Optimization and Optimal Control of Agent-Based Models

Matthew Oremland

Thesis submitted to the faculty of the Virginia Polytechnic Institute and State University in partial fulfillment of the requirements for the degree of

Master of Science
In
Mathematics

Reinhard C. Laubenbacher, Chair
Nicholas A. Loehr
Lizette Zietsman

May 3, 2011
Blacksburg, Virginia

Keywords: optimal control, agent-based model, individual-based model, heuristic algorithm, polynomial dynamical system, bioinformatics, systems biology, discrete model

Optimization and Optimal Control of Agent-Based Models

Matthew Oremland

Abstract

Agent-based models are computer models made up of agents that can exist in a finite number of states. The state of the system at any given time is determined by rules governing agents' interaction. The rules may be deterministic or stochastic. Optimization is the process of finding a solution that optimizes some value that is determined by simulating the model. Optimal control of an agent-based model is the process of determining a sequence of control inputs to the model that steer the system to a desired state in the most efficient way. In large and complex models, the number of possible control inputs is too large to be enumerated by computers; hence methods must be developed for use with these models in order to find solutions without searching the entire solution space. Heuristic algorithms have been applied to such models with some success. Such algorithms are discussed; case studies of examples from biology are presented. The lack of a standard format for agent-based models is a major issue facing the study of agent-based models; presentation as polynomial dynamical systems is presented as a viable option. Algorithms are adapted and presented for use in this framework.

Acknowledgements

I would first like to acknowledge my advisor on this project, Dr. Reinhard Laubenbacher. He helped me immeasurably during frequent meetings with suggestions on literature, ideas for exploration, and encouragement. His professional experience and expertise was brought to bear on the entire process and the project could not have been completed otherwise. I would also like to thank Dr. Nicholas Loehr and Dr. Lizette Zietsman for their continued support and feedback. They both helped point me in the right direction concerning my research and in my view of mathematics in general. I am grateful. I thank my colleagues and classmates as well; many fine details have been hashed out, written down, erased and re-written in the course of my studies. Franziska spent much time and patience explaining her work and much of the background material for this document. Andy, David and Kelli have all been there when the functions and variables were too baffling to confront alone. Finally, I would like to thank my family for their continued support and dedication; they have been nothing but helpful and encouraging. My wife Fangfang remains an excellent colleague, companion, and source of inspiration.

Table of Contents

1. Introduction

1.1 Overview of Modeling

Applied mathematics is the art of using mathematical ideas to answer questions. The questions may be motivated by real-world situations, or they may be formulated as intellectual curiosities that have no immediate apparent application. The mathematical methods of answering these questions vary greatly, as it is up to the mathematician to develop a method that is capable of solving a particular problem; therefore the methods involved are as various as the questions themselves. Nevertheless, it is possible to develop a general framework of methods that are particularly suitable to answering certain types of questions, and mathematical modeling is one such method.

A mathematical model is a way of encoding relationships or interactions into a formalized system that can then be studied and analyzed using mathematics. A model can take many different forms: it may be a system of ordinary differential equations, but may just as likely be a dynamical system. Mathematical models allow scientists to examine a system and its behavior, thereby enabling analysis of the system. For example, a model may be used in order to make predictions about the future behavior of a system, or it may be used to explicitly solve a complicated problem. By formalizing the interactions of a system in the language of mathematics, a set of tools becomes available that would not be available otherwise.

1.2 Scientific use of models

Models are used in many areas of scientific research. In engineering, they are used in operations research in order to solve problems including complicated job scheduling, weapons defense systems, and job assignments. Computer engineering uses models to determine the most efficient way of soldering circuits onto a processor. In geographical information systems, models are used to make predictions about the weather, the ocean, and other natural phenomena. Models are used extensively in the analysis of geospatial networks such as satellite systems and cellular

phone systems. Even the social sciences use models as ways of understanding how communities and other social groups interact with one another.

Biology, in particular, is a field in which models have been used extensively. Biologists use models to study interactions on a molecular and cellular level. A model can be used to study the immune system, for example, and its response to administration of a certain drug; another model may follow the spread and transmission of a disease. Much modeling has been done in the field of epidemiology, in which the interactions among a population are examined to study how epidemics begin, how they spread, and how they can be controlled or contained.

1.3 Continuous and Discrete Models

Mathematically speaking, models can be put into two basic categories: continuous and discrete. In a continuous model, all possible values within a certain range are considered as possibilities; in other words, there is no 'space' between one possible value and the next. In a discrete model, only certain values are permitted. For example, in a time-continuous model, the state of the system can be determined at any time value (i.e., any real number for which the system is defined). In a discrete-time model, on the other hand, the state of the system can only be determined at fixed time steps; i.e., every hour, minute, or second (or any other unit of time measurement). Continuous models are often described by differential equations, as they are typically continuous functions that can take on any value within a certain range. A discrete model might be given as a system of equations whose new values are updated based on previous values; they may have entries in a finite field, for example, rather than real numbers. Of course, these are just two examples; in practice, models take many different forms.

A model may be continuous in some respects but not in others: a model of disease transmission within a population may treat time as a continuous variable but population as discrete. Occasionally, certain properties of the models might suggest that it is more reasonable to view time as a discrete variable: in the disease case, for example, it is likely not necessary to model time continuously, because symptoms manifest and diseases spread at a relatively slow rate. By examining the population on a daily basis we will most likely capture the same dynamics of the system as if we were to examine the state of the population at every second, or at every instant. In other cases, as in those of interactions at the cellular level (or other situations

in which the population is very large), it might make sense to view the populations as continuous (while perhaps still treating time as discrete). This is especially true if we are not interested in the fate of individual cells but rather the overall properties of the cell population.

The type of model we should choose depends on various factors: what the model will be used for, the variables involved, and of course just plain personal preference, to name a few. There are analytical tools for both types of models – a question that is very difficult to answer about a discrete model may be answered simply in a continuous model, and vice versa. In this paper we will focus on discrete models and in particular a certain type of discrete model, which will be defined and described at the end of this chapter.

