Engineering Inhibitory Proteins with InSiPS: The In-Silico Protein Synthesizer *

Andrew Schoenrock
School of Computer Science
Carleton University
Ottawa, Canada
aschoenr@scs.carleton.ca

Daniel Burnside
Department of Biology
Carleton University
Ottawa, Canada
daniel.burnside@carleton.ca

Houman Moteshareie
Department of Biology
Carleton University
Ottawa, Canada
houman.moteshareie@carleton.ca

Alex Wong
Department of Biology
Carleton University
Ottawa, Canada
alex.wong@carleton.ca

Ashkan Golshani
Department of Biology
Carleton University
Ottawa, Canada
ashkan.golshani@carleton.ca

Frank Dehne
School of Computer Science
Carleton University
Ottawa, Canada
frank@dehne.net

ABSTRACT

Engineered proteins are synthetic novel proteins (not found in nature) that are designed to fulfill a predetermined biological function. Such proteins can be used as molecular markers, inhibitory agents, or drugs. For example, a synthetic protein could bind to a critical protein of a pathogen, thereby inhibiting the function of the target protein and potentially reducing the impact of the pathogen. In this paper we present the *In-Silico Protein Synthesizer (InSiPS)*, a massively parallel computational tool for the IBM Blue Gene/Q that is aimed at designing inhibitory proteins. More precisely, InSiPS designs proteins that are predicted to interact with a given target protein (and may inhibit the target's cellular functions) while leaving non-target proteins unaffected (to minimize side-effects). As proof-of-concepts, two InSiPS designed proteins have been synthesized in the lab and their inhibitory properties have been experimentally verified through wet-lab experimentation.

Keywords

Applications, Bioinformatics and Computational Biology, Computational Medicine and Bioengineering, Synthetic Biology, Protein Design

1. INTRODUCTION

Protein engineering refers to an expansive field of research that strives to design functionally optimized proteins. The discipline can be divided into two primary approaches: the modification of existing proteins to improve or amend endogenous function [4], and *de novo* protein design, which generates novel proteins optimized for specific function(s) [17]. Protein engineering can be considered the reverse problem of protein function prediction. In function prediction, one is given a protein sequence and tasked with determining the protein function, for example by approximating its three dimensional structure (protein folding) and determining the protein's interactions with other proteins (protein docking). In protein engineering, we are given a desired function of a yet unknown protein, and the task is to generate a sequence for a protein with such properties.

The function of a protein A is mediated by its interactions with other proteins, i.e. set of *target* proteins with which A interacts and the set of *non-target* proteins with which A does not interact [14]. Designed proteins with a given set of target and non-target proteins can be used as *regulatory* proteins that enhance or inhibit the function of their target protein(s). Designing proteins that can bind a target protein in a regulatory capacity can also accomplish the goal of modifying the structure of the target protein. Of particular interest is the design of *inhibitory proteins* which bind to a single target protein with high specificity (i.e. avoiding non-target proteins), thereby preventing the target from performing its normal biological function [1, 5]. For example, a designed inhibitory protein could attach itself to a critical protein of a pathogen, thereby inhibiting the function of that target protein and potentially reducing the impact of the pathogen. Such inhibitory proteins can form the basis for the development of various therapeutics (e.g. [15]).

A variety of approaches have contributed to the field of protein design over the past decades; e.g. [7, 10, 9, 16]. Recent computational methods have experienced mixed success in experimental validation, with most methods redesigning naturally occurring protein folds rather than designing novel structures *ab initio* [2]. Three-dimensional structure determination often plays a central role in protein design. The function of a protein is highly dependent on its 3D structure since most protein functions, in particular interactions with substrates and with other proteins, are mediated by physical contacts in 3D space. However, a major downfall of such approaches is a lack of known 3D structures for the majority of naturally-occurring proteins. This restricts re-

*Research partially supported by the Natural Sciences and Engineering Research Council of Canada (NSERC) and the Southern Ontario Smart Computing Innovation Platform (SOSCIP).

SC '15, November 15-20, 2015, Austin, TX, USA

DOI: http://dx.doi.org/10.1145/2807591.2807630

searchers to a limited number of highly studied proteins [8]. Protein structure prediction remains a difficult problem to solve, particularly when structures of similar proteins are not available. This motivates the creation of a protein design method that does not rely on protein 3D structures.

In this paper we present the *In-Silico Protein Synthesizer (InSiPS)*, a massively parallel computational tool for the IBM Blue Gene/Q that is aimed at designing inhibitory proteins. InSiPS designs proteins that are predicted to interact with a given target protein while leaving a given set of non-target proteins unaffected. A typical use of InSiPS for designing an inhibitory protein for a given target protein (e.g. a critical protein of a pathogen) would be to define the set of non-target proteins as "all other" proteins to avoid side-effects.

InSiPS requires as input only the sequences of the target and non-target proteins and a database of known protein-protein interactions. Unlike other approaches to protein design, InSiPS is purely sequence-based and does not require knowledge of 3D protein structures. The InSiPS algorithm is a combination of a genetic algorithm with a protein interaction prediction method that is based on mining large databases of known protein-protein interactions. Tens of thousands of experimentally-validated, high confidence binary protein-protein interactions have been reported, and have been curated in a variety of databases [3, 12, 13]. These databases form the basis for our InSiPS method and allow InSiPS to be applied to a broad range of protein design problems. The computational requirements for InSiPS are very large. InSiPS is a highly scalable parallel method and has been implemented for massively parallel IBM Blue Gene/Q platforms.

Computational methods that predict biological interactions can only be deemed credible when supported by targeted *in vivo* or *in vitro* experiments. In a living cell, proteins perform a dynamic array of functions but do so while interacting with other proteins and cellular components. As proof-of-concept, we performed wet-lab experiments for two InSiPS designed proteins "anti-YBL051C" and "anti-YAL017W" targeting yeast proteins YBL051C and YAL017W, respectively. For both experiments, the set of non-target proteins was the set of all other yeast proteins in the same cellular component. Baker's yeast *Saccharomyces cerevisiae* is a well-studied and often used eukaryotic model organism. We synthesized our two novel proteins, anti-YBL051C and anti-YAL017W, and then exposed living yeast cells to these proteins. Our wet-lab experiments show that in living yeast cells, our InSiPS designed synthetic proteins anti-YBL051C and anti-YAL017W do indeed have an inhibitory effect on YBL051C and YAL017W, respectively.

2. IN-SILICO PROTEIN SYNTHESIZER (INSIPS)

2.1 Algorithm Overview

The In-Silico Protein Synthesizer (InSiPS) is a massively parallel computational tool designed to produce proteins with specific protein-protein interaction profiles. More specifically, given a target protein and a set of non-target proteins, InSiPS aims to design a novel protein sequence which is predicted to interact with the target protein and predicted not to interact with the non-target proteins. For a given target protein, the non-targets typically include the other proteins which are present in the same cellular component as the target (to which binding is not desired).

