 11 and Figure 12) reinforce the statement that the LA algorithm returns an asymptotically correct sample, as the posteriors are equal (up to statistical effects) to the ones returned by the established DYN approach. In this setting, the effective sample size does however in average seem to be slightly smaller for the runs that employed LA scheduling.

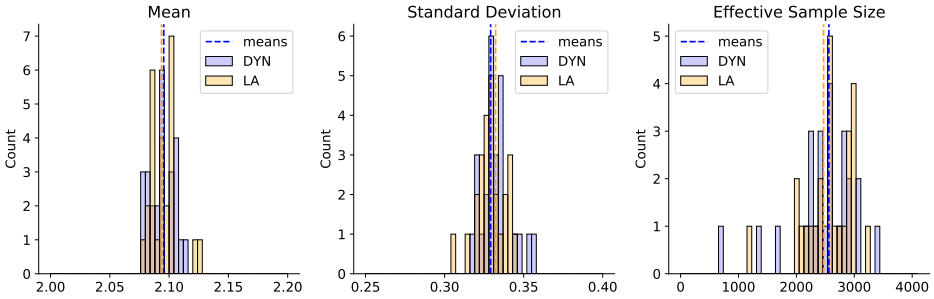


Figure 11: Results of 25 runs each of DYN and LA on 384 workers with a population size of 4096

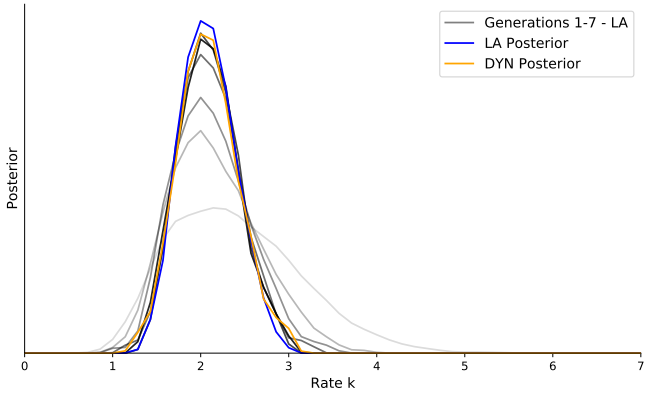


Figure 12: LA and DYN posterior as well as the development of the LA proposals over the generations for one randomly selected run.

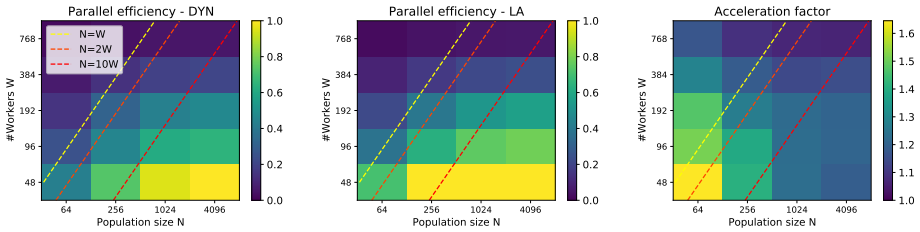


Figure 13: Parallel efficiencies for LA and DYN and acceleration due to LA for different population and worker sizes

When compared to the DYN scheduling emplying runs, LA again reduced the wall time by up to 35% (see Figure 13), depending on the choice of N and W. The outcome, however, suggests that the amount of workers is generally chosen a bit higher than reasonable for the model, as parallel efficiencies overall are quite poor whenever the amount of workers is greater than 48. This is likely due to the model being fairly simple and thus single simulations being generally quite fast (although already far more time consuming than (T1) without its idle time), increasing the fraction of time spent executing the other components of the ABC-SMC algorithm.

5.3 (T3) Unbalanced Modes

Usually, the run time of a single simulation will not be independent of the parameter candidate, but instead be somehow correlated to the region of the parameter space. A high variance of the run time paired with a meaningful correlation between the simulation time and the candidate could possibly lead to a strong imbalance in the preliminary proposal. This raises the question of what happens when it is likely that the preliminary population is strongly biased.

To analyze that phenomenon, we constructed a two mode scenario by considering a model that simply squares its only parameter, i.e. $y = \theta^2$. Assuming the observed data consists of one observation with value 1 and that the prior is uniformly distributed on $[-2, 2]$, we get two parameters that could equally be true, -1 and $+1$. By adding a large idle time to the evaluation whenever the suggested parameter is negative, it is possible that the preliminary population is entirely taken from the positive side. For population sizes that are not much larger than the amount of active workers, it even becomes quite likely that such a case occurs in at least one generation. If the reweighting works correctly, we should nonetheless end up with a posterior distribution that equally represents both modes, at least when the population size is large enough to dismiss stochastic effects.

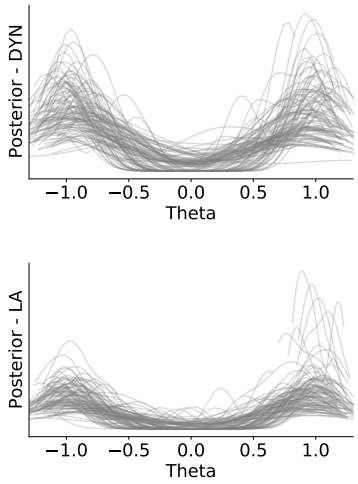


Figure 14: 100 Posterior distributions for (M3) with population size 16 on eight workers. DYN (top), LA (bottom)

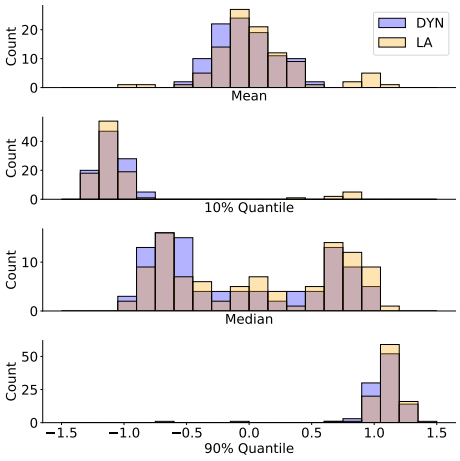


Figure 15: Comparison of means, standard deviations and quantiles for the same runs

If we use an unrealistically small population size (e.g. 16, see Figure 14), we can see some differences. In both the DYN and the LA version, we see that there are the two expected peaks, which are equally high in a majority of the runs. However, due to the limited population size, there are several posteriors that almost purely represent only one of the modes in both scenarios. Here we do see a difference between the two approaches, since using dynamic scheduling, even those unrepresentative posteriors are somewhat evenly spread whereas basically all biased posteriors in the LA case place their full weight on the region with faster evaluation time.

So there is some risk in sampling from a potentially strongly biased preliminary proposal. One way to counteract this and minimize the risk of purely towards one region biased results would be some form of limitation on what fraction of the particles can be accepted from the preliminary proposal. This might once again slow things down however, and in reality is not necessary as simply choosing an appropriate population size already leads to an unbiased posterior in almost all cases (see Figure 16).