1.4 Optimization and optimal control

1.4.1 Overview

Optimization is the process of finding the best solution with respect to a particular goal. For example, suppose we have a model of the immune system which is infected with bacteria, and we wish to study the effect of a certain type of drug. We may wish to find out which drug does the least amount of tissue damage while curing the infection; this is an optimization problem because we are searching for the best drug (or drugs) with respect to the stated goal. On the other hand, perhaps we wish to cure the infection in the shortest time possible, regardless of the tissue damage caused; this would be another optimization problem. It is likely, then, that the solution to the optimization problem depends entirely on the optimization goal. In this scenario, the drug which cures the infection the quickest may consequently do more tissue damage than other drugs (though not necessarily); on the other hand, the drug that causes the least tissue damage might not cure the infection in the shortest possible time. In this example, optimization is a process of minimization: in one case we wish to minimize tissue damage and in the other we wish to minimize the healing time. However, it is also possible that the solution of an optimization problem is a process of maximization: for example, given a number of machines in a factory and the fact that these machines break down and must be repaired from time to time, what is the repair schedule that produces the maximum profit? In the biological setting we might

ask the following: given a model of an immune system infected with a fatal illness, what treatment will enable the patient to survive the longest?

Thus, optimization is the process of finding the best solution, depending on the objective. In fact, it may be that the goal of the optimization problem is to minimize one value while maximizing another. For example, our optimization goal may be to minimize tissue damage and maximize the expected lifespan of the patient, given that we can only administer a particular drug a fixed number of times.

Typically, optimization is a complicated process, and becomes even more so when realistic constraints are put in place. Through the use of mathematical models it may be possible to obtain a solution to an optimization problem that is not feasible in actuality: the solution may exceed certain monetary limitations, for example, or require interaction with a percentage of the population that is too high to be realistic. Nevertheless, optimization remains a natural means of applying mathematics to solve real-world problems. Models that are created to simulate real-world processes have little use if they are not examined in order to answer some type of question. Such questions are typically optimization questions: what is the best outcome that can be produced based on the properties of the model?

Optimal control is a slightly different notion; in an optimal control problem, we ask a different question: given a particular state that we hope to reach, what is the best (i.e., most efficient) means of reaching that state? In other words, we know the state of the system that we hope to reach, and the control problem is to find the solution that steers the system to that state in the best manner. Using the model of an infected immune system as an example, we may formulate an optimal control problem as follows: given that we wish to eliminate the number of infected cells in one month, what is the best drug treatment schedule we can devise? To summarize: in an optimal control problem we know the state we hope to reach and we search for the best way to reach it, whereas in an optimization problem, a goal is stated and we compare solutions to see which solution maximizes or minimizes the stated goal of the problem.

1.4.2 Importance of the validity of the model

Models are implemented as a means of studying the dynamics of some process or interaction. One benefit to using models is that it precludes conducting an actual experiment, which can be costly and time-consuming. For example, if we wish to study the effects of a

certain treatment on its ability to prevent tumor growth, a computer simulation of a model can run the experiment many hundreds of times in one day, whereas reproducing the experiment in a laboratory might require meticulous observations of many subjects over the course of an entire year. The cost of simulating the model on the computer is more or less limited to computer time, whereas a lab experiment would require resources whose budget would almost certainly exceed that of computer run-time. Of course, the results obtained from the use of mathematical models ought to ultimately be tested *in vivo* (that is, in an actual laboratory) in order to verify that the results correspond to reality, but in general we can run many computer simulations in order to decide how to test the results *in vivo*.

Thus, models offer a substantial benefit to the user: they can reproduce results that would take much longer in real life, they cause no actual risk or harm since the subjects are all virtual, and they allow for in-depth analysis that may not be possible otherwise. The use of a mathematical model and its ability to predict results of real-life situations, then, depends largely on the validity of the model. If a model is not calibrated to empirical data or has not been otherwise validated, it is impossible to externalize the results and make predictions. Therefore it is necessary that the model reproduces the system it is purported to be modeling as faithfully as possible – this may be another factor when making the decision of which type of model to use.

1.4.3 Definition of Terms

We end this chapter with a definition of the terms that will be used in the descriptions of the algorithms and throughout the remainder of this paper. We do this in order to standardize the terminology within this document and to provide a consistent framework. An example is given, explaining how each term is meant to be used in the framework of that example. We then define the terms by abstracting the meaning of each in light of the example. In the next chapter we proceed with an overview of heuristic algorithms, which have been used to solve optimization and optimal control problems.

Example. Suppose that we have a model of the immune system, and we wish to study the effect of a certain drug when the immune system has been infected with lung cancer. We formulate our optimal control problem as follows: which treatment schedule should we choose in order to reduce the number of cancer cells to be below some fixed threshold over the course of

one year, given that we wish to minimize the number of times the drug is administered and maximize the number of healthy cells?

In this case, there will be many variables: the number of healthy cells, the number of cancer cells, the rate at which cancer cells grow, the rate at which healthy cells regenerate, the expected lifespan of the patient, the frequency with which the drug is administered, the type of drug, and so on. Some of these variables will have fixed values; for example, the rate at which healthy cells regenerate (when no treatment is administered) is fixed, and in fact this value helps to define the model itself. Such variables are referred to as **model parameters**; they are a part of the specification of the model. The interaction of the cells in the immune system before we introduce the infection and the treatment is referred to as the **model dynamics**.

Note that we will only have control over some of the variables, and others we will simply measure by observation. For example, during each day of the simulated treatment we can decide whether or not to administer the drug; thus we have direct control over the value of such variables. We refer to these as **control variables**, because we have direct control over their values. Other variables, such as the number of healthy cells that are present at a given time, we cannot control directly. We choose the treatment schedule and then observe this level; variables of this type are referred to as **state variables** because they help to describe the state of the system at any given time.

In this example, our only control variable is whether or not to administer the drug on a given day; thus there are only two possible values for this control variable at each time step. For any given day, we may represent that the drug was administered that day by giving that variable the value 1, and represent that the drug was not administered by assigning that variable the value 0. In this situation, a treatment schedule is a vector of length 365, of which each entry is either a 0 or a 1. Thus there are a total of 2^{365} possible treatment schedules. However, it is likely that not all possible treatments will reduce the number of infected cells below the fixed level; the treatment schedules that *do* achieve this are **solutions** to the optimal control problem. The **solution space** consists of all such solutions.

