Optimizing Resource Allocation for Tumor Simulations over HPC Infrastructures

Errikos Streviniotis
Technical University of Crete
Athena Research Center
estreviniotis@tuc.gr

Nikos Giatrakos
Technical University of Crete
Athena Research Center
ngiatrakos@tuc.gr

Yannis Kotidis
Athens University of Economics and Business
kotidis@aueb.gr

Thaleia Ntiniakou
Barcelona Supercomputing Center
thaleia.ntiniakou@bsc.es

Miguel Ponce de Leon
Barcelona Supercomputing Center
miguel.ponce@bsc.es

***Abstract*—We introduce RATS (Resource Allocator for Tumor Simulations), the first optimizer for the execution of tumor simulations over HPC infrastructures. The optimization framework of RATS incorporates 3 vital performance criteria (i) expected utility of a simulation in terms of effective drug combination on the simulated tumor, (ii) simulation execution time and (iii) number of cores required for achieving that execution time. RATS is to be used by life scientists at the Barcelona Supercomputing Center to not only remove the burden of blindly guessing the core hours we need to reserve from HPC admins to study various tumor treatment methodologies, but also to help in more rapidly distinguishing effective drug combinations, thus, potentially cutting time to market for new cancer therapies.**

I. Introduction & Motivation

In precision medicine scenarios, the use of virtual patients in in-silico medical trials holds the promise of revolutionizing the process of novel treatment development by providing an easy, cost-effective and patient-friendly way to perform preliminary assessment of drug safety and efficacy. In this context, simulation frameworks incorporating computational models of incurable diseases are used to explore the effect of various treatment methodologies towards remedying or improving the quality of patient life under a wide variety of circumstances. When a particular treatment shows promising results in-silico, it is selected to be further investigated, initially in-vitro and, then, potentially in in-vivo trials. This way, the time-to-market for effective treatments is considerably cut down.

In the battle against cancer, at the Barcelona Supercomputing Center, we develop a “virtual laboratory” for studying tumor growth and evolution using a prominent, open-source simulation framework, namely PhysiBoSS 2.0 [1]. For reasons we explain below, this endeavor entails challenges both for life- and computer-scientists.

PhysiBoSS incorporates in-silico models of cell systems found in in-vivo tumors and produces output streams describing the state of each individual cell agent and the tumor as a whole, as the simulation process is taking place. Tumor growth and evolution throughout an individual simulation is affected by the drugs of a particular treatment that are applied on the simulated tumor. Effective drug combinations lead tumor cells to necrosis or apoptosis, while ineffective ones result in cell proliferation. Figure 1 shows examples of a pair of running simulations and their visualized output. The simulation shown at the upper part of Figure 1 corresponds to an effective treatment methodology because the number of alive (green) cells is reduced, while the number of cells led to necrosis (red) or apoptosis (brown) become a majority as time passes. On the other hand, the simulation at the lower part of Figure 1 illustrates an ineffective treatment methodology since most of the simulated tumor cells remain alive or are proliferating.

The modeled phenomena throughout the simulation are subject to the inherent complexity of biological systems and affected by the interplay between different processes that occur at different scales. For instance, they depend on the molecular mechanisms by which individual cells can develop resistance to a particular drug [2], which are complex in their own right. Moreover, they depend on various types of dynamic processes concerning populations of cells. Examples of the latter include the variability in the gene expression profiles of different cells, which gives rise to heterogeneous populations, the competition for resources such as space and nutrients, as well as the interaction or cross-talk between different cells [3]. Consequently, tumor simulations are extremely computationally demanding. For instance, studying the effect of drug combinations on simulated tumors of realistic sizes can generate cell state data of 100 GB/min [4]. These simulations cannot run on commodity hardware and traditional computer clusters, but require High Performance Computing (HPC) resources.

A fundamental treatment methodology that is used in such tumor simulations studies the effect of the Tumor Necrosis Factor (TNF), a signalling molecule that binds to cell receptors and can trigger a wide range of different responses [5]. In particular, TNF can induce death in cancer cells by activating specific downstream signalling pathways, thus restraining the growth of a tumor. To simulate the effect of TNF, a sophisticated Boolean network is used for modelling intracellular signalling and cell fate [6].

To execute a simulation, PhysiBoSS receives as input a XML file composing a number of values describing the TNF-based treatment methodology. Each combination of TNF parameters that compose a particular treatment results in simulations the utility, in term of killing cancer cells, and the computational demands of which are impossible to estimate beforehand. This is due to the complexity of the employed simulation models and the sensitivity to altering the parameters from one tested treatment to another. Nonetheless, we need to apply in advance to HPC administrators in order to reserve (a) the cores that we will need for in-silico trials and (b) the time simulations would need to run those trials. For instance, in PRACE (https://prace-ri.eu/), proposals for granting access on the provided HPC infrastructure should explicitly declare the Total Core Hours. In our setup, a core hour refers to the number of processor units (cores) used to run simulations, multiplied by the duration of the job in hours. With 24 core hours one can simulate for 1 day on a 1 core machine, half a day on a 2 core machine and so on.

In this work, we propose RATS (Resource Allocator for Tumor Simulations) which is the first, to our knowledge, resource allocator for optimizing the execution of tumor simulations over HPC infrastructures. RATS focuses on learning (i) the trade-offs between *execution time* simulations require to complete vs the available *number of cores* and (ii) the *utility* of treatment methodologies on tumor simulations. In particular, RATS incorporates a novel combination of active learning and Bayesian Optimization-based estimators to predict the above performance criteria. RATS optimizes the execution of in-silico trials solving all the aforementioned practical barriers. Our contributions are:

1) RATS is the first optimizer that simultaneously (i) dictates the optimal number of cores for executing any given number of simulation trials, (ii) reduces the total simulation time under given core capacity constraints, and (iii) predicts potentially effective treatment methodologies and prioritizes their simulated execution higher compared to expectedly ineffective ones. RATS relieves life scientists from the responsibility of guessing (and frequently miscalculating) the core hours needed for simulations. Instead, it allows to focus on effectively applying field expertise on testing treatment methodologies.
2) We present the novel internal architecture of RATS.
3) We outline lessons learned throughout our endeavor of developing RATS, as best practices for similar efforts.
4) We provide an elaborate empirical analysis on real performance data from tumor simulations executed on the MareNostrum 4 supercomputer, one of the most powerful HPC infrastructures in the European continent. The results illustrate the ability of our optimization approach to cherry pick proper simulation resource allocations and eventually cut down time to market for new therapies facilitating the transfer of promising in-silico trials to in-vitro and in-vivo ones.

II. Application Problem Description

TNF-based treatment methodologies are described by (i) the TNF Administration Frequency, which shows how often the drug is injected and is measured in minute intervals, (ii) the TNF Duration, dictating for how long the drug is administered, and (iii) the TNF Concentration each time the drug is applied. These parameters span continuous value ranges and therefore the number of possible simulations is infinite.