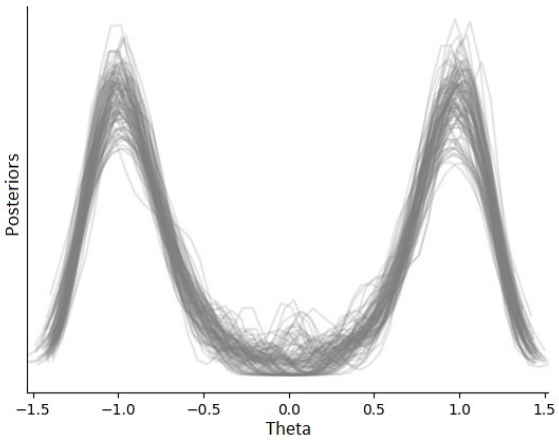
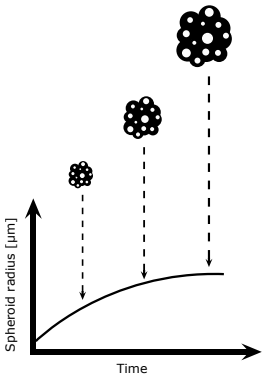


Figure 16: 100 Posterior distributions for (T3) with population size 256 on 128 workers (LA version)

Here one can observe how a, still fairly limited, population size of 256 can already almost fully eliminate the fraction of runs resulting in a biased posterior. This demonstrates why choosing a sufficiently large number of particles in each generation is crucial, as otherwise stochastic effects are too likely to lead to unexpected and undesired results.

5.4 (M1) Tumor Growth Model

The tumor growth model (M1) is our first test instance based on a real life example. The model was developed mainly in [Jag12] and aims to describe the growth of a tumor spheroid while taking into account its spacial structure. For example, the proliferating cells are almost exclusively the ones in the outer rim of the spheroid whereas the ones contained in the core are mostly necrotic.
As hybrid discrete-continuous model it uses different mathematical tools trying to accurately depict a real life biological system.



An agent-based approach (see e.g. [GSG13]), i.e. stochastic interactions between separate particles in a system, is used to model the relations between the individual cells, while a system of PDEs describes the extracellular matrix (see e.g. [The+16]). At the same time, mechanisms like cell division and cell death are modeled using a continuous time Markov process. For further details about the model see [Jag12; Jag+17] and for some biological background see e.g. [CMJ14; Kwa+14].

The version we used as test instance is a two dimensional implementation of the tumor growth model with seven parameters. It is available in the python package *tumor2d* (`https://github.com/ICB-DCM/tumor2d`). Even using several hundred workers and a population size of e.g. 1000, the parameter inference for the tumor growth model takes hours to days, making it

close to impossible to run the simulation on anything but large-scale parallelized infrastructure and strictly necessary to employ an efficient workload distribution scheme. However, the run time is not yet as exorbitantly high as for some other models, such that it was possible to perform several runs to obtain at least some reliability in the results. This was particularly important as the trajectories of the tumor growth model underlie stochastic fluctuations, especially in the early phases, as there the number of cells is still very limited. Towards the end, the cell numbers are far higher and the fluctuations are much less severe due to the averaging effect of the Law of Large Numbers.

Results – Correctness

We ran model (M1) with population sizes ranging from 200 to 2000, employing varying amounts of workers between 256 and 1536. The acceptance criteria for candidates of a generation are adaptively chosen after the previous one finishes as the 80% quantile of the previous accepted distances. The run finishes after a generation t in which the threshold ε_t falls below a certain value, in our runs $\varepsilon_{n_t} \leq 700$. Every time one scenario was run, one instance with dynamic scheduling and one with look-ahead scheduling was performed with the same setup to directly compare the results.

During the runs, no significant deviations in the quality of results were observed (see Figure 17 and Figure 18; detailed results of the other runs can be found in the attachments).

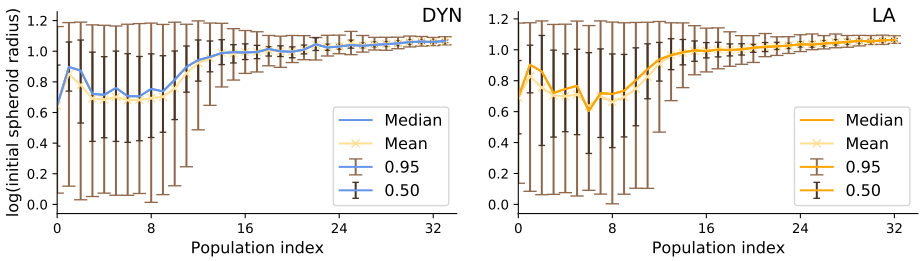


Figure 17: Direct comparison of the development of the credible intervals for one parameter from the run also shown in Figure 18.