In an optimal control problem there is an associated **cost function** that determines the relative value of each solution. We are attempting to steer the system to a particular state, and each solution achieves that goal at a certain cost. We wish to determine the best solution (i.e., the **optimal control**); thus we wish to find the solution that minimizes the cost function. Earlier we

noted that the best solution was the solution that involved the least number of treatments; thus in this case a reasonable cost function might be the sum of the entries in the solution (i.e., the number of days on which the drug was administered). For the purposes of this paper, the goal of the optimization and the optimal control problems will always be to minimize the associated cost function.

In this scenario, we wish to minimize *T*, the total number of days the drug is taken, and maximize *I*, the number of healthy cells. Then a reasonable **cost function** might be

$$c(T, I) = T - I.$$

Here we want to maximize *I*; this is the same as minimizing *–I* (in this way we can always formulate our problem so that the goal is to minimize the cost function). However, it is perhaps more realistic that one of our optimization goals has a higher priority than the other; in this case, for example, we may suppose that maximizing the number of healthy cells is more important than minimizing the number of treatment days. So we can adjust the **cost function** accordingly:

$$c(T, I) = T - 5I\ .$$

This has the effect of treating the healthy cell count as being five times more important than the number of treatments.

In order to use our model to obtain results, we implement some solutions, **simulate** the immune system response, and make observations on the results. We might, for example, administer treatment every day. This particular solution is represented as a string of 365 consecutive 1's; we implement this in the computer program and see how many healthy cells there are after one year of simulated time. We then evaluate the cost function based on this information. Note that we cannot hope to run all of the solutions (recall that there are 2^{365} different treatment schedules) because of the monumental amount of computing time involved; those that we do implement comprise one particular **population** of solutions. However, we can implement several other solutions and compare the results in order to decide which solution yielded the minimal value of the cost function; we consider this to be the **optimal solution**.

We now define the following terms, each of which have been illustrated in the previous example.

Terms

Model: the representation of a system or a process using a set of equations, inequalities, probability distributions, or other rules to describe how a combination of state variables, control variables, and model parameters interact.

Model parameters: quantities that are part of the model specification. They have fixed values.

Model dynamics: the interaction between the state variables in a model. In general, the model dynamics will be affected by the model parameters and the rules that govern the interaction of the variables.

State variable: variables whose values cannot be specified. State variable values affect the model dynamics; they are affected by the value of other state variables, model parameters, and control variable values.

Control variable: variables whose value can be specified by the user. Altering the value of a control variable will (in general) have some effect on the resulting model dynamics.

Solution: a sequence of inputs to the control variables. A **full** solution assigns a value to each control variable at each time step; a **partial** solution is a sequence wherein values are either only assigned to some of the control variables, or are assigned to the control variables at only certain time steps.

Solution space: the set of all possible solutions. If $p_1, p_2 \ldots, p_n$ are the control variables, and each parameter p_i (for $1 \le i \le n$) can take on a_i possible values, and there are a total of *t* time steps, then the solution space will consist of

$$\prod_{i=1}^{n} a_i^{\,t} = a_1^{\,t} \cdot a_2^{\,t} \cdot \ldots \cdot a_n^{\,t} = (a_1 a_2 \cdots a_n)^t$$

solutions. Thus each solution is a vector of length *t*, and the solution space is the set of all admissible vectors.

Population: a subset of the solution space. The population may be the entire solution space or a proper subset.

Cost function: a function that assigns a value to each possible solution. In general, the cost function will depend upon a combination of state variables and other quantifiable aspects of the model dynamics.

Simulation: the implementation of a solution via a computer program. Simulations can be set to run for a predetermined number of time steps, or until some stopping condition is met.

Optimal solution: In a general optimization problem the optimal solution is a solution that achieves the stated optimization goal in the best way. In an optimal control problem, it is the solution that is associated with the minimum value of the cost function.

---

2. Algorithms

2.1 Overview of Optimization Algorithms

The ability to use a model to find an optimal solution to a problem depends on several factors. The type of model, the complexity of the model, and the method of optimization all affect the results that analysis will produce. In a continuous model, it may be possible to solve a system outright and thus an optimal solution can be explicitly found. For sufficiently small discrete models (or models whose dynamics are easily modeled), exhaustive searches can be done which reveal the best choice of parameters.

In fact, algorithms for producing optimal parameters have been an issue of mathematical research since at least the 1950s. While there are methods for finding explicit optimal solutions in some cases (see [**1**]), many modern models are too complex for these methods to work. A discrete model of the immune system, for example, might involve millions of cells – each will have its own rules describing its interaction with other cells, and may also have many variables attributed to it. Analytic methods fail in these cases, and even exhaustive enumeration by computer is unfeasible due to the limitations of computer processing speeds. In fact, as the processing power of computers grows, so too does the possibility for creating models that increase in complexity, so that it is impossible to say that the computers will ever be able to 'catch up' to the complexity of the models.

For this reason, most of the optimization and optimal control methods currently used in discrete modeling are in the form of heuristic algorithms. An algorithm is a formal set of rules used in order to obtain a solution to a problem. A heuristic algorithm is a specific type of algorithm that conducts a 'smart search' of the solution space. It may make use of certain aspects of the problem to avoid having to search through every possible solution, or it may update its search method based on the input it receives from previously found solutions. These algorithms begin with a choice of control input values (i.e., full or partial solutions) and use various methods to refine the values so as to decrease the value of the associated cost function until no better solution can be found. While these methods do not guarantee an optimal solution, they do provide the opportunity for analysis. In many cases, this is the best method for model parameter optimization that is currently available.

Of course, every algorithm has associated risks and advantages, and the particular algorithm that is best suited for a given model or problem will depend on these risks and advantages. As such, we will follow the description of each algorithm with a brief discussion on this topic. Finally, each description is concluded with a block of pseudo-code that uses the given terminology to summarize the algorithm.