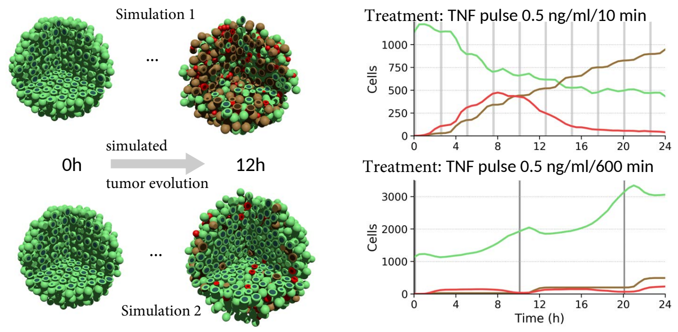


Fig. 1: Two examples of simulated tumors in PhysiBoSS. Tumors are visualized in various time instances (left). Green-colored cells are living tumor cells. Red and brown cells undergo apoptosis and necrosis, respectively. The number of cells per category (alive, necrosis, apoptosis) is also plotted (right). The horizontal axis of each time series is simulation wall time and the vertical axis the number of cells. Each time series' title mentions parameters of the applied drug treatment.

We first identify value ranges for the involved parameters that we want to study each time. We then discretize these parameter value ranges in a fine-tuned way to end up with a finite set of treatment methodologies under study. This number of potential treatment methodologies is equivalent to the total number of simulations that need to run. Hence, we have a separate $[low, high, step]$ triplet for each of the TNF Frequency (tnf_freq), TNF Duration (tnf_dur) and TNF Concentration (tnf_conc) parameters. The whole simulation set (SS) is described by all possible combinations of:

$\texttt{tnf\_freq} \in [\texttt{tnf\_freq\_low}, \texttt{tnf\_freq\_high}, \texttt{tnf\_freq\_step}]$
$\texttt{tnf\_dur} \in [\texttt{tnf\_dur\_low}, \texttt{tnf\_dur\_high}, \texttt{tnf\_dur\_step}]$
$\texttt{tnf\_conc} \in [\texttt{tnf\_conc\_low}, \texttt{tnf\_conc\_high}, \texttt{tnf\_conc\_step}]$

A. Preamble On Involved Optimization Problems

In Sections II-B and II-C we formulate 2 optimization problems. The first optimization stage is a prerequisite for RATS, because it helps in estimating the optimal core number per simulation, which directly relates to the total, optimal core number and core hours that are needed to submit the entire set SS for execution at once. If the first stage shows that core capacity constraints do not allow simultaneously submitting all simulations in SS, RATS proceeds to the second optimization stage which solves a Knapsack problem, accounting for core capacity constraints. Both the optimization problems do not directly optimize for core hours but, as we explain in Sections II-E, III, having derived the solutions of these stages, core hours can trivially be estimated. Table I summarizes the main symbols used in the paper.

B. Number of Cores per Simulation

The first optimization problem that we need to solve involves the number of cores that should be devoted to each individual simulation $S_i \in SS$ characterized by a specific $\{tnf\_freq,\ tnf\_dur,\ tnf\_conc\}$ triplet. Practically, we do not want to over-utilize or under-utilize cores for a simulation, but justly devote to it as many cores as it needs. Thus, for each individual simulation S_i we need to solve:

$$argmax\ THPRatio_i(k) = \frac{THP_i(k)}{THP_i(k/2)} \tag{1}$$

subject to $k \in ValCor = \{2, \dots, max\_cores\}$.

The above problem formulation says that the optimal number of cores, i.e., the optimal value for k for a given simulation $S_i \in SS$ representing a single $\{tnf\_freq, tnf\_dur, tnf\_conc\}$ combination, is the one that maximizes the throughput ratio (THPRatio) yielded when we choose to increase the number of cores from $k/2$ to k. Throughput (THP) is defined as the number of tuples being processed by the PhysiBoSS simulator, per time unit (secs) and there is an upper bound max_cores on the number of cores we allow to be devoted to a single simulation $S_i \in SS$.

This definition is quite intuitive and incorporates the practical observation that after a certain sweet spot, PhysiBoSS progressively stops exploiting parallelism since the biological processes it models commence to certain aggregation points (similar to AllReduce operations [7]). For instance, consider the case when a simulation provides a throughput of 10M tuples/sec with $k = 1$ core, 20M tuples/sec with $k = 2$ cores, 30M tuples/sec with $k = 4$ cores. Then, upon deciding to switch from 1 to 2 cores, the above throughput ratio is 2 and upon switching from to 2 to 4 cores, the respective ratio is 1.5. So the optimal core choice for this example is $k = 2$ because it constitutes the sweet spot where the benefits in terms of throughput increase are proportional to the increase in computational resources we devote.

Notice that, if we can derive the optimal number of cores per simulation, we can also extract the optimal, total number of cores for SS. As we explain in Section II-E we can also have an estimation about the execution time of SS. Therefore, we can compute the total core hours.

Although the above optimization problem is a prerequisite to have some say on the total, optimal number of cores, if the cardinality $|SS|$ of SS is high, requesting to reserve the total, optimal number of cores becomes infeasible. For instance, HPC technical guidelines pose upper bounds on the allowed number of cores [8]. In such a case, we need to also solve a different optimization problem which accounts for an upper bound on core capacity.

C. Simulation Time under Core Capacity Constraints

In this setup, we want to optimize (minimize) the total Simulation Time (ST) required for SS under given core capacity (cap) constraints. Roughly speaking, knowing the optimal ST and having cap as input to the problem at hand, we can then compute the total, optimal number of core hours.

TABLE I: Symbols used throughout the manuscript

Symbol	Explanation
tnf_freq	Frequency of TNF injection in minutes
tnf_dur	For how long TNF is administered each time
tnf_conc	TNF concentration each time
SS	All combinations of $\{tnf\_freq, tnf\_dur, tnf\_conc\}$
$\|SS\|$	Cardinality of the set SS
$SS_{done}(t)$	Completed simulations from SS at time t
S_i	Simulation $S_i \in SS$
k	Number of cores for a simulation
$ValCor$	Set of possible core configurations{2,...,max_cores}
S_i^k	Simulation $S_i \in SS$ being executed using k cores
max_cores	Maximum allowed number of cores for a simulation
$THP_i(k)$	Throughput (number of tuples being processed per time unit) using k cores for simulation S_i
UTL_i	Utility of simulation S_i assigned by the life scientist
$ST_i(k)$	Simulation Time for S_i executed with k cores
ST	Total simulation time for all simulations in SS
cap	Core capacity constraints
N	Sample simulation budget

We model this optimization problem as a Multiple-Choice Knapsack problem [9]. Let us explain why. We have a set of simulations SS composed of all possible $\{tnf\_freq,\ tnf\_dur,\ tnf\_conc\}$ triplets and each simulation can be executed with $k \in ValCor$. The total core capacity constraint cap is our knapsack and the multiple choice version of it comes from the fact that if we choose to execute S_i with a certain number of k' cores, i.e., $S_i^{k'} = \{tnf\_freq, tnf\_dur, tnf\_conc, k'\}$, we eliminate all other choices of k for S_i, i.e. we would not repeat the same simulation with different k since we already know the effect of the respective treatment. Moreover, each possible choice S_i^k entails a specific simulation time $ST_i(k)$, which is the value of that choice and k is its weight.