InSiPS uses a genetic algorithm to produce a protein sequence which optimizes a fitness function. For a given candidate protein sequence, this fitness function uses a protein-protein interaction prediction algorithm to determine whether the candidate is likely to interact with the target and non-targets. The predicted interaction scores are combined to give the candidate sequence an overall fitness. Our InSiPS fitness function is fully detailed in Section 2.2. It assigns each sequence a score between 0 and 1, with a higher fitness representing a better solution (i.e. more likely to interact/not interact with the target/non-targets than sequences with a lower fitness score). InSiPS begins by generating a predetermined number of random protein sequences. This set of random sequences represents its initial population of candidate solutions. Any set of protein sequences can be used as a starting population; however, to remove any forms of bias, a randomly generated set of sequences is recommended. The algorithm then repeatedly executes the following two steps; see Figure 1 for an illustration.

(1) *Evaluation of candidate solutions (fitness function).* For each sequence in the current generation, a prediction is made whether the sequence interacts with the target and non-target proteins. These prediction scores are then converted into a fitness function. Details are described in Section 2.2.

(2) *Construction of the next generation.* To build the next generation of protein sequences, sequences are randomly selected with a probability proportional to their fitness relative to the rest of the population. Three standard operations *copy*, *mutate* and *crossover* are applied to these sequences with probability *p_copy*, *p_mutate* and *p_crossover*, respectively. *Copy:* The chosen sequence is simply copied into the next generation. *Mutate:* Each amino acid in the chosen sequence is randomly mutated with a predetermined probability (*p_mutate_aa*). Typically this probability is relatively low as to preserve most of the current state of the protein. Note that, while each amino acid has the same initial mutation probability, the final mutation probabilities are different due to fitness selection. Even though all spot mutations are equally likely, favourable mutations will be readily accepted and unfavourable mutations will be rejected by the fitness function, meaning these unfavourable mutations have a slim chance in participating in future generations. *Crossover:* For two chosen sequences A and B, a cut-point in the protein sequences is randomly chosen, ensuring it is not too close to either end. The first portion of sequence A is then joined with the second portion of sequence B, and the first portion of sequence B is joined to the second portion of protein A to create two new hybrid sequences for the next generation.

The main contribution of this research lies in the fitness evaluation function discussed in the following Section 2.2 and the parallel implementation for Blue Gene/Q outlined is Section 2.3. To the authors' knowledge, InSiPS is the first successfully demonstrated sequence-based protein engineering method.

2.2 InSiPS Fitness Function

Let *target* denote the target protein sequence and let nt_1, ..., nt_k denote the non-target protein sequences. Consider

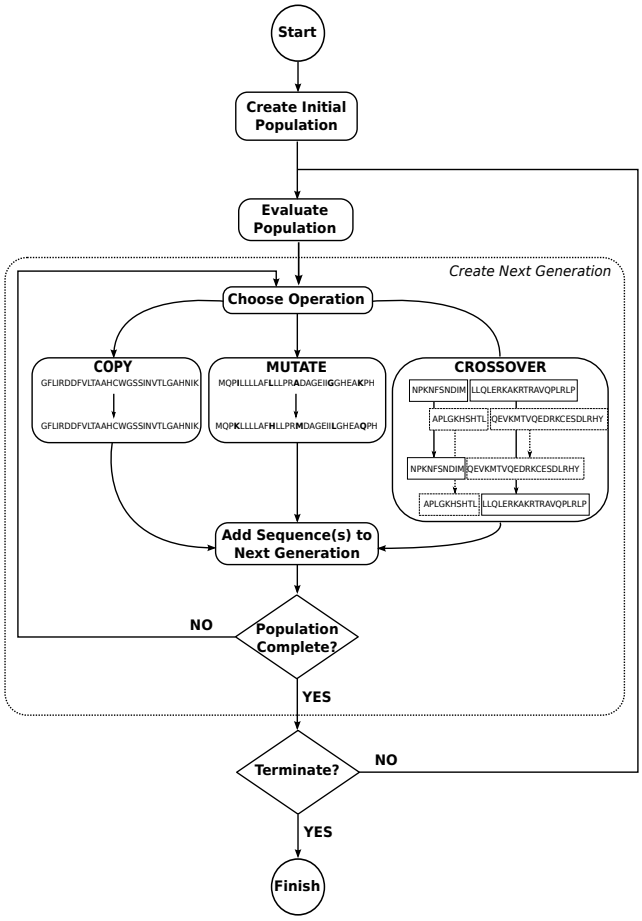

Figure 1: Graphical overview of the core InSiPS genetic algorithm.

a protein sequence *seq* produced by InSiPS. To determine the fitness value $fitness(seq)$ of *seq*, we calculate scores reflecting the likelihood that *seq* interacts with *target* and the likelihoods that *seq* interacts with $nt_1, \ldots, nt_k$. These scores are calculated via our previously published PIPE algorithm [11]. Note that the PIPE scores are not actual probability values but represent relative likelihoods in the sense that if PIPE(A, B) < PIPE(A, B') the predicted probability of protein A to interact with protein B is smaller than the predicted probability of protein A to interact with protein B'. PIPE is uniquely suitable for this task because it is purely sequence-based (ie. it only needs the query protein sequences to predict whether they interact or not) and has an extremely low false positive rate (0.05%).

PIPE relies on a database of known and experimentally verified protein interactions. The database is represented as an interaction graph G where every protein corresponds to a vertex in G and every interaction between two proteins X and Y corresponds to an edge between X and Y in G. The following outlines how, for a given pair (A, B) of query proteins, our PIPE method calculates a score PIPE(A,B) $\in [0, 1]$. In the first step of the PIPE algorithm, protein A is split into overlapping fragments of size w. For each fragment a_i of A, where $0 \leq i \leq |A| - w + 1$, we search for fragments "similar" to a_i in every protein in graph G. A sliding window of size w is used on each protein in G, and each of the resulting protein fragments is compared to a_i. For each protein that contains a fragment similar to a_i, all of that protein's neighbours in G are added to a list R. To determine whether two protein fragments are similar, a score is generated with the use of a PAM120 substitution matrix representing biochemical similarity [6]. (An alternative could have been to use BLOSUM matrices. They are inferred from conserved regions of proteins from distantly related organisms, and the observed substitutions are likely a conservatively biased subset of possible mutations. The PAM120 matrix is more inclusive.) If the similarity score is above a tuneable threshold then these fragments are said to be similar. In the next step of the PIPE algorithm, protein B is split into overlapping fragments b_j of size w $(0 \leq i \leq |B| - w + 1)$ and these fragment are compared to all (size w) fragments of all proteins in the list R produced in the previous step. We then create a result matrix of size $n \times m$, where $n = |A|$ and $m = |B|$, and initialize it to contain zeros. For a given fragment a_i of A, every time a protein fragment b_j of B is similar to a fragment of a protein Y in R, the value at position (i, j) in the result matrix is incremented. The result matrix indicates how many times a pair (a_i, b_j) of fragments co-occurs in protein pairs that are known to interact. Based on the result matrix, the score PIPE(A,B) is calculated. For more details, see [11].