2.2 Genetic Algorithms

Genetic algorithms (sometimes referred to as evolutionary algorithms) were originally developed as a means of studying the natural evolution process, and have been adapted for use in optimization since the 1960s (see [**2, 3, 4**]). A solution (i.e., a sequence of control variable values) is viewed as a chromosome. The chromosome is a string of values, or genes, each of which represents the value of one of the parameters. Each chromosome, then, represents an individual solution that can be implemented and whose associated cost can then be evaluated (or more typically, whose cost can be measured as an average of the costs obtained from repeated simulations, since there will be variations in the simulation due to the stochastic nature of the problems).

A typical genetic algorithm functions in the following way: several chromosomes are selected to form a population. The cost of each is then evaluated. The chromosomes are then ordered, beginning with the chromosome that produces the minimum cost. Half of the chromosomes are then selected to carry on to the next generation; that is, they will be kept on the list to be evaluated later. Then the process of *breeding* comes in to play. Two random chromosomes from those that have been carried over are selected to serve as parents. A child chromosome (i.e., a new solution) is then bred from these parents by some method of *crossover* (*uniform crossover* is a typical method in which there is an equally likely chance that the child will take each particular gene from either parent). The next step is *mutation*: each gene has the possibility of being mutated at random, changing from its current value to any admissible value of that gene. The child chromosome produced from such breeding is added to the next generation of solutions. This process is repeated until the remaining half of the new population has been re-populated with new child chromosomes. The entire process is then repeated: the chromosomes are evaluated and the best are set aside, new chromosomes are bred from them, and so on.

The algorithm continues for a pre-determined number of steps, or until a certain condition is met. For example, we may choose to run the algorithm for 100 generations. Another method is to repeat the process until no better solution has been found for, say, ten consecutive generations. When the algorithm terminates, we choose the best current solution as our candidate for an optimal solution (note that there is no guarantee that the best solution found by the algorithm is actually the optimal solution; given that we are never sure, we use *optimal solution* to mean the best solution produced by the algorithm).

As with other heuristic algorithms, there are many variations and the one presented here is provided as a standard procedure. We may modify the algorithm, for example, so that the initial chromosomes are selected in a certain way: perhaps we wish to choose them at random, or perhaps we wish to choose chromosomes that are very different from one another (i.e., solutions that come from different regions of the solution space). We may modify the crossover process so that one parent is favored over another, or we may forego the mutation step altogether. The likelihood of mutation is another area where user input is important: a high level of mutation will result in more variation of child chromosomes, and thus will not incorporate the relative fitness of the parent chromosomes as much. On the other hand, if the mutation step is not included then we run a greater risk of our solution converging to some solution that is only locally minimum; that is, it may be better than all of the solutions that are similar to it, but not necessarily better than solutions that are radically different (we think of these solutions as being farther away in the solution space). The advantage of using a genetic algorithm is that there is inherently some level of stochasticity; that is, there is always the possibility of mutating to a better solution (as long as the mutation step is included). Another advantage is that the methods are relatively well-established and there is a wealth of information in the literature.

GENETIC ALGORITHM – PSEUDO-CODE

0. Generate a set of n initial solutions as P_1, the initial population (n even).
1. Define **Sort**( ) function, which takes a set of solutions as input, evaluates them in the cost function, and then re-indexes the set so that the first entry corresponds to the lowest cost, and so on.

2. $i = 1$.
3. While(*stopping condition not reached*) do

    Sort(P_i);

    $P_t = \{\}$;

    for j from 1 to $\frac{n}{2}$ do

        $P_t = P_t \cup P_i[j]$;

    end

    $P_{i+1} = P_t$;

    for k from 1 to $\frac{n}{2}$ do

        P_r = **ChooseTenRandom**(P_t);

        Sort(P_r);

        $P_f = P_r[1]$;

        $P_m = P_r[2]$;

        P_c = **Breed**(P_f, P_m);

        Mutate(P_c);

        $P_{i+1} = P_{i+1} \cup P_c$;

    end

    $i = i + 1$;

    end
4. **Sort**(P_i).
5. **Return**($P_i[1]$).

2.3 Simulated Annealing

Annealing is a chemical process in which some material (for instance, a metal) is heated to a high temperature and then cooled very slowly. The cooler the material gets, the slower the cooling process; thus the closer the temperature gets to absolute zero, the more time the material spends at any given temperature. This process can be used on metals to make them more pliable

and easier to cut; chemists also use annealing to study the states of various materials at low temperatures. As the temperature cools, the energy of the system is lowered. One of the reasons that materials are cooled at a slower rate as the temperature decreases is so that the energy can reach equilibrium at each lowered temperature. This allows for stability: for example, crystals that form as a material is cooled show increased stability in their forms if they are allowed to reach equilibrium as the temperature is slowly lowered. As the temperature approaches absolute zero, the energy level lowers as well, approaching some absolute minimum level.

Simulated annealing is a computational process that was inspired by this chemical process, and has been in wide use since it was first introduced in 1983 [**5**]. In order to apply the algorithm, we must define what we mean by temperature and energy, since these are the two main concepts in the annealing process. The *energy* is viewed as the value of the cost function. Since we have framed our optimization problems in terms of minimization, this corresponds to lower energy states in the annealing process. The algorithm's *temperature* refers to the effect of changes to a solution on the value of the cost function; a high level of variability in solutions corresponds to a high temperature of the system. As the temperature is lowered, less variability to the optimal solution is allowed. If a variation of the current optimal solution lowers the value of the cost function, then this new solution is set as the current optimal solution. If the variations do not lower the value of the cost function, then the solution is selected based on some probability distribution (note that this means that a solution whose associated cost is higher than the current optimal solution has some probability of being selected as the optimal solution; in other words, small steps in the wrong direction are allowed probabilistically). The probability distribution used in determining whether a non-improving solution is selected was first described in 1953 [**6**] and is known as the Metropolis algorithm. This algorithm can in fact be used to not only decide these probabilities but also for generating the population space of solutions to be considered for the optimization problem. In order to make some physical connection with the simulated annealing process, we can view the probability of a non-improving solution being selected as being analogous to the physical system being cooled overall, but exhibiting a certain amount of bubbling as the material is cooled.