Simulations run in parallel. Taking that into consideration, we do not seek to optimize the sum of $ST_i(k)$, but instead reduce the maximum $ST_i(k)$ which defines the future timepoint at which all simulations will have completed. This optimization problem would be defined as $minimize \max_{S_i \in SS, k \in ValCor} ST_i(k) \cdot b_{ik}$, for $b_{ik} = 1$ only if we choose S_i^k, and zero otherwise. The traditional Knapsack definition involves a maximization problem and thus, we alter the above formulation to match a maximization problem, also negating $ST_i(k)$s.

$$maximize \min_{S_i \in SS \setminus SS_{done}(t), k \in ValCor} - ST_i(k) \cdot b_{ik} \tag{2}$$

subject to

$$\sum_{k \in ValCor} b_{ik} = 1, \forall S_i \in SS,$$
$$b_{ik} \in \{0,1\}, \forall S_i \in SS, \forall k \in ValCor,$$
$$\sum_{S_i \in SS \setminus SS_{done}(t), k \in ValCor} k \cdot b_{ik} \leq cap(t) \leq cap$$

Notice that at time t only a subset of SS can be included in the knapsack. The rest of the simulations that we need to run remain in a queue. At any given point t there is a $SS_{done}(t) \subseteq$

SS with completed simulations. Having $SS_{done}(t)$, we need to solve the optimization problem of Equation 2 again, this time having a Knapsack with capacity $cap(t)$. $cap(t)$ denotes the number of free/available cores at time t. This explains the temporal reference (t) in Equation 2.

D. Utility-based Prioritization

In order to abide by our goal of cutting time to market for new therapies, we want to transform SS into a utility-based priority queue based on the expected usefulness of a simulation, in terms of killing cancer cells. For this, the simulations should be ordered by decreasing utility order. $UTL_i \in [0, 1]$ denotes the utility of simulation $S_i \in SS$ (Table I), and every time $SS_{done}(t)$ frees resources, simulations will be chosen from $S_i \in SS \setminus SS_{done}(t)$ using that order.

E. Motivation for Bayesian Optimization

In order to solve the optimization problems we describe in Section II-B and Section II-C, we need to have accurate estimations for ST_i and THP_i involved in Equation 1 and Equation 2. Moreover, we also need UTL_i estimations for $S_i \in SS$ to transform SS into a utility-based priority queue.

Due to the complexity of the biological processes that are modeled, simulation studies under different $\{tnf\_freq, tnf\_dur, tnf\_conc, k\}$ parameters constitute a black-box, the behavior of which cannot be described using some analytic mathematical formula. We therefore resort to learning that behavior using Machine Learning (ML) tools.

The challenge we face is that for each new study we wish to conduct on treatment methodologies, we lack any training data. Therefore, it is necessary to carry out a small set of simulations and train ML models to learn the variables THP_i, ST_i, and UTL_i. We would like to keep this number of sample simulations to a minimum for two reasons:

- Simulations are time consuming and occupy system resources.
- UTL_i can only be determined by life scientists after a simulation is completed and vizualized (see Figure 1). Human experts can have the capacity of manually evaluating only few tens of sample simulations on the effect of respective treatment methodologies.

Note that ST_i, THP_i are system metrics that can be automatically derived using workload managers. In MareNostrum 4, we use Slurm (https://slurm.schedmd.com/) for that purpose.

To sum up, we have an unknown black-box function that characterizes the system's behavior and our objective is to learn this behavior in terms of THP_i, ST_i, and UTL_i using a limited number of samples, because the number of function evaluations (simulations in our case) is severely limited by time, cost and human effort. Bayesian Optimization (BO) is the most suitable ML approach for this scenario [10], [11].

BO optimizes an objective function f by iteratively evaluating the value of f in sampled points and constructing an estimate for the mean value of f over the set of all feasible points [11]. BO is composed of two parts: a) a statistical model used to estimate the objective function; b) an acquisition function used to efficiently sample the next points to be evaluated. A Gaussian Process Regression (GPR) approach is often followed to model the objective function. The Gaussian Process (GP) model used in the GPR approach serves as the statistical model of the optimization and provides a probability distribution that estimates the value of the objective function over the set of feasible points X [12], [13]. The model consists of a probability distribution over possible functions that fit the set of evaluated points. This distribution is updated with each new evaluation of the objective function. A GP is defined by a mean function m and a covariance function or kernel:

$$f(x) \sim N(m(x), kernel(x, x'))$$

The kernel of the Gaussian Process describes the smoothness of the distribution and specifies the covariance between the values of the objective function at different points. In other words, it quantifies the similarity of the objective function evaluations between nearby points. In particular, the kernel function determines the functions that are most likely under the GP prior distribution and, thus, incorporate prior beliefs we have about the objective function. In our study, due to the discrete nature of $\{tnf\_freq, tnf\_dur, tnf\_conc, k\}$, we use a Rational Quadratic Kernel [12].

The acquisition function allows us to select the next evaluation point in an informative manner, as it serves as a utility estimate for all feasible points. In particular, the acquisition function quantifies the contribution of the evaluation of the objective function to our estimation for each point. Commonly used acquisition functions are the Expected Improvement (EI), Probability of Improvement (PI), Lower Confidence Bound (LCB), Upper Confidence Bound (UCB) and Entropy Search (ES) [11]. Different acquisition functions estimate the importance of evaluating the objective function at potential points, taking into account different perspectives. They strike a balance between the exploration of unexplored regions and the exploitation of promising areas, considering the trade-off between these two aspects. Acquisition functions whose primary aim is the exploration of the parameter space select points for which the estimate of the objective function is of

Algorithm 1 Bayesian Optimization

```
1: Provide Sample Simulation Budget $N$
2: Place a Gaussian Process prior on $f$.
3: Evaluate $f$ at $n_o$ initial points. Set $n = n_0$. Update distribution based on initial evaluations.
4: while $n \leq N$ do
5:     Find $x_n$ that maximizes the acquisition function over $X$.
6:     Observe $y_n = f(x_n)$.
7:     Update posterior distribution of $f$ using all observed $f$ values.
8:     Increment $n$
9: end while
10: Return trained GPR model to get queried and provide estimations for $f$.
```

higher uncertainty, while those that aim at the exploitation of the space, sample points in which the expected mean value of the objective function is high.

Pseudo-code of the BO algorithm is shown in Algorithm 1. The initial step involves evaluating the objective function at n_0 randomly selected points from the feasible set. Subsequently, the acquisition function is employed to determine the next point to evaluate. This is done by selecting the point that maximizes the acquisition function. Once the next point is chosen, the objective function is evaluated at that point, and the posterior distribution is updated using all the observed values of f. The aforementioned process (sampling of next point $\rightarrow$ evaluation of objective function $\rightarrow$ update of posterior distribution) is repeated until we reach a declared number of total evaluated points N.