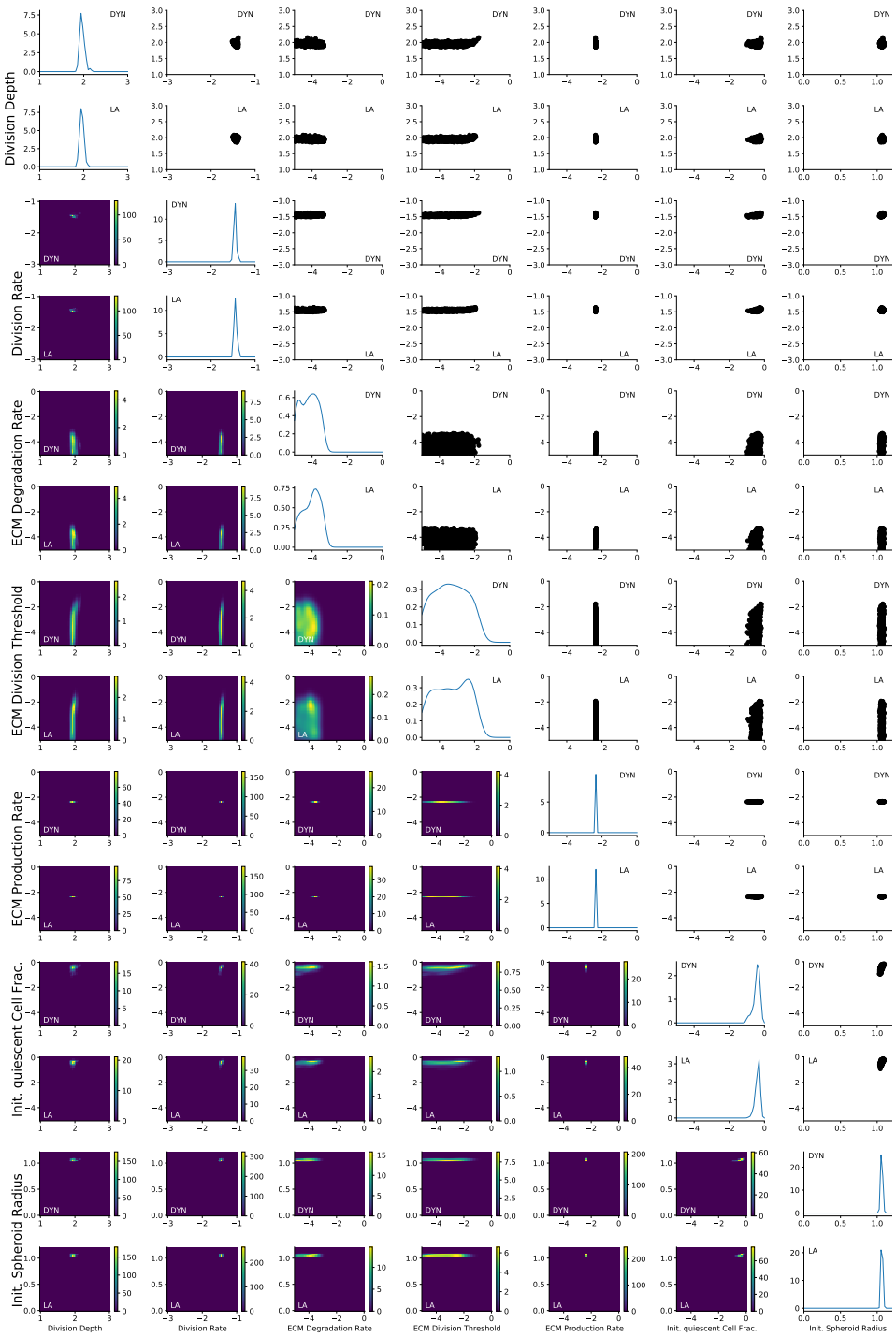


Figure 18: Direct comparison of the posterior distributions for all seven parameters of (M1) for a run with $N = 1000$ on $W = 1536$ workers using DYN and LA scheduling. The parameters with sharp peaks are visibly at the same location and if the inference returned a broader distribution, it did so in both cases.

While the fraction of preliminary particles tends to decrease over the course of the inference, it varies strongly from generation to generation (see Figure 19). As the time between the N-th acceptance in generation t and the last worker to finish is expected to be more or less constant, we have a similar amount of evaluations from the preliminary proposal in each generation. Generally, the acceptance rate decreases with the epsilon threshold however, and this also holds for the preliminary proposal based candidates. So, as more total evaluations are necessary to reach the desired number of accepted particles N, a smaller fraction of those should end up being sampled from the preliminary as the generations progress and the acceptance rate decreases. The fluctuations of this fraction exist because the acceptance threshold, and thus the acceptance rate, decreases less consistently than when using a static epsilon schedule, as can at least partially be observed in Figure 20.

As a large fraction of the population is based on the preliminary proposal for a significant part of the generations, it shows that these preliminaries have a large effect on the LA version of the ABC-SMC algorithm. Nonetheless, the posteriors are consistently similar, indicating that everything works as intended and no bias was introduced.

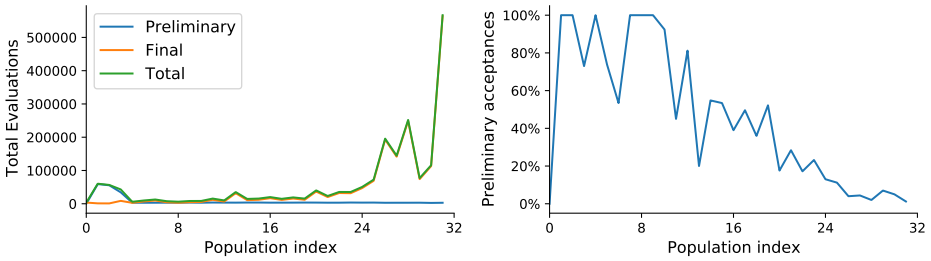


Figure 19: Total evaluations and fraction of accepted particles based on the preliminary population in each generation in the LA run

Results – Run Time

For more complex models, it is usually far more effective to use the adaptive epsilon schedule mentioned in the previous section. That however, leads to a different final acceptance threshold in every run, which strongly affects the run time. Together with the very stochastic start of the tumor growth model, this yields a high variance of the wall times, making a direct comparison more difficult. Nonetheless, we can for example observe the time point after which each epsilon value is achieved.

Figure 20 shows the development of the epsilon threshold for one of the more ordinary runs. In the more extreme cases, it also occurred that the LA scheduling took slightly longer than the corresponding DYN run, but similarly also that the LA run decreased the wall time by a factor of more

than 2 when compared to the DYN scheduling based one (see Figure 21).

These stronger differences in wall time usually seem to be traceable to one single generation taking vast amounts of time. Those long generations occur in all scheduling variants and exist most likely because the epsilon for that generation was chosen too optimistically.

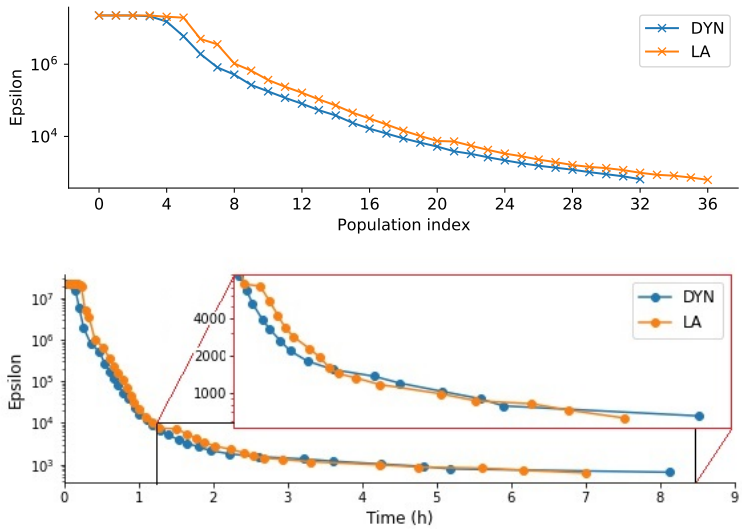


Figure 20: Development of the acceptance threshold over the generations and over time for a run with population size N=500 on W=768 workers.