The InSiPS fitness function $fitness(seq)$ combines the values PIPE(seq,$target$), PIPE(seq,nt_1), ..., PIPE(seq,nt_k). Let MAX(PIPE(seq,*non-targets*)) denote the maximum of PIPE(seq,nt_1), ..., PIPE(seq,nt_k), then we define

$$fitness(seq) = (1 - MAX(\text{PIPE}(seq,\textit{non-targets}))) \times \text{PIPE}\ (seq,\ target)$$

A heat map representation of the InSiPS fitness function $fitness(seq)$ is shown in Figure 2. We observe that $fitness$ (seq) increases when PIPE(seq,$target$) increases or MAX (PIPE(seq,*non-targets*)) decreases. The concentric iso-curves (points of same color) in Figure 2 indicate a smooth convergence towards the lower right corner where the fitness peaks at a value of 1 (yellow color).

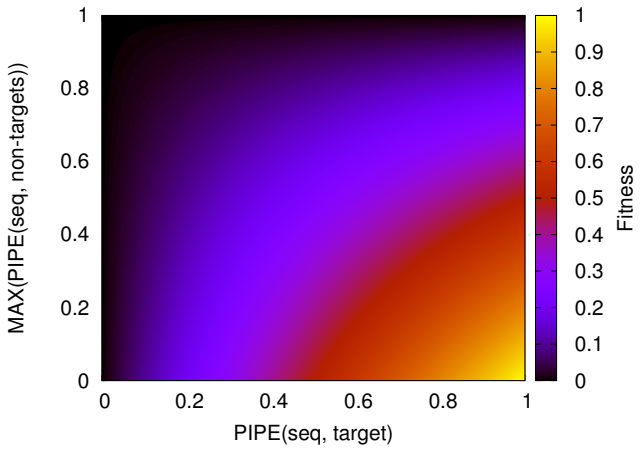

Figure 2: Heat map representation of the InSiPS fitness function.

2.3 InSiPS Implementation on Blue Gene/Q

In this section, we give an overview of the InSiPS implementation on the Blue Gene/Q. It consists of a two-level master-worker/all-workers parallel algorithm. The master/worker portion is implemented using MPI and the all-workers parallelization internal to the MPI processes is implemented in OpenMP. The InSiPS master process is responsible for all of the main genetic algorithm tasks, including the generation of the initial pool of synthetic candidate sequences, the calculation of the individual fitnesses for each synthetic sequence, the application of the GA operations to construct a new generation and the decision to terminate. The worker processes are responsible for providing the PIPE scores for candidate sequences against the target and non-targets. The candidate sequences are issued by the master process in an on-demand fashion, ensuring a balanced load across all of the worker processes. The InSiPS master process is multi-threaded, parallelizing the generation of the initial population, the application of the various GA operations, the calculation of sequence fitness, etc. The InSiPS worker processes implement an all-workers model. The individual PIPE predictions for the provided candidate protein sequence against the target/non-targets is embarrassingly parallel. However, it is crucial for the parallel threads within each worker to share as much data as possible; in particular the database of known interactions used by PIPE. Much care was taken to minimize the memory footprint per thread because otherwise the available memory restricts the number of possible parallel threads per worker.

An outline of the InSiPS master process is given in Algorithm 1. The InSiPS master process loads all relevant data (known protein-protein interaction graph, PIPE similarity database and index, sequences of all known proteins in yeast, and the current target and non-targets) and then broadcasts it to the worker processes. The initial population is created, in parallel, whereby each thread creates random synthetic protein sequences and adds them to the population until the desired number of candidate sequences is reached. Then, the main genetic algorithm loop is entered. To evaluate the current population, the master process waits for work requests from the worker processes. When a requests is received (including the result of previous work assigned to that worker), the master sends a candidate synthetic sequence from the current population to the respective worker. It is the worker process' responsibility to produce PIPE predictions scores for the received synthetic sequence against the target and non-targets. Once all of the synthetic sequences have been processed, their fitness is calculated in parallel. Each computational thread takes a synthetic sequence and, using the PIPE scores generated by the worker processes, it calculates that sequence's fitness as described in Section 2.2. Once all of the synthetic sequences have been assigned a fitness, the next generation can be created. In parallel, each thread randomly decides which genetic algorithm operation it will perform based on the user defined probabilities. After choosing one of these operations (copy, mutate or crossover), the thread then randomly selects one or two proteins (depending on which operation was chosen) from the current generation. The probability that a given synthetic sequence is chosen is based on its fitness relative to the rest of the population. The computational thread then applies the chosen operation to the selected synthetic sequence(s) and adds the new sequence to the next generation. Once the next generation is complete, it is set as the current generation and the main genetic algorithm loop starts again until a termination criterion is met. Once the termination criterion is met, the master process informs the worker processes and the program exits.

Algorithm 1: InSiPS master process.

```
load all required data from disk
broadcast all loaded data to worker processes

current_generation ← ∅
foreach computational thread in parallel do
    while current_population is not full do
        add random protein sequence to
        current_generation

while termination criteria is not met do
    while unanalyzed sequences remain in
    current_generation do
        receive work request from worker x
        receive previous results from worker x
        send sequence from current_generation to
        worker x for analysis

    foreach computational thread in parallel do
        while members of current_generation don't
        have assigned fitness do
            choose a sequence without an assigned fitness
            use results obtained from worker processes to
            compute fitness score

    next_generation ← ∅
    foreach computational thread in parallel do
        while next_generation is not full do
            randomly choose a GA operation
            select sequence(s) from current_generation
            apply GA operation & add new sequence(s)
            to next_generation

    current_generation ← next_generation
foreach worker process do
    receive work request from worker x
    send END signal to process x
```

An outline of the InSiPS worker processes is given in Algorithm 2. The worker processes do not load any data needed to produce the PIPE predictions from disk. Instead, all data is sent over the network from the master process, relieving considerable stress form the shared disks (which are a performance bottleneck in the Blue Gene/Q). Each InSiPS worker process is solely responsible for making PIPE predictions between the generated sequences and the current target and non-targets. To do this, it first sends a work request to the master process (including the result of previous work assigned by the master process). It then receives a candidate synthetic sequence from the master process. To be able to run PIPE on this sequence and the target/non-targets it needs to pre-process the synthetic sequence, building a cached data structure to optimize the performance of similarity searches. The preprocessing is completed offline, beforehand, for the known natural proteins and stored in a database which is among the data loaded and broadcast by

Algorithm 2: InSiPS worker process.

```
receive all necessary data from master process
current_results ← ∅
while true do
    send master process work request
    if message received is END signal then
        break
    send master process current_results
    receive sequence from master process
    sequence_similarity ← ∅
    foreach computational thread in parallel do
        build specified portion of sequence_similarity
    work ← target + non–targets
    current_results ← ∅
    foreach computational thread in parallel do
        while work ≠ ∅ do
            remove a protein from work
            run PIPE algorithm on this protein and
            sequence
            add result to current_results
```

the master process. The remaining preprocessing consists of comparing the given synthetic sequence to all of the known natural proteins to find real proteins which contain similar subsequences to the generated sequence. PIPE makes use of these similarity data to predict interactions. This similarity data is stored in a data structure called *sequence_similarity* and is created, in parallel, with the use of the known natural protein sequences initially sent from the master process and stored in memory for use by all of the computational threads. Once this is completed, while there are still proteins within the target and non-targets that have not been processed, each computational thread takes one of these proteins and runs the PIPE algorithm with this chosen protein and the current synthetic sequence. PIPE uses the newly generated *sequence_similarity* data structure and the standard PIPE database sent by the master. Both of these data structures are read-only and therefore can be accessed by all of the threads simultaneously. Once all of the proteins have been analyzed, the worker process notifies the master process by sending a new work request. This loop continues until the master process notifies the worker processes that there is no more work to do.