In our application, function evaluations correspond to runs of sample simulations. Additionally, we have 3 objective functions, namely throughput (THP), simulation time (ST) and simulation utility (UTL), while X is composed of $S_i \in SS$ for UTL and $\{S_i^k\}, \forall S_i \in SS, \forall k \in ValCor$ for ST, THP. For reasons we explain in Section IV, instead of using 3 acquisition functions for the 3 objective functions, we only use 2. Moreover, we set $n_0 = 0$ for each of them. We found out that this approach dramatically decreases the required number of sample simulations and we experimentally prove that this happens without practically sacrificing the accuracy of GPRs.

III. The RATS Resource Allocator

The RATS resource allocator we have developed is composed of two principal components: the RATS Modeler and the RATS Solver. These components have distinct internal architectures and serve different objectives.

A. The RATS Modeler

The RATS Modeler is used to build estimators for $THP_i(k)$, $ST_i(k)$ and UTL_i involved in Equations 1 and 2. The assumption is that RATS already has access to a minimum, small-scale core capacity, which is typically the case for life scientists employed by organizations hosting HPC infrastructures. To facilitate the resolution of the optimization problems stated in Equations 1 and 2, the RATS Modeler executes sample simulations in order to build GP regressors. This process iterates in loops, as illustrated in Figure 2.

Step 1: The RATS Modeler takes as input a set SS, which consists of triplets as described at the beginning of Section II. Additionally, it receives the budget N for sample simulations that it can execute to train GPRs using Bayesian Optimization. The values of the simulation budget N and the BO hyperparameters are determined based on the knowledge and experience gained during the development of RATS, which are discussed separately in Section IV.

Step 2: The acquisition function for the Throughput Regressor asks for a sample simulation S_i^k to get executed. The same applies for the acquisition function for the Simulation Time Regressor. All BO-based Regressors and their acquisition functions are included in a separate subcomponent termed, Bayesian Optimization Modeler, on the right of Figure 2. The BO Modeler interacts with a Benchmarker subcomponent asking for this pair of simulations to get executed.

Step 3: The Benchmarker subcomponent (bottom of Figure 2) creates a pair of XML files that configure the respective runs of the PhysiBoSS simulator and submits each of this pair of jobs (left of Figure 2).

Step 4: Upon the completion of a submitted simulation there are two types of outputs: (a) system metrics regarding the simulation time $ST_i(k)$ and $THP_i(k)$ and (b) visualization of the simulated tumor under the $S_i \in SS$ treatment methodology used. Simulation visualizations are presented to the life scientist who assigns a value in $[0, 1]$ for the utility UTL_i of $S_i \in SS$. This happens for the simulations, either initially requested by the Throughput Regressor or the Simulation Time Regressor.

Step 5: Both system measured and human-assigned metrics are collected by a Statistics Collector subcomponent (middle of Figure 2). The Statistics collector feeds system metrics to the Simulation Time and Throughput Regressors, respectively. It also feeds the utility value to the Utility Regressor which, on the one hand, does not have its own acquisition function, on the other hand receives utility value input for all simulations requested by either of the other 2 acquisition functions.

The above steps are repeated until the available simulation budget N is depleted. One could potentially train the Throughput Regressor and then proceed with training the Simulation Time Regressor. Moreover, sample simulations that are asked by one acquisition function and have already been requested and executed due to the other can be directly fed to the BO Modeler by having the Benchmarker and Statistics collector subcomponents keeping a recent history of S_i^ks and statistics, respectively. In that case, we do not increase the counter at Line 7 of Algorithm 1. All these optimizations are orthogonal to the procedure we describe above.

B. The RATS Solver

The RATS Solver component comes into play after the RATS Modeler has built its estimators. The RATS Solver focuses on (a) prioritizing simulations based on their expected utility, (b) solving the optimization problems defined in Equations 1 and 2 using predictions provided by the regressors of the RATS Modeler. It is important to highlight that the RATS Solver solely relies on the RATS Modeler to obtain estimations of THP_i, ST_i and UTL_i. It does not utilize any HPC resources whatsoever. The Solver operates as follows:

Part 1: The RATS Solver receives as input the set SS in text format as the RATS Modeler does. It also receives the core capacity constraint *cap*. It then converts SS into a priority queue by sorting the text tuples based on the expected utility of each S_i. For that purpose, the Solver derives the expected utility of each $S_i \in SS$ by querying the Utility Regressor previously built by the RATS Modeler. This is illustrated on the left-hand side of Figure 3. Then, it replicates the sorted tuples adding possible core numbers k, the throughput $THPRatio_i(k)$ of which we want to estimate (not shown in

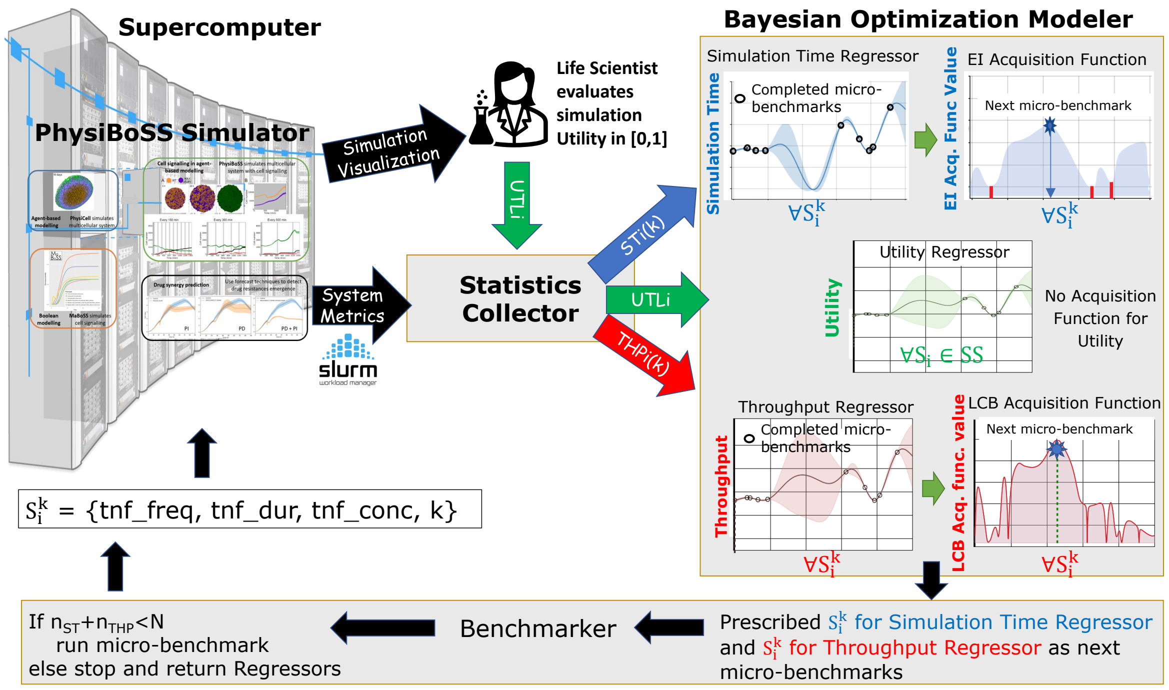


Fig. 2: Overview of the RATS Modeler.

the figure for readability purposes). These are also the items we consider in our Knapsack (if needed).