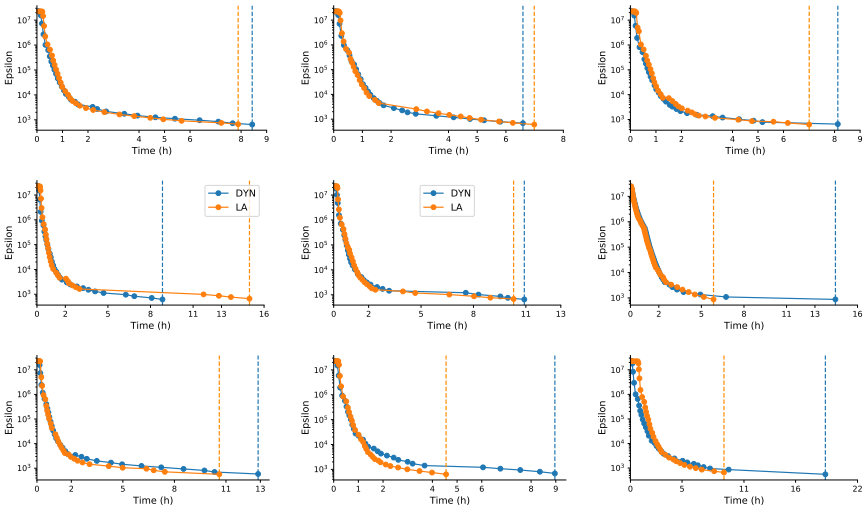


Figure 21: Development of the acceptance threshold over time for the different runs. These are all runs we performed while testing, mostly using different settings for each one, making a direct comparison difficult.

In average, it seems that an acceleration of around 10% to 20% is the expected value for the tumor model when the population size is in the same order of magnitude as the amount of workers. Over the ten times we have executed the inference of the tumor model with an adaptive epsilon schedule, we observed a mean acceleration of 17.5%, with median value being 19%.

For a more direct comparison we also tried fixing the epsilon schedule based on the experienced we gathered in the previous adaptive runs (see Figure 22 and Figure 23).

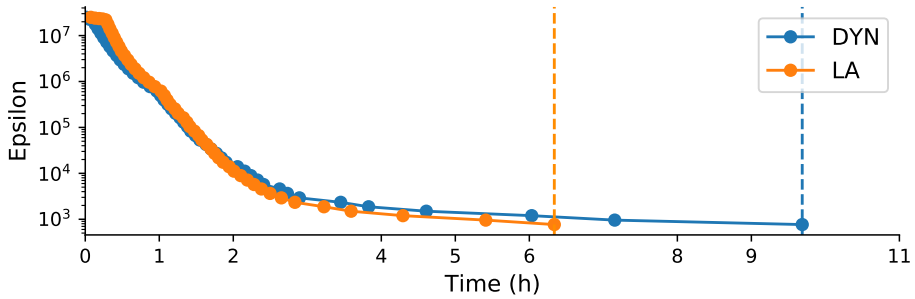


Figure 22: Epsilon over time for N=1000, W=1536 using a static epsilon schedule

Such a static epsilon rules out the possibility, that one run stops because the final epsilon is just below the threshold, while a different run might have to perform sampling for another full generation due to its previous epsilon value being just slightly too high to fulfill the termination criterion. However, the stochastic effects are still very present.

We therefore used a lower population size (N=500) on the maximum amount of workers (W=1536), to be able to run a single scenario multiple times. That way, the inference for the tumor model using LA and DYN scheduling with a static epsilon was performed another nine times each. The results for that scenario demonstrate the varying run-times quite well. They also explain why it is neither unlikely nor contradictory that for two randomly selected runs, the LA version occasionally has a higher wall time than the one working with a DYN approach, as for example observed in two of the runs presented in Figure 21. These runs also show that in this setting an acceleration of around 10% is a realistic average (see Figure 23), even though it is not chosen to optimize that acceleration.

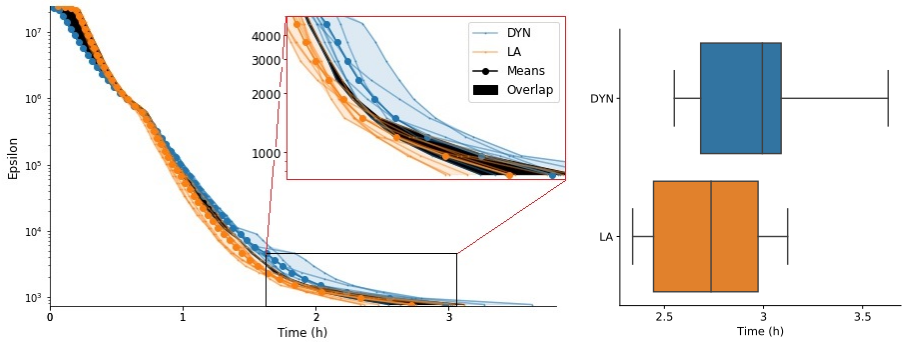


Figure 23: Development of the epsilon threshold over time for the nine repetitions of LA and DYN scheduling on 1536 workers for a population size of 500 (left). The overlap, i.e. the area where the fastest run employing DYN is faster than the slowest one using LA scheduling is marked in black. Additionally, a box-plot displaying the wall time of all runs (right).

5.5 (M2) HIV Model

Our second real life application example is a 14 parameter model developed in [Iml+19], which attempts to study the spread of HIV pathogens within a tissue-like 3D-structure of human T-lymphocytes in collagen. It combines mechanisms to quantify the replication of the pathogen with ones describing single cells and population dynamics. A Cellular Potts model (CPM) (see e.g. [GG92; SR16]) is used to describe the cell motility while the pathogen spread is modeled using ODEs.

Results

The model was run on 128 and 256 workers on the Bioquant Cluster in Heidelberg. Due to the high complexity of the model we had to use a smaller population size of 256 particles for the model to finish within less than two weeks. While this could potentially increase volatility of the trajectories, we nonetheless observed no significant difference in the quality of the results (see Figure 24), as was also the case in (M1).