3. PERFORMANCE EVALUATION

To benchmark the performance of InSiPS, we investigated how well the code scales as more threads are used within the master and worker processes as well as the impact of using different numbers of worker processes on the overall run time. These benchmarks were broken down into two separate tests and both tests were run on the SciNet Blue-Gene/Q cluster in Toronto, Canada (BGQ). Each node on this cluster consists of 16 1.6 GHz PowerPC based CPUs (capable of supporting 4 computational threads each) and 16GB of RAM. First we will evaluate how the code scales as more threads were used by the worker processes followed by an evaluation of how the number of worker processes used influences the run time.

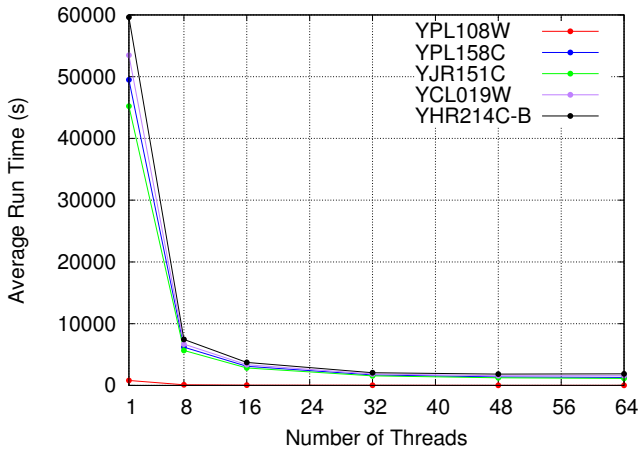

Figure 3: Runtime results for InSiPS threads/worker benchmarking tests on the Scinet BGQ cluster.

3.1 Performance Test 1: Number of compute threads used per worker process

The focus here was on the worker processes since nearly all of the computation time is spent performing protein-protein interaction predictions. The test measures the entire time it takes the worker process to receive the sequence from the master, build the necessary similarity data structure and carry out protein-protein interaction predictions between this sequence and all 6707 yeast proteins on a single BGQ. In this test, five different protein sequences with a range of computational difficulty were evaluated. The computational difficulty of a given sequence depends largely on how many proteins within the PIPE database contain matching subsequences. The more matching proteins a given sequence has, the more known protein-protein interactions PIPE has to investigate. On the other hand, a protein which matches very few proteins in the PIPE database leaves little work for the PIPE algorithm to do. The sequences tested here (listed easiest to hardest) are: YPL108W, YPL158C, YJR151C, YCL019W and YHR214C-B. As can be seen in the run time results (Figure 3) and the speedup results (Figure 4), we see a dramatic decrease in run time when the number of threads on a single BGQ node is increased. We see perfectly linear speedup when using 16 threads and close to linear speedup when using up to 32 threads. After this point we still see an improvement in performance when using up to 64 threads, which happens to be the imposed thread limit on the BGQ cluster nodes.

Overall, we see an extremely solid performance improvement (linear speedup) when using the same number of threads as physical cores available (16). From here we still see an improvement in performance (although not as dramatic) when using up to the number of hardware supported threads. The reason why we hit a reduction in performance improvement when using more threads than the number of threads as physical cores available is due to the fact that InSiPS is memory-IO bound. Since the algorithm does not contain any floating-point arithmetic, the threads spend most of their time doing memory look-ups. When each thread is assigned its own physical compute core with

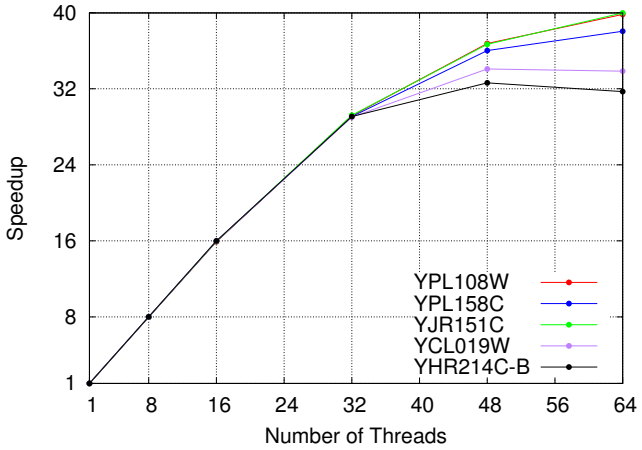

Figure 4: Speedup results for InSiPS threads/worker benchmarking tests on the Scinet BGQ cluster.

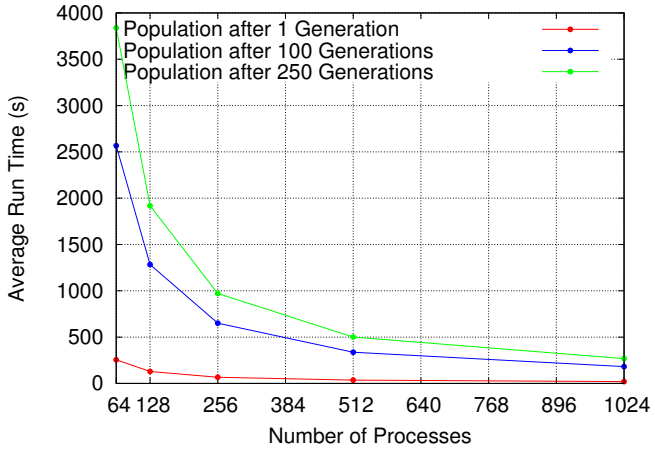

Figure 5: Runtime results for InSiPS worker process benchmarking test on the Scinet BGQ cluster.

its own access to main memory then they will not interfere with each other and we will see good performance. However, when the physical cores are overloaded with computational threads and need to share the communication channels with main memory, we see a reduction in overall speedup.

3.2 Performance Test 2: Number of worker processes used

The second benchmarking test was done to see how well the code scales when more cluster nodes/worker processes are used. These tests measured the entire time it took for a generation to be computed, including the master processes disseminating the sequences to the worker processes, the worker processes internally processing the sequences to prepare the similarity data and then run the PIPE protein-protein interaction prediction method against the targets and non-targets, the worker processes sending the results back to the master process, the master process then calculating the fitness of each sequence and finally the master processes generating the sequences for the next generation. The test problem in question was generated by choosing a total of 250 random target and non-targets and the sequence population consisted of 1500 sequences. It should be noted that the time it takes for a given sequence to be processed is largely based on its complexity (how similar it is to real proteins, how many proteins it is predicted to interact with, etc.). For this reason, this test was performed on three different sequence populations. The first population was a randomly generated set of sequences which represents a typical start point for this method. The second and third populations were taken after running InSiPS after 100 and 250 generations, respectively. The first data set is used to characterize the performance of InSiPS during the first few generations when most synthetic sequences are unsuitable, whereas the latter two data sets characterize the performance of InSiPS later in the process, once the generations of synthetic sequences begin to converge towards useful solutions.