Part 2: The RATS Solver solves Equation 1 deriving the optimal number of cores for each and every $S_i \in SS$. To compute $THPRatio_i(k)$ for $S_i \in SS$, it queries the Throughput Regressor of the RATS Modeler (middle of Figure 3). The optimal number of cores per simulation is summed up to compute the total, optimal number of cores for SS.

Part 3: If this total number is smaller than *cap*, all simulations can be submitted at once. Then, the RATS Solver queries the Simulation Time Regressor of the RATS Modeler to find out the estimated total simulation time, based on which it computes the total core hours and returns it to the life scientist. In that case, the RATS Solver stops. This is shown in Figure 3 using red-colored arrows. Otherwise it proceeds to Part 4.

Part 4: If the capacity (*cap*) constraint prevents the submission of the set SS with the optimal number of cores all at once, the RATS Solver may continue to solving the optimization problem in Equation 2 as shown with the blue-colored arrows and icons in Figure 3. Recall that this problem is solved in rounds. In each round, as described in Section II-C some S_i^ks from the utility-based priority queue are selected to be included in the Knapsack so that the minimum value of the included simulations (negated simulation times ST_i) is maximized. In that case, the RATS Solver queries the Simulation Time Regressor of the RATS Modeler to obtain estimations about $ST_i(k)$ at the beginning of this step, but also at any subsequent time t upon there are simulations entering $SS_{done}(t)$ and freeing resources to $cap(t)$.

Final Part: If the priority queue is empty at some point, the execution of the RATS Solver concludes and the number of core hours is returned as the product of *cap* and the *aggregated* minimum execution time from solving the Knapsack at the various timepoints t.

IV. Insights and Lessons Learned

Based on the experience gained from performing hyperparameter tuning of RATS, we have learned several valuable lessons. These lessons can be summarized as follows:

Kernel Selection: The kernel that provides the best estimation performance is the Rational Quadratic kernel parameterized by a length scale of 1.0 and α parameter of 2.5. Please refer to [12] for further details on these parameters.

Simulation Budget: The total simulation budget N for training the involved regressors can be set to a value between 5% and 10% of the cardinality of SS. In other words, the sample simulations are 10 to 20 times fewer compared to $|SS|$.

No initial set n_0: Although the traditional BO, as described in Algorithm 1, entails a number of simulations n_0 in a warmup phase, our practical experience says that having set the kernel function as described above, this n_0 does not aid significantly in improving the final accuracy of the regressors. Therefore, we advise that the whole experimental budget is used with sample simulations driven by the respective acquisition functions.

Acquisition Function Choice: For training the Throughput Regressor of the RATS Modeler, an LCB acquisition function should be used. LCB favors exploration vs exploitation (Section II-E) and better captures trends in throughput instead of absolute values [14]. This seems to favor accurate estimations of a ratio, such as $THPRatio_i(k)$. The acquisition function

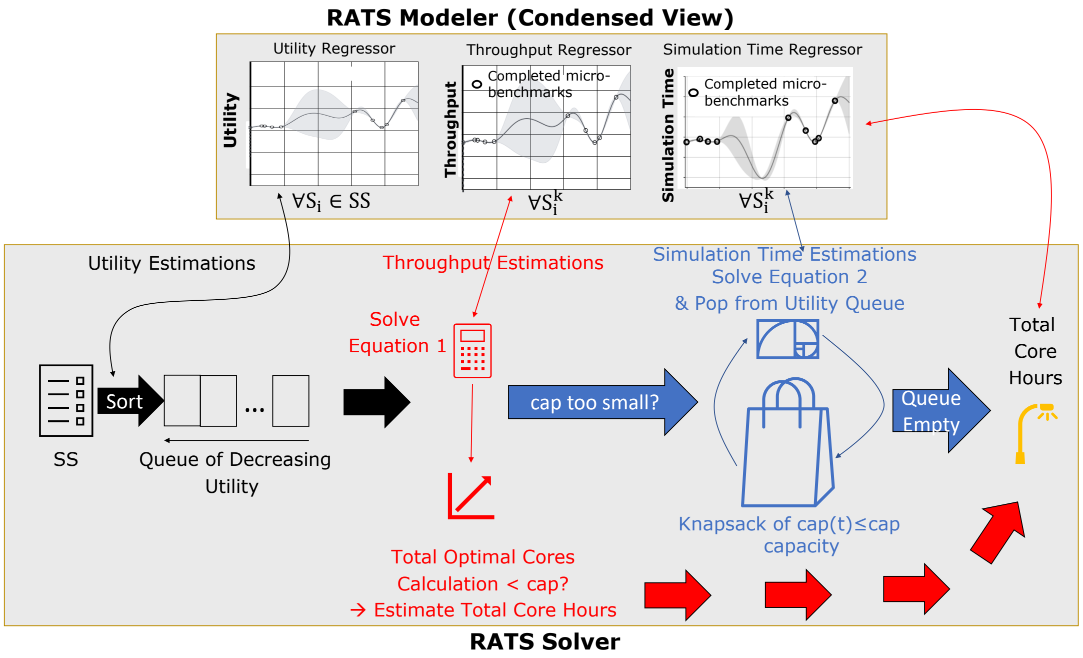

RATS Modeler (Condensed View)
Utility Regressor
Throughput Regressor
Simulation Time Regressor
Utility
Throughput
Simulation Time
Completed micro-benchmarks
Completed micro-benchmarks
∀S_i ∈ SS
∀S_i^k
∀S_i^k
Utility Estimations
Throughput Estimations
Simulation Time Estimations
Solve Equation 2
& Pop from Utility Queue
Solve Equation 1
Total Core Hours
Sort
SS
Queue of Decreasing Utility
cap too small?
Queue Empty
Knapsack of cap(t)≤cap capacity
Total Optimal Cores Calculation < cap?
→ Estimate Total Core Hours
RATS Solver

Fig. 3: RATS Architecture - RATS Solver and Built RATS Modeler.

of choice for the Simulation Time Regressor is EI instead, because now we are interested in absolute values. In both functions, the ξ_i and κ parameters respectively, should be set to a high value above 100 tuning the exploration vs exploitation trade-off. Please refer to [11] for further details.

V. Related Work

There is a line of work that focuses on pruning the size of SS by predicting and distinguishing treatment parameters that yield useful simulations [15]–[19]. All these approaches are orthogonal to RATS and can be used in order to prune simulations of expected low utility from the set SS, thus reducing the size of the priority queue. RATS stands out from these approaches by effectively providing the necessary core hours to life scientists, which cannot be achieved by any of the alternative methods.