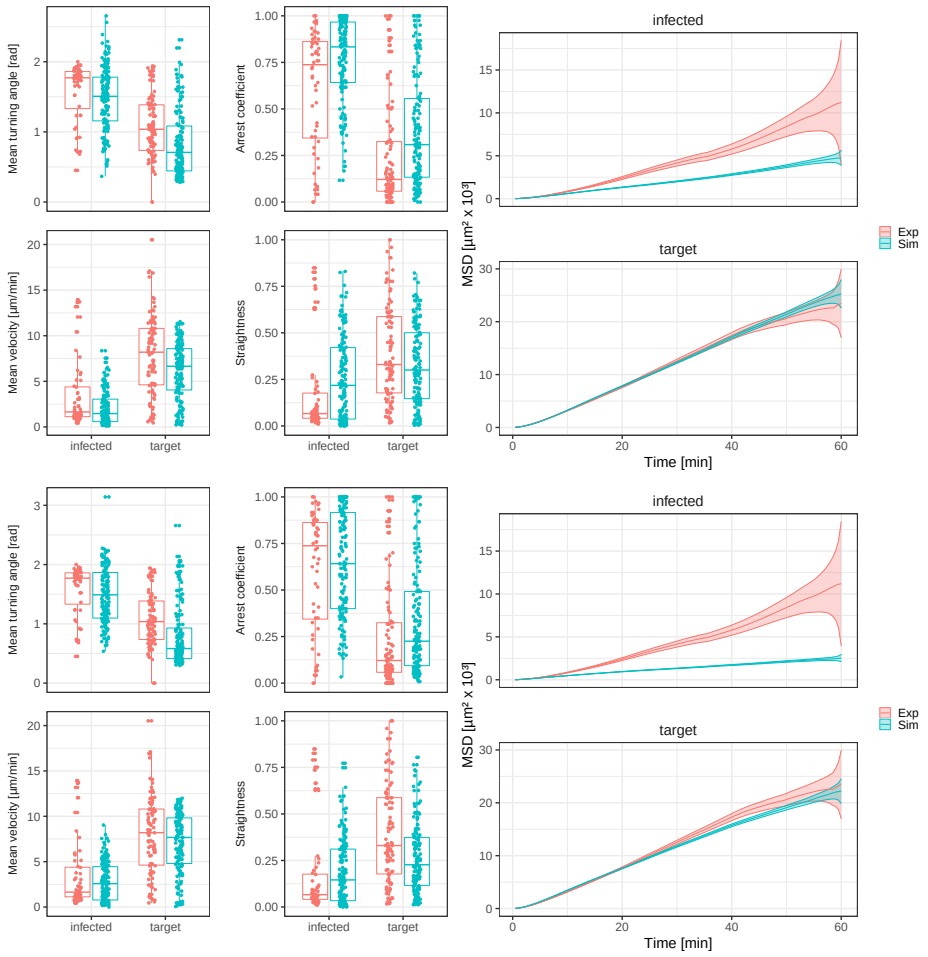


Figure 24: Comparison of best parameter fit for (M2) using DYN (top) and LA (bottom) for a population size of $N = 256$

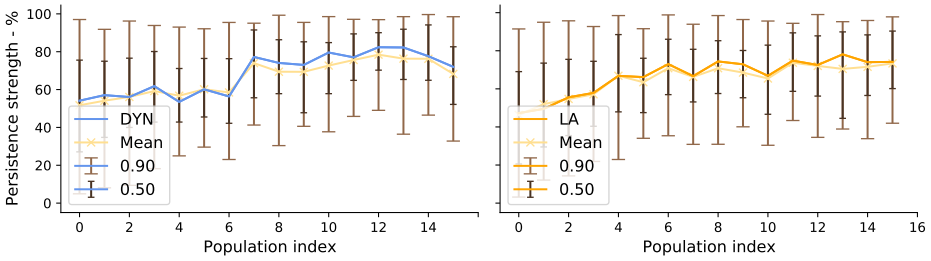


Figure 25: Credible intervals for one parameter

In total, the model was run with an adaptive epsilon schedule and with fixed thresholds and on 128 and 256 workers. Parameter inference was per-

formed using DYN and using LA scheduling for each setting once.

As the model is too computationally expensive with its run time of up to two weeks, it was not feasible to perform several repetitions of the scenarios. Thus, the results regarding the difference in wall time are not as statistically reliable, nonetheless they seem to indicate a similar average acceleration of between 10%-20% (see Figure 5.5) as we have observed in the tumor model (M1).

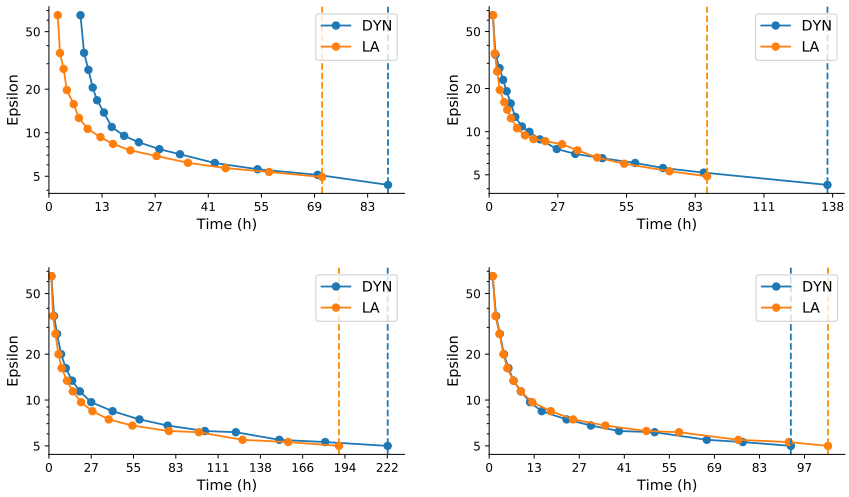


Figure 26: Development of acceptance threshold over time for the different runs of M2

One of the runs, the one with a static epsilon schedule on $W = 256$ workers, took longer to return results using the LA concept than the corresponding DYN run. As we have seen in the tumor model (specifically Figure 23), this is likely to be simply due to the existing high wall time variance, however we can not entirely rule out whether there also is another reason behind this.

6 Further Enhancements

It may be possible to further accelerate the inference using different smaller enhancements, some of which are motivated by details mentioned in the previous sections. In this section we discuss these and some other possibilities more thoroughly. Some of the enhancements have also already been implemented and can be found in the attachments.

Optimizing Effective Sample Size

As mentioned in Section 4.1, which fraction of the total weight we assign to which sub-population is freely chosen, since, due to the properties of the importance weights, a sample from any of those sub-populations gives an asymptotically correct approximation of the posterior. Therefore, it is possible to not simply give each sub-population the weight corresponding to the fraction of its particles in the population N_1/N, but instead choose the factor α in a way that maximizes the resulting effective sample size. In the generations where there is only one proposal, because either no or all particles were sampled from a single preliminary, we can simply skip these calculations.

In the case with only one preliminary proposal alongside the final one, this is a simple one-dimensional optimization problem. If α is the total fraction of the weight we want to assign to the preliminary particles (with individual weights $w_1 \dots w_{N_1}$), the effective sample size is given by

$$n_{ess}(\alpha) = \frac{\left(\sum_{i=1}^{N_1} \alpha w_i + \sum_{i=N_1+1}^{N} (1-\alpha) w_i\right)^2}{\sum_{i=1}^{N_1} (\alpha w_i)^2 + \sum_{i=N_1+1}^{N} ((1-\alpha) w_i)^2}. \tag{36}$$

If we use the self-normalizing weights as introduced in Section 4.1, each sub-population is at first assigned a total weight of 1, i.e. $\sum_{i=1}^{N_1} w_i = 1 = \sum_{i=N_1+1}^{N} w_i$. For simplicity of notation we write $\sum_{i=1}^{N_1} (w_i)^2 = a$ and $\sum_{i=N_1+1}^{N} (w_i)^2 = b$. This lets us write the derivative of n_{ess} as

$$\frac{d}{d\alpha} n_{ess}(\alpha) = \frac{d}{d\alpha} \frac{\alpha + 1 - \alpha}{\alpha^2 a + (1-\alpha)^2 b} \tag{37}$$

$$= \frac{-2(\alpha a - (1-\alpha) b)}{(\alpha^2 a + (1-\alpha)^2 b)^2}. \tag{38}$$

Since all weights and thus a,b are strictly positive, the only root of this term

and therefore our maximum is at

$$\alpha = \frac{b}{a+b} = \frac{\sum_{i=N_1}^{N} (w_i)^2}{\sum_{i=1}^{N_1} (w_i)^2 + \sum_{i=N_1+1}^{N} (w_i)^2}. \tag{39}$$

For multiple preliminary proposals, the analytical solution maximizing the effective sample size is not quite as easy to formulate. However, we can define a function that computes the ESS for an arbitrary number of proposals and given $\alpha_i \in [0, 1]$ with $\sum_i \alpha_i = 1$ and use numerical optimization to compute the approximately optimal weight assignments, i.e. the maximum of that function. This does require performing a single optimization of said function in each generation, which should however take only trivial amounts of time when compared to the simulations we perform.