These tests were performed on the BGQ cluster where the minimum number of nodes to use for a job are 64, so this was used as a baseline (1 master process, 63 worker processes). Multiples of 64 nodes were then added to see how well the

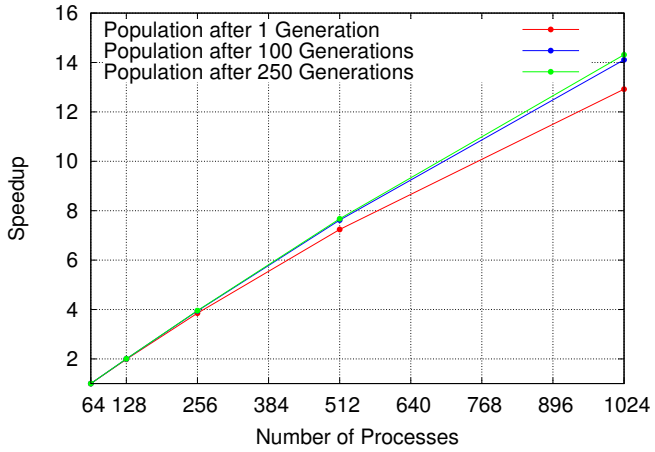

Figure 6: Speedup results for InSiPS worker process benchmarking test on the Scinet BGQ cluster. 64 nodes was the minimum job size and is thus used as a baseline. A speed-up of 16x using 1024 nodes would represent linear speed-up in this scenario.

code scales.

The results for this test on the BGQ cluster are shown in Figures 5 and 6. InSiPS maintains a near-linear speedup when using a moderate number of compute nodes. When InSiPS uses an extreme number of nodes (1024), we see a slight drop-off from linear speedup (12x as opposed to 16x, which would be perfect). This most likely results from two factors. First, it is possible that when there are many worker processes, the master process becomes slightly overwhelmed by work requests, such that some worker processes might have to wait for their requests to be granted. This results in the process idling for a short period of time thus contributing to the lack of perfect scaling. Secondly, a portion of the computation is done by only the master process every generation (the fitness calculation as well as generating the sequences for the next generation). This part of the computation can not benefit from the addition of more cluster nodes (Amdahl's law), although it is computed in parallel

within the master node. Fortunately, this portion only represents a very small percentage of the overall computation and therefore does not have a significant impact. These things considered, InSiPS preforms very well even on an extremely large number of nodes. It is also interesting to note that as the sequence pool becomes more complex (ie. contains protein sequences more likely to be predicted to interact with the target protein and not with the non-targets), InSiPS performs better. This is due to the fact that the individual sequences are becoming more difficult to process giving the worker processes more work to do, leading to a reduction in idle time.

Our performance results are for a one rack BGQ because that was the architecture available for our experiments. To scale to multiple racks, we would set one master process per rack and sync between masters after each round of the genetic algorithm. Since each master's state information is small and the number of racks would also be relatively small (less than 100), the synchronization overhead would be small. This would also allow the initial loading of data to be done in parallel, where the one master per rack would load the needed data upon start-up and broadcast it to all other nodes within the same rack.

4. EXPERIMENTAL VALIDATION

To evaluate the effectiveness of InSiPS on real world, testable scenarios, a set of candidate problems was created. These candidate problems were then experimentally validated in the model organism *S. cerevisiae*. To make these scenarios more amenable to wet-lab experiments using live cells, the set of problems used target proteins with the following properties:

1. Cytoplasmic (proteins only ever shown to localize in the cytoplasm).
2. Relatively small in size (less than 1,500 amino acids in length).
3. Moderate level of abundance (3,000-10,000 transcripts per cell).
4. Absence of the protein resulted in increased sensitivity of the cell to a well-defined external stimuli/stressor.

The task for InSiPS was to design a protein which binds to one of these cytoplasmic proteins of interest (targets) without interacting with with any of the other 1,701 cytoplasmic proteins (non-targets). 18 such target proteins were identified.

4.1 InSiPS Parameter Tuning

As previously mentioned, the probability that a given evolutionary operation is used during the creation of a new sequence by InSiPS's genetic algorithm is predefined by the user. The key input parameters *p_copy*, *p_mutate* and *p_crossover*, shape the way InSiPS builds new sequences in search of a sequence which maximizes the performance criteria for a given problem. The only restriction on these parameters is that they must sum to 1.0, otherwise any combination of positive real number values will work. Given that these are real number parameters, the parameter space is massive in practice. Therefore, the runtime of an exhaustive parameter optimization (even using smart heuristic searches) would be prohibitive. This is compounded by the variability in InSiPS performance for a given set of parameters due to the stochastic nature of the genetic algorithm itself. It is very possible, and indeed likely, for two separate runs of InSiPS on the same problem with the same parameter settings to produce solutions of vastly different quality. For example, one run could benefit from its randomly generated starting pool containing a few very good sequences. This could act as a huge head start over another run, which may take several generations to produce sequences of a similar quality to those that the first run started with. Furthermore, it is very probable that a given set of parameters works better for one problem than it does for another. As previously stated, to do a thorough investigation of these parameters (keeping in mind the effects that the random seed or problem in question can have on InSiPS's performance) is time prohibitive. However, it is important to at least try a number of different settings to see how these three aspects interact with respect to the final sequence quality.

To investigate this, three of the 18 target proteins discussed above were randomly chosen. These targets were YAL054C, YBR274W and YOL054W and the non-targets were the remaining cytoplasmic yeast proteins. To test how using different parameter sets affected the overall performance, five different parameter settings were used. In these settings, *p_copy* was kept consistent at 0.10 (since this operation doesn't add anything new to the next population). Also, in each of these settings, when the mutate operation was chosen, each amino acid in the selected protein sequence would be randomly switched to another amino acid with a probability of 0.05, as this seemed a reasonable mutation rate. The settings of the other parameters were:

- Set 1: $p\_crossover = 0.45$, $p\_mutation = 0.45$
- Set 2: $p\_crossover = 0.30$, $p\_mutation = 0.60$
- Set 3: $p\_crossover = 0.60$, $p\_mutation = 0.30$
- Set 4: $p\_crossover = 0.75$, $p\_mutation = 0.15$
- Set 5: $p\_crossover = 0.15$, $p\_mutation = 0.75$

These parameter settings give a good sampling of different scenarios one might want to use from the balanced approach (parameter set 1) to a set heavily biased in favour of one operation (parameter set 4 or 5). To try to account for the sheer amount of randomness involved in the running of InSiPS, each problem was run with each parameter setting using three different random seeds. When a random number generator is seeded with a given number, it will always produce the same set of random numbers. This way we can assure, for instance, that two different runs of InSiPS have the same initial population. Typically InSiPS is not seeded with a specific seed as different sets of random numbers are generally desirable. That said, we have three problems, each using five different parameter sets, each being run with three different random seeds for a total of 45 runs. The goal here is to see if one parameter setting produces significantly better sequences consistently across different problems and different random seeds. The results of InSiPS runs on targets YAL054C, YBR274W and YOL054W can be seen in Tables 1, 2 and 3, respectively. Here, the fitness of an experiment is the fitness of the highest scoring synthetic sequence observed after 50 generations of the genetic algorithm.