A different, relevant line of work comes from workload optimization over cluster, database and Big Data infrastructures [20]. EasyFlinkCEP [14] employs Bayesian Optimization to auto-tune the optimal parallelism of FlinkCEP programs. Wayeb [21] applies BO to find (near-)optimal training configurations for the Complex Event Forecasting Engine. Seagull [22] uses machine learning models to predict customer load per server, and optimize service operations for database systems in the cloud. CBTune [23] uses reinforcement learning and utilizes a deep deterministic policy gradient method to find the optimal database instance configurations in high-dimensional continuous spaces adopting a try-and-error strategy to learn knob settings. While all these approaches primarily emphasize Big Data and data management analytics, RATS sets itself apart by specifically catering to the more complex case of modelling and predicting simulated biological processes. RATS effectively synthesizes multiple BO models and builds them side-by-side leveraging shared samples. It also incorporates the RATS Solver, which allows life scientists to observe the entire lifespan of the study on the desired treatment methodology set SS, in advance.

Finally, Kunjir and Babu [24] compare black box methods, including BO and Deep Distributed Policy Gradient, against a proposed white-box algorithm to determine close-to-optimal tuning for memory-based analytics. Such an approach is orthogonal to RATS to make it also account for memory consumption.

VI. Experimental Evaluation

RATS is implemented in Python also exploiting scikit-optimize (https://scikit-optimize.github.io/) for the RATS Modeler. The code of the PhysiBoSS 2.0 simulator is available online (https://github.com/PhysiBoSS/PhysiBoSS). To test the performance of RATS, we collect real data by executing 2560 simulations on the MareNostrum 4 supercomputer, one of the most powerful supercomputers at the European continent level. These simulations are composed of $|SS| = 512$ TNF combinations with $[low, high, step]$ triplets (see Section II):
`tnf_freq` ∈ `[100, 450, 50]`
`tnf_dur` ∈ `[10, 45, 5]`
`tnf_conc` ∈ `[0.01, 0.045, 0.005]`
and core configurations per simulation up to $max\_core = 32$. We emphasize that we do not use all 2560 simulations to train the RATS Modeler. We utilize the complete set of simulations as the ground truth dataset, encompassing all possible simulations and core configurations. This dataset serves as a benchmark to evaluate and compare the performance of RATS against the ground truth system and simulation utility statistics.

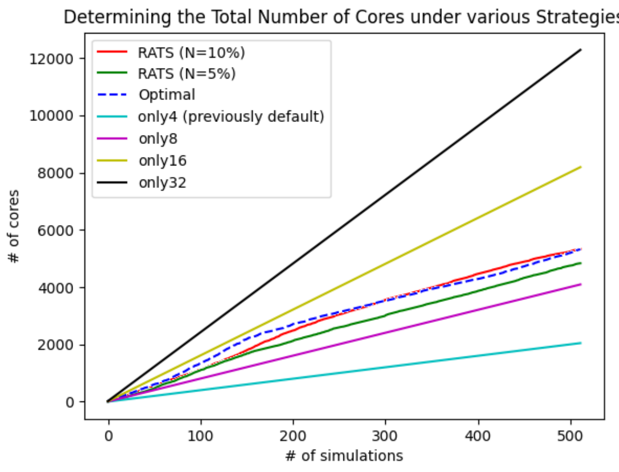


Fig. 4: Determining the Total Number of Cores applying Part 1 and Part 2 of the RAT Solver, having trained the RAT Modeler using 5% and 10% of all possible simulation configurations $\{tnf\_freq, tnf\_dur, tnf\_conc, k\}$. Naive assignment using only 4, 8, 16, 32 cores to each simulation are included.

RATS is configured according to the best practices we outline in Section IV and we test two different versions of RATS operating with simulation budgets of $N = 5\%$ and $N = 10\%$, respectively. These are labeled as "RATS (N=5%)" and "RATS (N=10%)" in our graphs. We compare RATS against a series of competitors which we term "only4", "only8", "only16", "only32" which constitute manual simulation core configurations by having the life scientist executing each and every simulation with 4,8,16,32 cores, respectively. We begin our search with "only4" because this is what we would use as the default number of cores per simulation, before the development of RATS. There is also the "Optimal" competitor in our graphs which computes the optimal number of cores, simulation time and utility based on the entire, ground truth dataset. In our graphs we note the "previously default" property of "only4" just once in Figure 4 and omit it from the rest of the graphs, for readability purposes. To speed up the execution of the RATS Solver, we limit the number of Knapsack solutions that are explored, each time $cap(t)$ cores are available, to 100K.

Main Findings: Before presenting our experimental evaluation in detail, we outline our main findings. When the entire simulation set can be submitted at once to the supercomputer (red-colored path in Figure 3), "RATS (N=10%)" approximates the optimal core hours within a 9% error in the worst case and below 3% in the best. When the RATS Modeler is trained with fewer samples ($N = 5\%$) this error increases reaching 10% to 18%. Under core capacity constraints (blue-colored path in Figure 3) RATS coincides with the "Optimal" approach in all the cited cases: (i) when severe capacity constraints exist, which we model by setting $cap = 10\%|SS|$, RATS reduces the total simulation time by up to 4 days and the total core hours by up to 15% compared to the second best (manual configuration) competitor. RATS behavior coincides with what "Optimal" does, (ii) when capacity constraints are looser ($cap = 50\%|SS|$), RATS reduces the total simulation time by up to 10 hours and the total core hours by up to 12% compared to the second best (manual configuration) competitor, again coinciding with the "Optimal". Furthermore, in both scenarios (i) and (ii), RATS holds an advantage by prioritizing simulations according to their utility, which is the value assigned by the life scientist within the range $[0, 1]$. Over the entire lifespan of the simulation, RATS achieves an aggregate utility that is up to 36% higher compared to the closest competitor, making it the superior choice. As a result, RATS not only accurately assigns near-optimal values to the required core hours for a treatment methodology study, but also substantially accelerates the completion of useful simulations. This gives life scientists the important option to conclude the study before the execution of the entire SS.

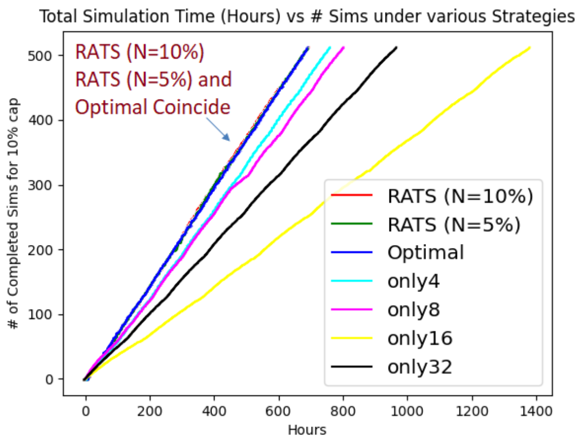
TABLE II: Error in Core Hours under No Capacity Constraints ($-/+$ stands for under-/over- estimation respectively)

#Simulations	RATS (N=5%)	RATS (N=10%)
100	-10.33%	-8.78%
200	-14.48%	-8.36%
300	-17.12%	2.26%
400	-13.08%	4.10%
500	-13.32%	2.53%