In the terminology of heuristic algorithms, this means that the solution path is allowed to make 'uphill' moves; that is, there is some probability that a temporarily worse solution may be accepted at any given temperature. Note that not all solutions have the same probability of being

selected; those whose cost value is closer to the cost value of the current optimal solution have a higher chance of being selected than those whose cost values are much higher than the cost value of the current optimal solution. The notion of uphill moves is an important feature of many heuristic algorithms, and it serves the purpose of attempting to avoid being stuck with a solution that is a local minimum while there may be better solutions elsewhere in the solution space. In genetic algorithms, for example, the mutation step serves much the same purpose. Many heuristic algorithms incorporate some method of solution perturbation to achieve this goal.

The amount by which the temperature is lowered and the time which is spent at each temperature are referred to as the annealing schedule. This schedule can be modified very easily on a computer and thus simulated annealing has the advantage of ease of implementation. For systems that are non-deterministic polynomial-time hard (NP-hard), the worst case scenario refers to the upper bound on the amount of calculations that must be undertaken in order to find an optimal solution. One advantage of simulated annealing is that for larger scale models, this worst-case scenario becomes increasingly unlikely [**5**]. Another advantage is that as the size of the model increases, the computational expense of the algorithm tends to increase at a rate that is not much higher than the order of the model size (i.e., within a small power of the model size). Thus the algorithm scales well and can be used on complex models. For an application to combinatorial optimization problems, see [**7**] and [**8**].

SIMULATED ANNEALING – PSEUDO-CODE

0. Define cost function **E**( ), representing the 'energy' of a solution.
1. Set initial variability level T , representing the 'temperature'.
2. Define *minTemp*, the minimum temperature.
3. Define **Prob***( )*, probability function for accepting a less-fit solution.
4. Generate s_1, random initial solution.
5. Set current optimal solution, $s_{opt} = s_1$.
6. While($T \geq minTemp$) do

    While (*equilibrium condition not satisfied*) do

        s = **ChooseNeighbor**(s_{opt});

if ($E(s) < E(s_{opt})$) then

$s = s_{opt}$;

end

else if (**Prob**(T, $E(s)$, $E(s_{opt})$) > **random**(0,1))

$s = s_{opt}$;

end

end

DecreaseTemp(T);

end

7. **Return**(s_{opt}).

2.4 Tabu Search

The method of Tabu search was introduced by Fred Glover in 1989 ([**9, 10, 11**]). The algorithm begins much like genetic algorithms and simulated annealing: an initial (perhaps random) solution is evaluated, and then a local search is done for better solutions nearby in the solution space. In this case, the precise meaning of 'nearby' is determined by a suitable metric imposed on the solution space. The local search that is performed, however, is not exhaustive. Instead, a series of rules (or 'moves') are examined in order to determine the best nearby solution. These moves are referred to as the *tabu list*; this list is defined by the moves that are *not* allowed.

For example, the tabu list may consist of the previous five solutions; thus, in the next search, none of these previous five moves may be selected. In this case, the idea is to avoid repetition and reversals: it is important to avoid reversals because they represent a waste of the previous step, and it is important to avoid repetition because it is possible to get stuck in a loop or on a particular solution that is not anywhere close to optimal. Of the two issues, reversals are perhaps more important to avoid; this is because repetition may actually be informative if it is not immediate. That is, it may be that returning to a previously visited solution can be a good move depending on the solutions that have been examined since the solution was previously considered. The tabu list need not consist only of previous moves (in fact, it need not consist of previous moves at all). Another choice may be that any solution that would cause an increase of

the cost function is placed on the tabu list. As the search proceeds, more and more solutions would be added to the tabu list, thus minimizing the effective search space for some period of time. In implementation, there is typically a statute of limitations on the time a solution remains on the tabu list; thus after a pre-determined number of iterations, a solution that was previously on the tabu list may be considered as an acceptable solution. Tabu searches can also involve a feature known as an *aspiration level*: this is a threshold that is set at the beginning of the algorithm. Ordinarily, a solution that is on the tabu list cannot be accepted; however, if the cost evaluation of that solution is within the aspiration level of the current optimal solution, then the solution is permitted. Thus solutions that are very good can override the restriction placed on them by membership on the tabu list.

An advantage of tabu searches is that they are typically more aggressive; that is, there is never any reference to local minima and so the possibility of finding a local minimum too quickly is not a risk. This is in contrast to simulated annealing, wherein the gradual lowering of the temperature has the eventual effect of making local minima hard to escape. A disadvantage is the possible complexity of determining the rules by which the tabu list is updated; the user can implement any particular scheme they choose, but there may not be any mathematical justification for doing so. For an application of tabu search to combinatorial optimization, see [**12**].

TABU SEARCH – PSEUDO-CODE

0. Set initial solution s_1.
1. Set current optimal solution, $s_{opt} = s_1$.
2. Set tabu list $T = \{\ \}$.
3. Define **Optimal**(*s*, *N*), the method for finding optimal solution in a neighborhood *N* of solution *s* (this function may call another algorithm, for example).
4. Define **UpdateTabuList**( ), a function that updates *T*.
5. While(*stopping condition not satisfied*) do
   While (*SearchSpace not empty*) do

Set *SearchSpace* = **Neighborhood**(s_{opt}) – T;

s_{opt} = **Optimal**(s_{opt}, *SearchSpace*);

UpdateTabuList( );

end

end

6. **Return**(s_{opt}).

2.5 Ant Colony Optimization

Ant colony optimization (ACO) [**13**], alternately referred to as Ant System, is another heuristic algorithm based on a physical system. In this case, the inspiration comes from the behavior of ants in a colony [**14**]; in particular, it refers to the foraging behavior of the ants and how they travel to a food source. As an ant ventures out of the colony, it deposits a pheromone trail behind it. The more ants that follow a specific path, then, the more the contribution to the accumulated pheromone deposit on that particular path. Suppose, for instance, that a path from the ant colony to the food source involves the crossing of one of two bridges, and that one bridge is substantially longer than the other. In this case, the pheromone trail will become stronger on the shorter bridge because it takes less time to traverse, and thus the pheromone will evaporate at a slower rate. Over time, then, pheromones accumulate on the shorter path. Subsequent ants follow the strong pheromone trail with a certain probability; thus over time, the path to the food source converges to the shortest path.