When examining these results we see that the fitness achieved varies similarly between random seeds as it does

Parameters	Seed 1	Seed 2	Seed 3	Avg.
Set 1	0.3564	0.3584	0.3259	**0.3469**
Set 2	0.2852	0.3549	0.2898	0.3100
Set 3	0.3307	0.3100	0.3422	0.3276
Set 4	0.3381	0.3168	0.3301	0.3283
Set 5	0.3293	0.2926	0.3171	0.3130
Avg.	**0.3280**	0.3265	0.3210	

Table 1: InSiPS parameter testing results for target YAL054C. The fitness of each run is presented. Bold indicates the best average fitness across the different parameter settings as well as across the different random seeds used.

Parameters	Seed 1	Seed 2	Seed 3	Avg.
Set 1	0.4438	0.3479	0.3713	0.3877
Set 2	0.2962	0.3266	0.3120	0.3116
Set 3	0.3608	0.3680	0.3732	0.3673
Set 4	0.4220	0.3327	0.4299	**0.3949**
Set 5	0.2874	0.3559	0.3432	0.3289
Avg.	0.3621	0.3462	**0.3659**	

Table 2: Same as Table 1, but using target YBR274W.

Parameters	Seed 1	Seed 2	Seed 3	Avg.
Set 1	0.3613	0.4148	0.4109	0.3956
Set 2	0.3542	0.3531	0.3908	0.3660
Set 3	0.3927	0.4047	0.3429	0.3801
Set 4	0.4078	0.3903	0.4162	**0.4048**
Set 5	0.3586	0.3630	0.3159	0.3458
Avg.	0.3749	**0.3852**	0.3753	

Table 3: Same as Table 1, but using target YOL054W.

between using different parameter settings. For example, for problem YBR274W, parameter set 4 achieves nearly the best fitness from two different random seeds (fitnesses 0.4220 and 0.4299). However, using the third random seed produces one of the lowest fitness values among all combination of parameters and random seeds for that problem (fitness 0.332688). InSiPS seems to perform best with a relatively balanced parameter set. We also note that the overall performance doesn't change drastically between different parameter sets when the different problems and random seeds are considered. The inherent stochastic nature of InSiPS' genetic algorithm seems to provide a level of robustness, allowing any "bad" parameter set to be overcome. Overall, this stability over a variety of input parameters frees the user from having to perform lengthy parameter tuning experiments, due to the fact that InSiPS' performance doesn't vary significantly when these parameters are changed.

4.2 Wet-Lab Experimental Validation

InSiPS was run on all of the 18 experimental candidates. A population size of 1,000 sequences was used with the following paramters: $p\_crossover = 0.5$, $p\_mutation = 0.4$, $p\_copy = 0.1$, $p\_aa\_mutation = 0.05$. InSiPS was run for a minimum of 250 generations. Once this was achieved, it continued running until a new best sequence wasn't found for 50 generations. The three experimental candidates with the fittest generated solution were identified (*S. cerevisiae* target proteins: YDL001W, YAL017W and YBL051C) and were each rerun another 3 times (each run using a different random seed). The best results for these runs were prepared for experimental validation. InSiPS created protein sequences whose predicted interaction score is much higher for the target than even the highest-scoring non-target protein. When instead considering the average non-target predicted interaction score for each generated protein sequence, the separation was even more pronounced. The respective "learning curves" for these InSiPS runs can be seen in Figure 7. This figure shows how the fittest individuals' PIPE predicted interaction score (against the target, highest scoring non-target and average non-target score) changes as the generations progress in a given run of InSiPS.

For each target protein, the coding DNA for the generated anti-target protein designed by InSiPS was commercially synthesized and cloned into an expression vector under the transcriptional control of a promoter. In this way the anti-target proteins could be expressed in yeast cells and their ability to bind to the respective targets and disrupt their function could be evaluated. To validate the efficacy of the generated inhibitory proteins, conditional sensitivity tests were conducted. As previously stated, one criterion for selecting the original experimental candidates was that their deactivation/absence resulted in sensitivity to certain stressors. In this way if the gene which codes for the candidate target protein was knocked-out (deleted), the cell would exhibit increased sensitivity to specific environmental stress (such as the presence of a chemical compound or damaging radiation). If the designed anti-target binds and inhibits the activity of the target protein specifically, it should exhibit a similar conditional sensitivity to the environmental challenge as the gene deletion stain would. This is due to unnatural binding of the target protein to the designed anti-target protein causing alteration in its activity. This is similar to the deactivation of an antigen by antibodies. These

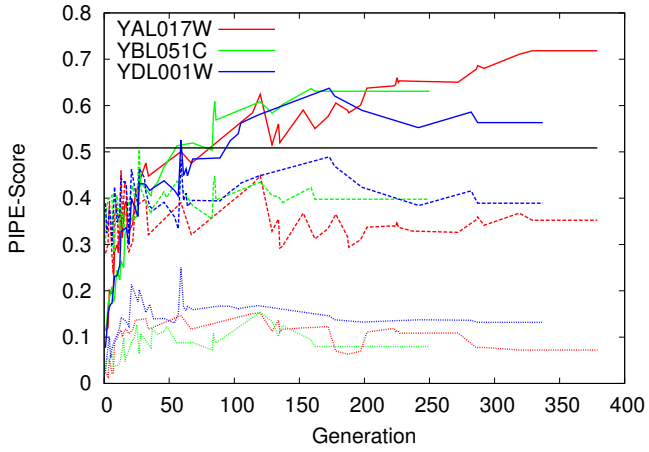

Figure 7: Progression of the best-performing InSiPS runs on the three experimental candidates. In each generation, the three lines of a given colour plot the PIPE predicted interaction score of the fittest sequences against the target (solid line), highest-scoring non-target (dashed line) and the average non-target score (dotted line). The production of the synthetic anti-YAL017W, anti-YBL051C and anti-YDL001W are plotted in red, green and blue, respectively. The black line represents the PIPE acceptance threshold; any protein pair with a score above this threshold is predicted to interact (with a false positive rate <0.5%).

experiments are carried out as follows. First, four different *S. cerevisiae* strains are used. These are the wild-type control strain (WT), a second control strain which contains an empty plasmid (the vector for transporting and housing the gene which codes for the InSiPS protein) ($WT+$), a strain containing a plasmid inducing the production of the generated anti-target protein ($WT + InSiPS$) and a strain in which the gene for the target protein is deleted. In these tests, the first two strains are used as negative controls that reveal the activity of the target protein and the last strain in which the target gene is deleted is used as a positive control to represent the characteristics of the cells with the function of the target protein absent. The strain producing the anti-target protein will resemble the gene deletion strain when challenged by our experimental condition if our protein performs its predicted function. The desired result is to observe a relative decrease in the number of viable cells (increased sensitivity) in our experimental strains compared to the negative control strains when challenged by a specific condition.