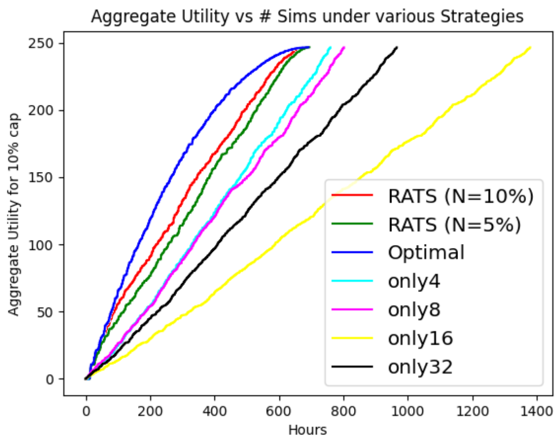
A. *Total Optimal Number of Core Hours for the entire SS*

In this first set of experiments we evaluate the performance of RATS when the entire set SS can be submitted for execution simultaneously (red-colored path in Figure 3). Figure 4 plots the number of cores devised by RATS vs the number of simulations one may submit to the supercomputer. RATS is tested under two different configurations: one that spends a 5% budget on training the RATS Modeler and a second one with N set to 10% (green and red lines in the figure, respectively). RATS is compared with manual competitors, as we discussed above, and the blue-colored line corresponds to the "Optimal" total number of cores across various numbers of simulations. As illustrated in Figure 4, "RATS (N=10%)" almost perfectly fits on the "Optimal" plot line, while "RATS (N=5%)" emerges as the second-best alternative, slightly under-estimating the necessary number of cores. The best manual configuration approach is "only 8" which often under-utilizes more than 500 and up to 700 cores. All other manual approaches are impractical since they excessively over-utilize thousands of cores (for "only 16" and "only 32") or severely under-utilize resources (for "only 4").

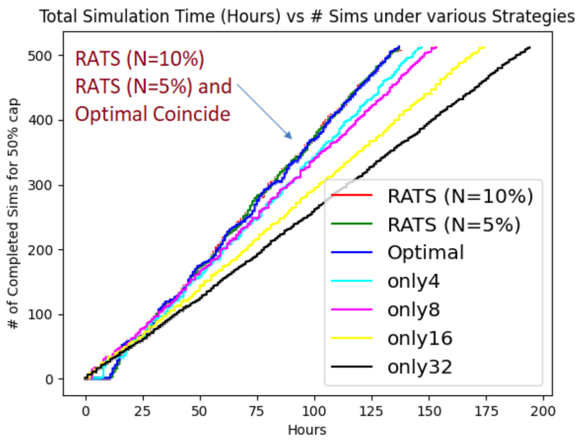
Now, following the process in the red-colored path of Figure 3, we use the number of cores devised by "RATS (N=10%)" or "RATS (N=5%)" to derive an accurate estimation about the total core hours required for SS. As can be deduced by Table II, "RATS (N=10%)" approximates the optimal core hours within a 3% to 9% error in the worst case. When the RATS Modeler is trained with fewer samples ($N = 5\%$) this



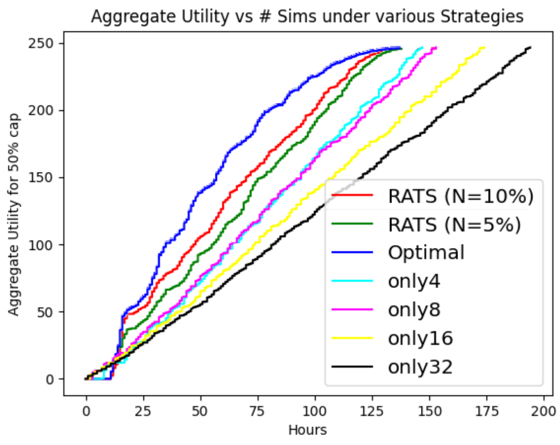
(a) Number of Simulations vs Time (Hours), cap = 10% $|SS|$.



(b) Aggregate utility vs Time (Hours), cap = 10% $|SS|$.



(c) Number of Simulations vs Time (Hours), cap = 50% $|SS|$.



(d) Aggregate Utility vs Time (Hours), cap = 50% $|SS|$.

Fig. 5: RATS Solver Performance in Simulation Time and Aggregate Utility. Comparison against various Competitor Strategies under various Constraint and Sample Simulation Budget N Configurations.

error increases reaching 10% to 18%. An interesting observation arises from the scenario where the simulation set can be submitted at once. In such cases, it becomes advantageous for the life scientist to dedicate additional effort to train the RATS Modeler. This upfront investment in training pays off as evidenced by the performance of "RATS (N=10%)", which achieves highly accurate optimization results. As we show in the next plots, if capacity constraints do not allow to submit $|SS|$ at once, "RATS (N=5%)" may be preferable instead.

B. RATS under Capacity Constraints

When the entire set of SS cannot be submitted at once, the RATS solver follows the blue-colored path in Figure 3. In such cases, simulations are retrieved from the utility-based priority queue and the Knapsack problem is solved at any given time t so that the number of simulations that run at t satisfies the constraint $cap \geq cap(t)$. The process concludes when the priority queue becomes empty, unless the life scientist chooses to halt the Solver before completion for reasons we explain below. It is important to recall that the RATS Solver itself does not perform the actual execution of simulations. Instead, it relies on the estimations provided by the trained RATS Modeler to predict simulation times and utility metrics. The manual approaches in this experiment simply execute simulations without prioritizing them.

Figure 5 plots the number of completed simulations and the aggregate utility as more simulations are completed vs the total simulation time in hours. Each row of plots should be examined side by side since they offer complementary information that enhances understanding. In the first row, Figure 5a illustrates the progression of completed simulations over time, while Figure 5b displays the aggregate utility of the simulations that have been completed up to each specific timestamp. It provides a visualization of how the aggregate utility evolves over time as more simulations are executed and completed. Both figures are generated under the scenario of severe core capacity constraints, modeled by setting $cap = 10\%|SS|$. Figure 5c and Figure 5d repeat this experiment, but this time under loose core capacity constraints modeled by setting $cap = 50\%|SS|$. By examining each row of the plot, one can deduce not only the speed at which a specific number of simulations is completed but also the level of usefulness in terms of their effectiveness in killing cancer cells.

Throughout the entire set of experiments depicted in Figure 5, both the "RATS(N=10%)" and "RATS(N=5%)" lines align perfectly with the "Optimal" one. The second best competitor is the "only4" manual configuration approach, which practically means that it may be more preferable to execute more simulations with fewer cores under given capacity constraints, than few simulations with more cores attributed to each. Still, both versions of RATS in Figure 5a, reduce the total simulation time by up to 4 days and therefore the total core hours (as the total simulation time across the horizontal axis multiplied by cap) by 15% compared to "only 4" for $cap = 10\%|SS|$. This is because RATS, at any given time across the horizontal time axis, elects for execution the set of simulations that minimize the maximum execution time by solving the Knapsack problem at time t. The simulations that are completed at any given time, based on RATS's prioritization, exhibit significantly higher aggregate utility as illustrated in Figure 5b. Similarly, under looser capacity constraints, $cap = 50\%|SS|$, RATS reduces the total simulation time by up to 10 hours and the total core hours by up to 12% compared to "only4" also providing higher aggregate utility up to 36%. Therefore, based on the results of Figure 5b and Figure 5d, the life scientists may decide not to execute the entire set SS, but restrict themselves to the top-K most useful simulations. Based on the graphs of Figure 5, RATS will dictate the nearly-optimal number of core hours in every such case. When capacity constraints are in place, both "RATS (N=5%)" and "RATS (N=10%)" demonstrate comparable performance in terms of total simulation time. As a result, if the life scientist is willing to accept a slightly lower aggregate utility, we may utilize fewer samples for training the RATS Modeler.