ACO attempts to implement an analogous system in order to search the solution space; while many of the features present in actual ant colonies are modeled in the algorithm, there are also features that lack a physical counterpart. In particular, while real ants follow continuous paths in search of food, the virtual ants move in the discrete solution space in search of an optimal solution. In terms of a discrete model optimization algorithm, virtual ants traverse sequences of the control input space, the totality of which is considered a path (i.e., a solution). How the ants decide which path to take may initially be random, or have some other initial configuration determined by the user, depending on the nature of the problem.

The algorithm proceeds as follows: a population of virtual ants is created, and each takes a path based on the initial probability distribution. A pheromone matrix is then constructed based on the number of ants that selected a certain value for each control input, with a higher pheromone value being attributed to those paths (solutions) that led to lower cost functions; in other words, the better a path found by an ant, the more it contributes to the pheromone trail. As subsequent ants search for paths, they now use the pheromone matrix to calculate their probabilities for how they will traverse the control input space; thus, the more ants that choose a good path, the more likely that future ants will follow a similar path. However, the pheromones simply represent probabilities; thus there is no guarantee that an ant will follow the strongest path on any given run. Finally, an evaporation function is applied to the pheromone matrix that rescales the probabilities by some set amount, ensuring that the sum of all possible probabilities for each choice is exactly 1. The evaporation process allows paths that have not been followed in a long time to lose their strength, indicating some possibility that those paths were perhaps only locally optimal.

The number of ants, the initial probability distribution, the pheromone update schedule, and the evaporation rate are all variables that can have considerable effect on the results of the ACO algorithm. As is the case with all heuristic algorithms, most of these variables will be problem-dependent. For examples of specific methods for updating the pheromone matrix and the evaporation process, see [**13**]. For a case study using ACO as a method of combinatorial optimization see [**15**]; for a more general treatment of ACO in solving a more abstract mathematical problem, see [**16**].

ANT COLONY OPTIMIZATION – PSEUDO-CODE

0. Set initial pheromone matrix *P*( ) so that there is an equal probability of an ant choosing each admissible control value at each point in the sequence.
1. Define *m*, the number of ants.
2. While(*stopping condition not satisfied*) do
   PathSet = **GeneratePaths**(*m*, *P*);
   EvaluateAndSortPaths(*PathSet*);

s_{opt} = *PathSet*[1];

UpdatePheromoneMatrix(*P*, *PathSet*);

EvaporatePheromone(*P*);

end

3. **Return**(s_{opt}).

2.6 GRASP

GRASP refers to greedy randomized adaptive search procedures, which constitute a class of algorithms that are well-suited for large-scale models [**17**]. It can be combined with other algorithms as sub-processes, and thus is most aptly described as a meta-heuristic.

The overall process involves two phases: construction and search. The first value for a control variable in a solution is selected from some restricted candidate list (RCL); this list may include all possible values of the variable (if the number of choices is finite) or it may have some other problem-specific delineation. The next control variable value in the sequence is then selected based on an adaptive greedy function of an RCL of possible subsequent control variable values. This process is repeated until an entire solution has been constructed.

The search process then uses the constructed solution as a starting point for its local search: an exhaustive local search may be performed, or some other algorithm may be implemented to find a local optimum for the constructed solution. The way in which subsequent solutions are constructed will also involve other algorithms: it will be dependent upon the rules governing the creation of the RCL, as that is the starting point for initial constructions.

One advantage to GRASP is that the process consists of only two phases; thus it is easily implemented to be run in parallel on multiple computer processors. It also allows the user the flexibility of choosing different algorithms to implement in both of the two steps. This flexibility naturally induces the corresponding disadvantage that the implementation tends to be fairly problem-specific. If we wish to choose a different algorithm to implement there may be substantial effort involved in redefining how the RCL is constructed. Another potential disadvantage of GRASP is that it does not have a solution perturbation mechanism, so there is

perhaps a greater risk of the search returning a local optimum instead of a global optimum. For an example of GRASP as a method of combinatorial optimization, see [**18**].

GRASP – PSEUDO-CODE

0. Set *G*, a parameter in [0,1] that defines how greedy the construction process should be (0 representing maximum greediness).
1. Generate an initial solution, s_{opt}.
2. Define *N*(*s*), a neighborhood of a solution *s*.
3. Define **UpdateRCL**(s), a function that specifies how the restricted candidate list should be updated based on the current construction of *s*.
4. While(*stopping condition not satisfied*) do
   s = { };
   for *i* from 1 to *SolutionSize* do
      s[i] = **SelectFromRCL**(*G*);
      UpdateRCL(*s*);
   end
   s_N = **Optimal**(*s*, *N*(*s*));
   if (cost(s_N) < cost(s_{opt})) then
      $s_{opt} = s_N$;
   end
   end
5. **Return**(s_{opt}).

2.7 Evolutionary Squeaky Wheel Optimization

Squeaky wheel optimization [**19**] is another meta-heuristic algorithm. The approach of this method is to ‘repair’ solutions rather than working with an entirely new solution from scratch (hence the name ‘squeaky wheel’, referring to the portion of the solution to be repaired).

A solution is a sequence of control inputs; in some settings, a natural grouping of certain inputs arises. We refer to these groups as components.

Consider the example given in the previous chapter, wherein the optimal control problem is to determine a treatment schedule that maximizes the healthy cell count and minimizes the number of required treatments. Suppose we wish to extend this example so that we now wish to consider several different drugs. This may require us to reformulate the optimal control problem; for example, perhaps we now wish to minimize the actual monetary cost of producing the drug and minimize the time it takes to produce a large quantity of it. Then in our model there will be several control inputs: production cost and production time (both of which can be thought of in monetary terms), and the treatment schedule. In this case, we see a natural delineation of the control inputs into two components: monetary-based and schedule-based.