These validation experiments were carried out on two of the above three candidates. The wet-lab experiments are rather time consuming and took approximately six months.

4.2.1 Target: YBL051C

Using YBL051C as a target, InSiPS created a synthetic protein sequence with a fitness of 0.379912. It's predicted interaction score with YBL051C was 0.6309 with a maximum off-target score of 0.3978. The average off-target predicted interaction score was 0.0797. This solution gives a very pronounced separation between target and non-target scores, making it a good candidate for validation.

Run	WT	$WT+$	$WT + InSiPS$	$\Delta PIN4$
1	86%	88%	52%	24%
2	87%	96%	53%	31%
3	90%	95%	58%	29%
4	97%	91%	59%	23%
5	91%	87%	55%	28%
Avg.	90%	91%	56%	27%

Table 4: Colony counts for each of the four *S. cerevisiae* strains (WT: wild-type; $WT+$: wild-type cells containing an empty plasmid; $WT + InSiPS$: wild-type strain containing a plasmid that produces anti-YBL051C proteins; $\Delta PIN4$: a knockout strain for YBL051C) after exposure to 65ng/mL of cycloheximide. Colony counts after exposure are normalized to the average colony counts observed under normal conditions.

When the gene which codes for YBL051C (PIN4) is deleted, it is known that *S. cerevisiae* becomes more sensitive to cycloheximide, an inhibitor of protein biosynthesis. Without this protein the cell cannot generate new proteins as easily once challenged with low concentrations of cycloheximide. In this test, all four of the *S. cerevisiae* stains (WT, $WT+$, $WT + InSiPS$, $\Delta PIN4$) were exposed to 65ng/mL cycloheximide and then the number of living cells for each strain were counted on the basis of the colonies that were formed. The summary of the colony counts are given in Table 4. The averages seen in these five experiments are also shown in Figure 8. As can be seen in these figures, the two control strains (WT and $WT+$) show similar sensitivity to cycloheximide. On the other hand, we see a much larger reduction in the number of cells for both the $WT + InSiPS$ and the positive control $\Delta PIN4$ strains. The reduction in viable colony counts for the $WT + InSiPS$ strain, although not as dramatic as in the positive control strain $\Delta PIN4$, suggests that the InSiPS designed anti-YBL051C protein is successfully inhibiting YBL051C, increasing the sensitivity of the cells to cycloheximide.

4.2.2 Target: YAL017W

Using YAL017W as a target, InSiPS created a protein sequence with a fitness of 0.4652. Its predicted interaction score with YAL017W was 0.7183 with a maximum off-target score of 0.3524. The average off-target predicted interaction score was 0.0721. As in the previous example, we see a significant separation between target and non-target scores, where we would expect very few if any non-targets to interact with the designed protein.

When the gene which codes for YAL017W (PSK1) is deleted, it is known that *S. cerevisiae* becomes more sensitive to ultraviolet light. Without this protein the cell's ability to repair the DNA damage caused by exposure to UV light is significantly diminished. In this test, all four of the *S. cerevisiae* stains (WT, $WT+$, $WT + InSiPS$, $\Delta PSK1$) were exposed to UV light for 30 seconds. The living cells were then allowed to grow to form colonies. As above, colony

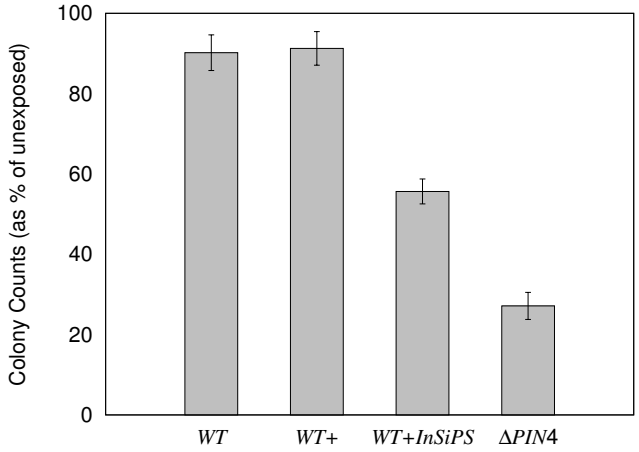

Figure 8: Average colony counts for each of the four *S. cerevisiae* strains (WT: wild-type; $WT+$: wild-type cells containing an empty plasmid; $WT + InSiPS$: wild-type strain containing a plasmid that produces anti-YBL051C proteins; $\Delta PIN4$: a knockout strain for YBL051C) after exposure to 65 ng/mL of cyclohexamide. Average colony counts after exposure are normalized to the average colony counts observed under normal conditions. Error bars represent standard deviation.

counts were used to measure cell survival and hence cell sensitivity. The summary of the living colony counts are given in Table 5. The averages seen in these five experiments are also shown in Figure 9.

Spot test analysis was also carried out for these experiments and the results are illustrated in Figure 10. As can be seen in this table and figure, the two control strains (WT and $WT+$) show similar sensitivity to UV exposure. On the other hand, we see a large reduction in the number of viable colonies for both the $WT + InSiPS$ and $\Delta PSK1$ strains (which was expected in the positive control strain $\Delta PSK1$). The increased UV sensitivity in the $WT + InSiPS$ strain suggests that the InSiPS designed anti-YAL017W protein is successfully inhibiting YAL017W activity, lowering its resistance to UV light.

5. CONCLUSION

In this paper we have presented InSiPS: The In-Silico Protein Synthesizer, a new massively parallel algorithm for designing novel inhibitory proteins. Unlike other approaches which rely on 3D protein structure data, InSiPS is purely sequence-based, allowing it to be applied to a much wider range of problems. Given only the sequences of a target and set of non-target proteins and a list of known protein-protein interactions, InSiPS can produce a novel protein which is predicted to interact specifically with the target while not interfering with the non-targets. InSiPS was benchmarked on the IBM Blue Gene/Q cluster and was shown to efficiently use all computational threads on each compute node while also scaling near-linearly up to 1024 nodes. Experiments revealed that InSiPS' performance is relatively stable across a large set of parameters, allowing users to forgo lengthy parameter tuning exercises before applying the algo-

Run	WT	$WT+$	$WT + InSiPS$	$\Delta PSK1$
1	54%	51%	20%	13%
2	56%	55%	5%	4%
3	53%	56%	8%	5%
4	55%	55%	16%	11%
5	57%	56%	17%	14%
Avg.	55%	54%	14%	10%

Table 5: Cell colony counts for each of the four *S. cerevisiae* strains (WT: wild-type; $WT+$: wild-type cells containing an empty plasmid; $WT + InSiPS$: wild-type strain containing a plasmid that produces anti-YAL017W proteins; $\Delta PSK1$: a knockout strain for YAL017W) after exposure to 30 seconds of ultraviolet light. Colony counts after exposure are normalized to the average colony counts observed under normal conditions.