Multiple Generation Look-Ahead

Figure 5 and Figure 19 demonstrate that there are several cases in which every single accepted particle was proposed based on the preliminary distribution. This happens mainly in the early generations where the acceptance rate is still very high, but can also occur if the run time of different simulation varies strongly or if the amount of workers is large when compared to the population size. The number of samples in preliminary mode scales with the run time variance and the amount of employed workers W, so any of these scenarios can lead to a number that is higher than the population size N divided by the acceptance rate, resulting in a case in which we already have N acceptances before we finish evaluating all preliminary samples.

If this occurs frequently, it might make sense to not simply discard all the remaining candidates, since their simulations have already been performed, but instead to cache them and evaluate their acceptance in the generation that follows. Then, of course, using the updated acceptance criteria of that generation. While this will probably lead to an even lower fraction of such particles being accepted, it does reduce the amount of time-consuming evaluations.

On the other hand, this again means we have to store our simulation results somewhere for an entire generation. Some systems may not have as much accessible memory to allow saving of (close to) arbitrarily many of these results, possibly even to a degree where the memory becomes the limiting factor. Again, this almost exclusively becomes an issue in early generations, where simulation times tend to be more heterogeneous, as the proposal covers a larger region of the parameter space. There, the amount of preliminary samples can possibly become multiple orders higher than required or useful. So, even if this is uncommon, it might be necessary to

generally restrict how many samples we want to perform from the preliminary in order to not exceed available storage capacities, trying to buffer all simulation results. One way to do this is to stop sampling from the preliminary once the amount of samples exceeds some number which is proportional to the population size N divided by the acceptance rate of the previous generation; e.g. multiplied by some factor to account for the expected decrease of the acceptance rate.

These two ideas contradict each other to some degree, since looking multiple generations ahead would require a huge amount of samples and therefore be straining on storage space whereas limiting the amount of preliminary samples renders the first suggestion mostly useless.

Whether any of the approaches are actually beneficial and which one would be more fruitful is fully dependant on the circumstances (i.e. the model and the infrastructure used) and therefore there is no generally correct choice.

Several Preliminary Proposals

The initial concept chooses its preliminary proposal as soon as the first N particles have been accepted, and samples all candidates from a distribution based on this proposal until the final one is known. Since this proposal is biased towards particles with a faster evaluation time, it can be beneficial to update it as workers finish and more of the final acceptances become known. This new preliminary proposal should have a weaker bias, at least when not considering stochastic fluctuations. Simulations for candidates sampled from the new distribution could thus possibly have a higher chance of being close enough to the data to be accepted. And even if the acceptance rate does not show a strict increase, it at least is non-decreasing for large N.

Also conceptually, it would be no problem to use more than one preliminary proposal, as mentioned in chapter 4.2.

What remains is the question of how and when to update our proposal. One strategy is to compute a new distribution whenever a certain fraction of workers finish their work on the original generation, e.g. whenever the next tenth of the workers has finished we generate the new distribution. However, all those workers could return a rejected particle in which case no changes will be made. In theory, it would be possible to compute a new preliminary proposal whenever a worker finishes and returns a new acceptance. That would require generating an additional proposal for each additional acceptance. Since this is not too computationally demanding, it would even be feasible to proceed in such a way, the benefit of a single new particle is however also very questionable.

Instead, the most promising approach is to update the proposal whenever a certain fraction of the previous preliminary particles have been replaced by

new ones. How high that fraction should be chosen or even what strategy works best is however again highly dependent on the model and circumstances.

7 Conclusion

Whenever the likelihood function can be formulated and efficiently evaluated, there is no reason to use an ABC method [Bea10]. However, in most contemporary models this is not the case, mostly because evaluation is too computationally expensive. Then it is crucial to have a feasible alternative. In most applications that feasible alternative is running an ABC-SMC scheme on large scale parallel infrastructure.

In currently common approaches, a large part of the infrastructures workers frequently become idle and remain in that state for at least a non-trivial amount of time. In this thesis, we have presented a new strategy, the look-ahead scheduling, which fully exploits all available resources at almost all times in order to ensure nearly optimal usage of the infrastructure at hand. To make use of the otherwise idle workers, LA uses a potentially biased preliminary proposal distribution to proactively start sampling for the next generation. Although this means that there is a difference in how the sampling is performed, we have shown that the new strategy returns unbiased, asymptotically correct results. Further, we demonstrated that LA is combinable with previously developed enhancements which improve on other aspects of ABC-SMC schemes, such as the adaptive selection of the algorithms components.

We developed an algorithm implementing the new strategy and tested it on various models. The results show an acceleration of the inference by around 10% to 20%, when compared to established parallelization approaches. The improvement is especially evident, when the amount of workers employed is of the same order of magnitude as the population size, which is a very realistic setting when working with high performance clusters.

Depending on specific details of the model at hand and the available infrastructure, it may be possible to further accelerate the inference using smaller enhancements, by for example weighting each sub-population in a ESS-maximizing way or by using more than one preliminary proposal. We have presented several such enhancements, and also implemented some of them, however, most of them have a significant effect only in specific scenarios.

References

[Bea10] Mark Beaumont. 'Approximate Bayesian Computation in Evolution and Ecology'. In: *Annual Review of Ecology, Evolution, and Systematics* 41 (2010), pp. 379–406. DOI: http://dx.doi.org/10.1146/annurev-ecolsys-102209-144621.

[BP63] Thomas Bayes and Richard Price. 'An essay towards solving a problem in the doctrine of chances. By the late Rev. Mr. Bayes, F. R. S. communicated by Mr. Price, in a letter to John Canton'. In: (1763). DOI: https://doi.org/10.1098/rstl.1763.0053.

[CMJ14] Kyle Carver, Xin Ming and Rudolph L Juliano. 'Multicellular Tumor Spheroids as a Model for Assessing Delivery of Oligonucleotides in Three Dimensions'. In: *Molecular Therapy - Nucleic Acids* 3 (2014), e153. ISSN: 2162-2531. DOI: https://doi.org/10.1038/mtna.2014.5. URL: https://www.sciencedirect.com/science/article/pii/S2162253116302931.