We may find that there are many solutions that maximize the healthy cell count (i.e., many solutions which all produce the highest number of healthy cells, allowing for some variation due to simulation). Then in the search for an optimal solution, we need only to evaluate, or 'repair', the monetary-based component of the solutions. So we view the monetary component as the squeaky wheel, and our goal is to repair this component. In general, there may be many control inputs and many components, and we may be repairing more than one.

The squeaky wheel optimization process consists of three basic stages: construction, analysis, and prioritization. In the construction stage, a greedy algorithm is used to construct a solution. During analysis, the solution is examined and the components that cause the most 'trouble' are marked for repair. Those that cause more trouble than others are assigned a higher priority for repair in the prioritization stage. The process is then repeated, with the greedy algorithm now attempting to repair trouble components in order of priority. A benefit of this process is that time is saved by only investigating components that need to be altered, thus reducing the necessary search time. However, two immediate disadvantages of this process are that it is highly dependent on the initial solution, and the convergence rate to an optimum is slow because only small changes are being made to the solution at each step.

In order to rectify these issues, a more elaborate version of this heuristic process, evolutionary squeaky wheel optimization (ESWO), has been developed [**20**]. ESWO consists of five stages: analysis, selection, mutation, prioritization, and construction. The stages are similar to those just described, with a few minor alterations. In the analysis stage, solutions are examined

and components are evaluated. The selection stage then assigns each component a certain probability of being repaired, based on the amount of trouble it causes in the solution. The mutation stage is similar to selection, with components being selected for repair according to fixed probability (regardless of the amount of trouble they cause). The prioritization stage is the same as above: it determines how the components should be updated based on information from the analysis. In the construction stage, information from previous stages is used in order to repair components that have been selected for repair (either in the selection or the mutation stage).

The advantage of this process is that for each of the selection, mutation, and construction stages, there is an associated probability matrix that can be computed in a straightforward fashion. Then the product of these three matrices is easily computed, and represents the effects of one cycle of the algorithm. Another advantage is that since each component has some non-zero probability of being selected for repair, it is assured that the global optimum is reachable during any cycle with probability 1 (for proof see [**20**]). The process also adapts very well to changes in any of the individual stages, because the associated matrices can be formed independently, and the re-calculation of the overall transition matrix is easily performed. A disadvantage is that these matrices will grow in size with increasing complexity of the model, and thus may become too large to be effective for systems with large numbers of control inputs.

SQUEAKY WHEEL OPTIMIZATION – PSEUDO-CODE

0. Set M, the mutation rate.
1. Generate initial solution s_1 and set $s_{opt} = s = s_1$.
2. For a solution s, separate full set of control inputs into n disjoint components $C_1(s), \ldots, C_n(s)$.
3. Define **Analyze**(s), a function that evaluates the cost associated with s and determines the relative harm done by each component of solution s (this can be done, for instance, by setting a benchmark level for each component).
4. Define a length n vector *Harm*[ ], whose k^{th} entry is the probability for repairing component C_k based on the results of the **Analyze**( ) function.

5. Define **Selection**(*s*, *H*), a function that assigns probability for each component to be repaired based on the harm threshold and the results of the **Analyze**( ) function.
6. Define **Repair**(*s*, *RepairList*), a function that repairs components on the repair list by local search or other optimization method.
7. While(*stopping condition not satisfied*) do

    Analyze(*s*);

    RepairList = { };

    for *i* from 1 to *n* do

        if (**Random**(0,1) < *Harm*[C_i(*s*)]) then

            RepairList = *RepairList* $\cup$ C_i(*s*);

        end

    for *i* from 1 to *n* do

        if (**Random**(0,1) < *M*) then

            RepairList = *RepairList* $\cup$ C_i(*s*);

        end

    Sort(*s*, *Harm*);

    s_{new} = **Repair**(*s*, *RepairList*);

    if (cost(s_{new}) < cost(*s*)) then

        $s_{opt} = s_{new}$;

    end

    else **GenerateNewSolution**( );

    end
8. **Return**(s_{opt}).

2.8 Variable neighborhood search

A variable neighborhood search (VNS) is a heuristic method that focuses on the issue of local optima. While most algorithms run the risk of converging to a local optimum, the variable neighborhood search is performed to ensure that this does not happen. A VNS works as follows:

the solution space is split into a finite number of disjoint neighborhoods N_1, … N_t, which are ordered in some way. A random solution is generated in N_1, and a local search is performed in order to find a local optimum (this local search may be any of the previously described algorithms, for example). If this local optimum is better than the best solution that has been found so far, then it is stored as the best current solution. The process is repeated in each neighborhood.

This process can become very computationally expensive depending on the local search method used in each neighborhood; however, there are no negative consequences of achieving a local optimum too quickly (as there is in simulated annealing, for example). The algorithm is also amenable to variations: we may choose to search only certain neighborhoods, or to order our neighborhoods in such a way as to search farther away in the solution space so that differing regions of the solution space are considered.

In [**21**] several variations are discussed. The first is random VNS (RVNS); in this scenario, the local search is abandoned altogether, and one random solution from each neighborhood is compared. This has the advantage of substantially less computing time. In another variant, the local search is not applied to the entire solution, but rather to components of it; thus we may think of the local search as being a squeaky wheel optimization. A third variant relaxes the condition for acceptance of a new current best solution: if the solution is within a certain threshold of the current best solution, it is accepted as the current best solution. Furthermore, solutions whose distance in the solution space is further from the current best solution are given preference for acceptance. This has the effect of combating stochastic variation and the risk of being stuck at a local optimum. Two pseudo-code algorithms are shown here; the first is a general form of the VNS and the second is a simple RVNS.

VARIABLE NEIGHBORHOOD SEARCH – PSEUDO-CODE 1 (VNS)

0. Separate solution space into *k* disjoint neighborhoods N_1, … , N_k.
1. Define function **NeighborhoodSearch**(*s*) to find the optimal solution in the neighborhood containing *s* (this function may call previous algorithms, for example).