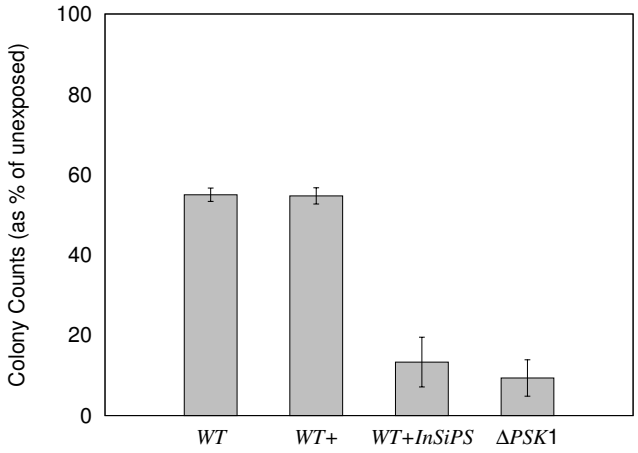

Figure 9: Average colony counts for each of the four *S. cerevisiae* strains (WT: wild-type; $WT+$: wild-type cells containing an empty plasmid; $WT + InSiPS$: wild-type strain containing a plasmid that produces anti-YAL017W proteins; $\Delta PSK1$: a knockout strain for YAL017W) after exposure to 30 seconds of ultraviolet light. Average colony counts after exposure are normalized to the average colony counts observed under normal conditions. Error bars represent standard deviation.

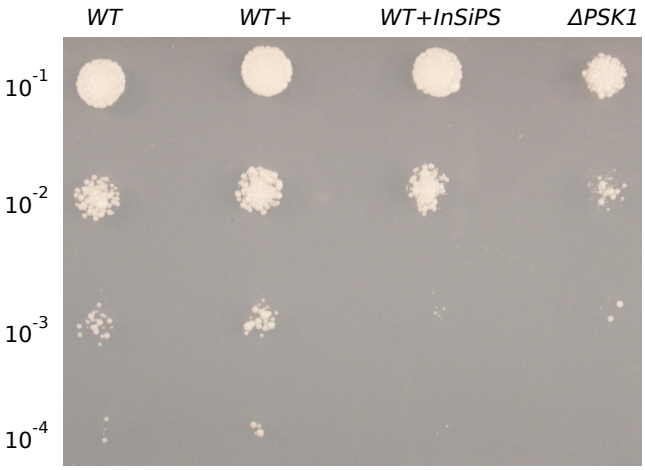

Figure 10: Spot test for the four *S. cerevisiae* strains (WT: wild-type; $WT+$: wild-type cells containing an empty plasmid; $WT + InSiPS$: wild-type strain containing a plasmid that produces anti-YAL017W proteins; $\Delta PSK1$: a knockout strain for YAL017W) grown for 48 hours after 30 seconds of exposure to ultraviolet light. Each column contains an equal number of cells diluted 10X down each row. Decreased growth in columns 3 and 4 indicates that the expression of anti-YAL017W sensitizes cells to UV in a similar manner as the absence of YAL017W.

rithm to real problems. Finally, InSiPS was used to design inhibitors for proteins with known function in *S. cerevisiae*. These designed proteins were shown through experimental validation to successfully inhibit their intended targets, leaving the cell in a state similar to one where the target protein is not present at all.

Future research directions for InSiPS include designing inhibitory proteins to be used as molecular markers for pathogenic strains of *E. coli* and designing inhibitory proteins to obstruct the spread of certain viruses (e.g. HIV).

6. ADDITIONAL AUTHORS

James R. Green
Department of Systems and Computer Engineering,
Carleton University
jrgreen@sce.carleton.ca

7. REFERENCES

[1] B. Alberts, A. Johnson, J. Lewis, M. Raff, K. Roberts, and P. Walter. Protein Function. In *Molecular Biology of the Cell*. Garland Science, New York, NY, 4 edition, 2002.

[2] P. Benjamin Stranges and B. Kuhlman. A comparison of successful and failed protein interface designs highlights the challenges of designing buried hydrogen bonds. *Protein Science*, 22:74–82, 2013.

[3] A. Ceol, A. Chatr-aryamontri, E. Santonico, R. Sacco, L. Castagnoli, and G. Cesareni. DOMINO: a database of domain-peptide interactions. *Nucleic acids research*, 35(Database issue):D557–60, Jan. 2007.

[4] R. A. Chica. Protein engineering in the 21st century. *Protein Science*, 24(4):431–433, 2015.

[5] G. M. Cooper. Regulation of Protein Function. In *The Cell: A Molecular Approach*. Sinauer Associates, Sunderland, MA, 2 edition, 2000.

[6] M. Dayhoff, R. Schwartz, and B. Orcutt. A model of evolutionary change in proteins. In *Atlas of protein sequence and structure*, pages 345–352. 1978.

[7] S.-g. Kang and J. G. Saven. Computational protein design: structure, function and combinatorial diversity. *Current opinion in chemical biology*, 11(3):329–34, June 2007.

[8] S. M. Lewis and B. a. Kuhlman. Anchored design of protein-protein interfaces. *PLoS ONE*, 6(6), 2011.

[9] R. J. Pantazes, M. J. Grisewood, and C. D. Maranas. Recent advances in computational protein design. *Current opinion in structural biology*, 21(4):467–72, Aug. 2011.

[10] J. G. Saven. Computational protein design: Advances in the design and redesign of biomolecular nanostructures. *Current opinion in colloid & interface science*, 15(1-2):13–17, Apr. 2010.

[11] A. Schoenrock, F. Dehne, J. R. Green, A. Golshani, and S. Pitre. MP-PIPE: a massively parallel protein-protein interaction prediction engine. In *Proceedings of the International Conference on Supercomputing*, pages 327–337, 2011.

[12] C. Stark, B.-J. Breitkreutz, A. Chatr-Aryamontri, L. Boucher, R. Oughtred, M. S. Livstone, J. Nixon, K. Van Auken, X. Wang, X. Shi, T. Reguly, J. M. Rust, A. Winter, K. Dolinski, and M. Tyers. The BioGRID Interaction Database: 2011 update. *Nucleic acids research*, 39(Database issue):D698–704, Jan. 2011.

[13] The Uniprot Consortium. Activities at the Universal Protein Resource (UniProt). *Nucleic acids research*, 42(Database issue):D191–8, Jan. 2014.

[14] C. Turano, E. Gaucci, C. Grillo, and S. Chichiarelli. ERp57/GRP58: A protein with multiple functions, 2011.

[15] B. O. Villoutreix, M. a. Kuenemann, J. L. Poyet, H. Bruzzoni-Giovanelli, C. Labbé, D. Lagorce, O. Sperandio, and M. a. Miteva. Drug-like protein-protein interaction modulators: Challenges and opportunities for drug discovery and chemical biology, 2014.

[16] F. Yu, V. M. Cangelosi, M. L. Zastrow, M. Tegoni, J. S. Plegaria, A. G. Tebo, C. S. Mocny, L. Ruckthong, H. Qayyum, and V. L. Pecoraro. Protein design: toward functional metalloenzymes. *Chemical reviews*, 114(7):3495–578, Apr. 2014.

[17] A. Zanghellini. de novo computational enzyme design. *Current opinion in biotechnology*, 29:132–8, Oct. 2014.