[Dut+17] Ritabrata Dutta et al. 'ABCpy: A User-Friendly, Extensible, and Parallel Library for Approximate Bayesian Computation'. In: *Proceedings of the Platform for Advanced Scientific Computing Conference*. PASC '17. Lugano, Switzerland: ACM, 2017, 8:1–8:9. ISBN: 978-1-4503-5062-4. DOI: 10.1145/3093172.3093233. URL: http://doi.acm.org/10.1145/3093172.3093233.

[FBS11] Sarah Filippi, Chris Barnes and M Stumpf. 'On optimal kernels for ABC SMC'. In: *arXiv preprint arXiv:1106.6280* (2011).

[FP12] Paul Fearnhead and Dennis Prangle. 'Constructing summary statistics for approximate Bayesian computation: semi-automatic approximate Bayesian computation'. In: *Journal of the Royal Statistical Society: Series B (Statistical Methodology)* 74.3 (2012), pp. 419–474. DOI: 10.1111/j.1467-9868.2011.01010.x. eprint: https://rss.onlinelibrary.wiley.com/doi/pdf/10.1111/j.1467-9868.2011.01010.x. URL: https://rss.onlinelibrary.wiley.com/doi/abs/10.1111/j.1467-9868.2011.01010.x.

[GG92] François Graner and James A. Glazier. 'Simulation of biological cell sorting using a two-dimensional extended Potts model'. In: *Phys. Rev. Lett.* 69 (13 Sept. 1992), pp. 2013–2016. DOI: 10.1103/PhysRevLett.69.2013.

[Gil77] Daniel T. Gillespie. 'Exact stochastic simulation of coupled chemical reactions'. In: *The Journal of Physical Chemistry* 81.25 (1977), pp. 2340–2361. DOI: 10.1021/j100540a008. eprint: https://doi.org/10.1021/j100540a008. URL: https://doi.org/10.1021/j100540a008.

[GSG13] David Gammack, Elsa Schaefer and Holly Gaff. 'Chapter 4 - Global Dynamics Emerging from Local Interactions: Agent-Based Modeling for the Life Sciences'. In: *Mathematical Concepts and Methods in Modern Biology*. Ed. by Raina Robeva and Terrell L. Hodge. Boston: Academic Press, 2013, pp. 105–141. ISBN: 978-0-12-415780-4. DOI: https://doi.org/10.1016/B978-0-12-415780-4.00004-1. URL: https://www.sciencedirect.com/science/article/pii/B9780124157804000041.

[Iml+19] A. Imle et al. 'Experimental and computational analyses reveal that environmental restrictions shape HIV-1 spread in 3D cultures'. In: *Nature Communications* 10.1 (May 2019), p. 2144. DOI: 10.1038/s41467-019-09879-3.

[Jag+17] Nick Jagiella et al. 'Parallelization and High-Performance Computing Enables Automated Statistical Inference of Multiscale Models'. In: *Cell Systems* 4 (2017), pp. 194–206. DOI: http://dx.doi.org/10.1016/j.cels.2016.12.002.

[Jag12] Nick Jagiella. 'Parameterization of Lattice-Based Tumor Models from Data.' In: (Sept. 2012).

[Kan+16] Antti Kangasrääsiö et al. 'ELFI: Engine for Likelihood-Free Inference'. In: *NIPS 2016 Workshop on Advances in Approximate Bayesian Inference*. Barcelona, Spain, 2016.

[KH17] Emmanuel Klinger and Jan Hasenauer. 'A Scheme for Adaptive Selection of Population Sizes in Approximate Bayesian Computation - Sequential Monte Carlo'. In: *Computational Methods in Systems Biology*. Ed. by Jérôme Feret and Heinz Koeppl. Cham: Springer International Publishing, 2017, pp. 128–144. ISBN: 978-3-319-67471-1.

[KRH18] E Klinger, D Rickert and J. Hasenauer. 'pyABC: distributed, likelihood-free inference'. In: *Bioinformatics* 34.20 (2018), pp. 3591–3593. DOI: 10.1093/bioinformatics/bty361.

[KTB11] D. P. Kroese, T. Taimre and Z.I. Botev. *Handbook of Monte Carlo Methods*. Wiley, 2011.

[Kwa+14] Karina Kwapiszewska et al. 'A microfluidic-based platform for tumour spheroid culture, monitoring and drug screening'. In: *Lab on a chip* 14 (May 2014), p. 2096. DOI: 10.1039/c4lc00291a.

[Lee12] Peter M. Lee. *Bayesian Statistics: An Introduction*. 4th ed. Wiley, 2012.

[MDJ06] Pierre Del Moral, Arnaud Doucet and Ajay Jasra. 'Sequential Monte Carlo Samplers'. In: *Journal of the Royal Statistical Society. Series B (Statistical Methodology)* 68.3 (2006), pp. 411–436. ISSN: 13697412, 14679868. URL: http://www.jstor.org/stable/3879283.

[MM00] P.Del Moral and L. Miclo. 'A Moran particle system approximation of Feynman–Kac formulae'. In: *Stochastic Processes and their Applications* 86.2 (2000), pp. 193–216. ISSN: 0304-4149. DOI: https://doi.org/10.1016/S0304-4149(99)00094-0. URL: https://www.sciencedirect.com/science/article/pii/S0304414999000940.

[MU49] N. Metropolis and S. Ulam. 'The Monte Carlo Method'. In: *Journal of the American Statistical Association* 44.247 (1949). PMID: 18139350, pp. 335–341. DOI: 10.1080/01621459.1949.10483310. eprint: https://www.tandfonline.com/doi/pdf/10.1080/01621459.1949.10483310. URL: https://www.tandfonline.com/doi/abs/10.1080/01621459.1949.10483310.

[Owe13] Art B. Owen. *Monte Carlo theory, methods and examples.* 2013.

[Pra17] Dennis Prangle. 'Adapting the ABC Distance Function'. In: *Bayesian Analysis* 12.1 (2017), pp. 289–309. DOI: 10.1214/16-BA1002.

[RC04] C.P. Robert and G. Casella. *Monte Carlo Statistical Methods.* Springer, 2004. DOI: 10.1007/978-1-4757-4145-2.

[SFB18] Scott A Sisson, Yanan Fan and Mark Beaumont. *Handbook of approximate Bayesian computation.* CRC Press, 2018.

[SFT07] S. A. Sisson, Y. Fan and Mark M. Tanaka. 'Sequential Monte Carlo without likelihoods'. In: *Proceedings of the National Academy of Sciences* 104.6 (2007), pp. 1760–1765. ISSN: 0027-8424. DOI: 10.1073/pnas.0607208104. eprint: https://www.pnas.org/content/104/6/1760.full.pdf. URL: https://www.pnas.org/content/104/6/1760.

[SH20] Yannik Schälte and Jan Hasenauer. 'Efficient Exact Inference for Dynamical Systems with Noisy Measurements using Sequential Approximate Bayesian Computation'. In: *bioRxiv* (2020). DOI: 10.1101/2020.01.30.927004. eprint: https://www.biorxiv.org/content/early/2020/01/31/2020.01.30.927004.full.pdf. URL: https://www.biorxiv.org/content/early/2020/01/31/2020.01.30.927004.