VII. Conclusions & Future Work

RATS takes away from life scientists the burden of determining the total core hours for in-silico studies on tumor treatment methodologies and automates the system configurations for such trials. This fact, combined with the prioritization of simulations based on their expected utility, helps in more rapidly distinguishing effective drug combinations, cutting time to market for new cancer therapies. Our future work is on extending RATS to more complex treatment methodologies and on integrating it with relevant frameworks [15].

Acknowledgment

E. Streviniotis, N. Giatrakos and Y. Kotidis were supported by the EU project EVENFLOW under Horizon Europe agreement No. 101070430. T. Ntiniakou and M. Ponce de Leon were supported by the EU project CREXDATA under Horizon Europe agreement No. 101092749.

References

[1] M. P. de Leon, A. Montagud, V. Noël, G. Pradas, A. Meert, E. Barillot, L. Calzone, and A. Valencia, "Physiboss 2.0: a sustainable integration of stochastic boolean and agent-based modelling frameworks," *bioRxiv*, 2023. [Online]. Available: https://www.biorxiv.org/content/early/2023/03/27/2022.01.06.468363

[2] Y. A. Fouad and C. Aanei, "Revisiting the hallmarks of cancer," *Am J Cancer Res*, vol. 7, no. 5, pp. 1016–1036, May 2017.

[3] S. M. Shaffer, M. C. Dunagin, S. R. Torborg, and et al, "Rare cell variability and drug-induced reprogramming as a mode of cancer drug resistance," *Nature*, vol. 546, no. 7658, pp. 431–435, Jun 2017.

[4] N. Giatrakos, N. Katzouris, A. Deligiannakis, and et al, "Interactive extreme: Scale analytics towards battling cancer," *IEEE Technol. Soc. Mag.*, vol. 38, no. 2, pp. 54–61, 2019.

[5] J. Li, Q. Yin, and H. Wu, "Chapter five - structural basis of signal transduction in the tnf receptor superfamily," ser. Advances in Immunology, F. W. Alt, Ed. Academic Press, 2013, vol. 119, pp. 135–153.

[6] P. Bloomingdale, V. A. Nguyên, J. Niu, and D. E. Mager, "Boolean network modeling in systems pharmacology," *Journal of Pharmacokinetics and Pharmacodynamics*, vol. 45, pp. 159–180, 2018.

[7] "Openmpi software documentation," (Accessed: 12 May 2023). [Online]. Available: https://www.open-mpi.org/doc/v4.0/man3/MPI_Allreduce.3.php

[8] "Regular access call – technical guidelines - prace-ri.eu." (Accessed: 12 May 2023). [Online]. Available: https://prace-ri.eu/wp-content/uploads/Technical_Guidelines-Regular_Access.pdf

[9] K. DudziÅski and S. Walukiewicz, "Exact methods for the knapsack problem and its generalizations," *European Journal of Operational Research*, vol. 28, no. 1, pp. 3–21, 1987.

[10] D. R. Jones, M. Schonlau, and W. J. Welch, "Efficient global optimization of expensive black-box functions," *J. Glob. Optim.*, vol. 13, no. 4, pp. 455–492, 1998.

[11] E. Brochu, V. M. Cora, and N. de Freitas, "A tutorial on bayesian optimization of expensive cost functions, with application to active user modeling and hierarchical reinforcement learning," 2010.

[12] C. E. Rasmussen and C. K. I. Williams, *Gaussian processes for machine learning*, ser. Adaptive Computation and Machine Learning series. MIT Press, 2005.

[13] J. Wang, "An intuitive tutorial to gaussian processes regression," *CoRR*, vol. abs/2009.10862, 2020. [Online]. Available: https://arxiv.org/abs/2009.10862

[14] N. Giatrakos, E. Kougioumtzi, A. Kontaxakis, A. Deligiannakis, and Y. Kotidis, "Easyflinkcep: Big event data analytics for everyone," in *CIKM*, 2021.

[15] C. Akasiadis, M. P. de Leon, A. Montagud, E. Michelioudakis, A. Atsidakou, E. Alevizos, A. Artikis, A. Valencia, and G. Paliouras, "Parallel model exploration for tumor treatment simulations," *Comput. Intell.*, vol. 38, no. 4, pp. 1379–1401, 2022.

[16] B. K. Lind, P. Mavroidis, S. Hyödynmaa, and C. Kappas, "Optimization of the dose level for a given treatment plan to maximize the complication-free tumor cure." *Acta oncologica*, vol. 38 6, pp. 787–98, 1999.

[17] F. S. Lobato, V. S. Machado, and V. Steffen, "Determination of an optimal control strategy for drug administration in tumor treatment using multi-objective optimization differential evolution," *Computer methods and programs in biomedicine*, vol. 131, pp. 51–61, 2016.

[18] R. Mohan, G. S. Mageras, B. Baldwin, L. J. Brewster, G. J. Kutcher, S. A. Leibel, C. Burman, C. C. Ling, and Z. Fuks, "Clinically relevant optimization of 3-d conformal treatments." *Medical physics*, vol. 19 4, pp. 933–44, 1992.

[19] N. Jagiella, D. Rickert, F. J. Theis, and J. Hasenauer, "Parallelization and high-performance computing enables automated statistical inference of multi-scale models." *Cell systems*, vol. 4 2, pp. 194–206.e9, 2017.

[20] D. Tsesmelis and A. Simitsis, "Database optimizers in the era of learning," in *IEEE ICDE*, 2022.

[21] V. Stavropoulos, E. Alevizos, N. Giatrakos, and A. Artikis, "Optimizing complex event forecasting," in *DEBS*, 2022.

[22] O. Poppe, T. Amuneke, D. Banda, and et al, "Seagull: An infrastructure for load prediction and optimized resource allocation," *Proc. VLDB Endow.*, vol. 14, no. 2, pp. 154–162, 2020.

[23] J. Zhang, Y. Liu, K. Zhou, G. Li, Z. Xiao, B. Cheng, J. Xing, Y. Wang, T. Cheng, L. Liu, M. Ran, and Z. Li, "An end-to-end automatic cloud database tuning system using deep reinforcement learning," in *SIGMOD*, 2019.

[24] M. Kunjir and S. Babu, "Black or white? how to develop an autotuner for memory-based analytics," in *SIGMOD*, D. Maier, R. Pottinger, A. Doan, W. Tan, A. Alawini, and H. Q. Ngo, Eds., 2020.