[SR16] A. Szabò and M. H. M. Roeland. 'Cellular Potts Modeling of Tumor Growth, Tumor Invasion, and Tumor Evolution'. In: *Frontiers in oncology* 3 (Apr. 2016). DOI: 10.3389/fonc.2013.00087.

[Sri02] Rajan Srinivasan. *Importance Sampling. Applications in Communications and Detection*. 1st ed. Springer, 2002. DOI: 10.1007/978-3-662-05052-1.

[ST10] A. Sottoriva and S. Tavaré. 'Integrating approximate Bayesian computation with complex agent-based models for cancer research'. In: *COMPSTAT 2010 – Proceedings in Computational Statistics*. Ed. by G. Saporta and Y. Lechevallier. Springer Physica-Verlag HD, 2010, pp. 57–66.

[Tav+97] S. Tavaré et al. 'Inferring coalescence times from DNA sequence data'. In: *Genetics* (1997).

[The+16] Achilleas D. Theocharis et al. 'Extracellular matrix structure'. In: *Advanced Drug Delivery Reviews* 97 (2016). Extracellular Matrix (ECM) and ECM-like materials: Therapeutic Tools and Targets in Cancer Treatment, pp. 4–27. ISSN: 0169-409X. DOI: https://doi.org/10.1016/j.addr.2015.11.001. URL: https://www.sciencedirect.com/science/article/pii/S0169409X15002574.

[Ton+09] T. Toni et al. 'Approximate Bayesian computation scheme for parameter inference and model selection in dynamical systems'. In: *J. R. Soc. Interface* 6 (July 2009), pp. 187–202.

[Ver38] P. F. Verhulst. *Correspondance mathématique et physique. Notice sur la loi que la population suit dans son accroissement*. 1838. Chap. 10, pp. 113–121.

Acknowledgements

First and foremost, i want to credit Yannik Schälte and Emad Alamoudi, as they were fully involved in the development, implementation and testing of the LA concept from start to finish. Further, i want to thank Jan Hasenauer for his supervision, always making sure we were working in the right direction. I also want to thank Alexander Effland for acting as the second supervisor and examiner. As the group around the FitMultiCell project was always providing valuable input, they too have my thanks, especially Nils Bundgaard, who provided large parts of the HIV model.

Des weiteren möchte ich Anselm, Hannes und dem Stochagier dafür danken, dass sie sich die Zeit genommen haben die Arbeit zu lesen, Unverständlichkeiten und (hoffentlich) sämtliche Fehler hervorzuheben und diese auszumerzen.

Attachments

The following documents can found on the attached USB-stick (and on `https://github.com/FelipeR888/MTAttachments`). They include all scripts and programs used to perform the tests and visualize their results, and a version of the relevant python packages. All test results are contained as well, often however only in their visualized form, as the databases in which all details are saved are too large.

- **BaseBatchScripts** A generally functional version of the shell scripts used initiate the pyABC based parameter inference on an HPC. When executed, it starts a Redis-server, connects workers to the server and runs the python program containing the inference assignments. Originally provided by Emad Alamoudi.
- **MasterThesis** A digital version of the thesis.
- **Models** Contains all files required to run the models, evaluate their results and create the figures included in the body of this thesis. Also contains the results of the runs we performed, however mostly without the database in which the full run details are saved, as these files are too large (usually several hundred MB for each run). The databases remain saved on the cluster infrastructure on which the respective test was run and can be made available upon request.
  - **M1_Tumor** Everything required to run and evaluate the tests of the tumor growth model (M1). The inference is performed in the python programs *TumorAdaptiveEps.py* and *TumorListEps.py* with an adaptive and a static epsilon respectively. Executing these files returns detailed information about each run in a database file, and some additional statistics about the effect of the look-ahead scheduling is written into .csv file.
    - **BatchScript** A version of the shell scripts tailored to the tumor model test runs. The main changes are a different worker call, necessary to prevent daemonic processes, and the amount of active nodes being handed to the python program call as an argument.
    - **Figures** The jupyter notebooks load the run data in order to visualize the results as seen in Figures 17-22 (created in *TumorVisualization.py*) and Figure 23 (created in *TumorVisStat.py*). The figures are the ones summarizing the results of the runs with a static epsilon schedule on equal conditions (see Section 5.4).
    - **Testresults** The results of the performed (M1) runs, sorted by population size and workers used. Does not include the

databases themselves, but instead some extracted information summarizing each generation, the .csv files and all visualizations for each individual run.

- **M2_HIV** The python programs used to perform parameter inference for the HIV model (M2) and the results of the eight runs (.csv files). Also, the notebook used to visualize these results, creating Figures 25, 26. Model and posterior plots (Figure 24) provided by Nils Bundgaard.
- **T1_ODE** Everything required to run and evaluate the tests of the ODE model (T1). The inference is performed multiple times by executing the python program *ODEWLogfiles.py*. Executing this file returns detailed information about each repetition in a separate database file, some additional statistics about the effect of the look-ahead scheduling for each run in different .csv files and additionally a .txt file containing summarized wall time statistics.
  - **BatchScript** A version of the shell scripts tailored to the ODE model test runs. Mainly features some additional arguments being passed between the shell scripts.
  - **Figures** The python program (*ODEBoxPlotCreation.py*) reads in all databases and .csv files to plot the summarizing graphs showing equivalence of the posteriors as seen in Figures 3-5. The jupyter notebook creates the wall time comparison plots (Figures 6-10) from the wall time summaries in the .txt files.
  - **Testresults** The summarized wall time results of the test sorted by the runtime variance and the amount of nodes. Databases and .csv logfiles remain on the server, as there are several hundred of them.
- **T2_MJP** Everything required to run and evaluate the tests of the Gillespie algorithm based model (T2). Basically a copy of the files in T1_ODE, only with adapted paths and the model in the test file (*MJPRuntimeTest.py*) changed to the one for (M2). See above for details about the sub-folders.
- **T3_UnbModes** Files used to run model (T3) both locally (*UnbalancedModes.ipynb*) and with minor modifications on an HPC (*UnbalancedModes.py*). Additionally, two of the returned databases of the HPC runs (one using DYN, one LA scheduling) and the visualizations of the local and the cluster runs.

- **Other Figures** The remaining Figures used in the main body (Figure 1,2 and the sketch of the tumor model). Including the program used to create Figure 1, which reads in a file with a list of accepted or rejected particles with their simulation times and imitates a STAT,

DYN and LA scheduling approach to show how these particles would be distributed on the workers.

- **pyABC** Two versions of the pyABC package: Version 0.10.14 which was used for testing and a modified version, including some changes to the built-in visualization functions and a first implementation of some of the Enhancements mentioned in Section 6.
- **tumor2d** Version 1.0.0 of the tumor2d package required to run the tumor model (M1).