2. Generate a random solution s in N_1; set $s_{opt} = s$.
3. for i from 2 to k do

    s = **RandomSolution**(N_i);

    s_{new} **= NeighborhoodSearch**(s);

    if (cost(s_{new}) < cost(s_{opt})) then

        $s_{opt} = s_{new}$;

    end

    end
4. **Return**(s_{opt}).

VARIABLE NEIGHBORHOOD SEARCH – PSEUDO-CODE 2 (RVNS)

0. Separate solution space into k disjoint neighborhoods $N_1, \ldots, N_k$.
1. for i from 1 to k do

    $s[i]$ = **RandomSolution**(N_i);

    sort(s[ ]);

    end
2. **Return**(s[1]).

2.9 Population indifference zone

The objective of the population indifference zone method [**22**] is phrased slightly differently than previous algorithms: the goal is to find a solution that has probability P of being within δ of the global optimum. That is, it does not promise to find the global optimum, nor necessarily even seek to; it simply seeks out a solution that has a certain probability of being *very*

close to the global optimum. The algorithm relies more heavily on statistical analysis than the others in order to achieve this goal.

A random sample of solutions is selected from the solution space. The cost function is evaluated for each (actually, the average of the cost function upon running a simulation a number of times). The best solution is chosen from this random sample, and its associated cost is labeled *b*. The standard deviation from *b* of each of the cost functions associated with the other solutions is then calculated. Those that are within a certain δ-threshold of *b* are kept; those that are not are discarded (this is where the algorithm takes its name, as the number of solutions that fall within the threshold will vary, so the algorithm is indifferent with respect to the sample population size). All solutions that have been kept are run again in the simulation a number of times. The standard deviation is re-calculated. A particular weight function (given in [**22**]) is then evaluated for each of the solutions. If any of the solutions are determined to be satisfactory, then the algorithm terminates. Otherwise, the kept solutions comprise 10% of the next generation of solutions, and an additional 90% are generated either by random selection or the use of another heuristic algorithm. The algorithm repeats on the new generation of solutions, and it continues until a solution is found that satisfies the conditions given in the weight function.

One benefit of the population indifference zone algorithm is that it is easily adapted to make use of other algorithms; in a sense, it is a meta-heuristic because it focuses on the analysis of the solutions rather than the means by which the solutions are obtained. Moreover, Allen and Vuckovich [**22**] have proven that given enough generations, the algorithm will converge to within some range of the optimal solution. A drawback of this algorithm is that it has potential to be expensive, and convergence may require an unfeasible number of generations.

POPULATION INDIFFERENCE ZONE – PSEUDO-CODE

0. Define δ, the acceptance threshold.
1. Define weight function **Weight**(*s*) to determine the fitness of solution *s*.
2. Set *method* to be user's choice of solution selection (any previously described algorithm, for example).

3. Define a function **GenerateSolutions**(*method*, *n*) that uses *method* to generate *n* solutions.
4. P_1 = **GenerateSolutions**(*method*, *n*).
5. Set $i = 1$.
6. While(*stopping condition not satisfied*) do

    Evaluate(P_i);

    Sort(P_i);

    b = cost($P_i[1]$);

    P_{i+1} = *{ };*

    for j from 2 to *size*(P_i) do

        if (cost($P_i[j]$) < $b + \delta$) then

            $P_{i+1} = P_{i+1} \cup P_i[j]$;

        end

    end

    Evaluate(P_{i+1});

    for k from 1 to *size*(P_{i+1}) do

        if (**Weight**($P_{i+1}[k]$) < *WeightThreshold*) then

            $s_{opt} = P_{i+1}[k]$;

            GoTo(Step 7);

        end

    end

    P_t = **GenerateSolutions**(9 * *size*(P_{i+1}));

    $P_{i+1} = P_{i+1} \cup P_t$;

    $i = i + 1$;

    end
7. **Return**(s_{opt}).

---

3. Agent-Based Models

3.1 Overview

Agent-based models provide means for examining a complex system by studying the dynamics of the system via simulation. We use the following from [**23**] as a definition for this paper:

> **Definition:** An **agent-based simulation** of a complex system is a computer model that consists of a collection of agents/variables that can take on a typically finite collection of states. The state of an agent at a given point in time is determined through a collection of rules that describe the agent's interaction with other agents. These rules may be deterministic or stochastic. The agent's state depends on the agent's previous state and the state of a collection of other agents with whom it interacts.

Thus agent-based models are a means of computational analysis for complex systems. By only allowing agents to assume a finite number of possible states and by placing the dependence of those states on a series of rules, agent-based models are well-suited for implementation by computer. With the advent of advanced computing capability, the models we have at our disposal have grown in numbers and complexity. Systems such as the human immune system (which were previously too difficult to accurately represent via computer simulation) are now in widespread use as more and more research relies on simulation to study the properties of this and other complex systems. For a good tutorial on agent-based models see [**24**].

In developing the best drug to administer to treat a particular disease, researchers in the past have relied on a trial-and-error method via repeated experimentation on live subjects. With an accurate and valid model of the subject, however, such experiments can now be done *in silico*; that is, they can be run on a computer and analyzed without having to carry out as many physical experiments. This can save money by precluding the necessity of preparation and the purchasing of equipment; perhaps more importantly, computer simulation saves time. Given an accurate model of the immune system, we can run a simulation and make predictions on results of tests

administered over the course of years or decades, all within a matter of minutes or hours of computer time. This allows time that was previously being devoted to experimentation to be devoted instead to developing fresh ideas and making predictions based on the results of the simulations.

Experimentation remains part of the process; a model can only be so predictive and must be correlated against empirical data from time to time in order to ensure that the models are indeed faithful simulations of the actual physical system; however, the time taken to do this is much less than it was in the past.

Agent-based models have another distinct advantage over continuous models: for complex systems, the number of equations used in a continuous model may be well into the thousands or even orders of magnitude higher than that. Typically, such systems are solved and analyzed by computers through numerical methods. But computers can only handle a certain amount of 'continuity'; that is, at a certain level, everything run by the computer must be discretized. From this point of view, it is perhaps more natural to work with discrete models as opposed to continuous models. Agent-based models in particular have the advantage of being well-suited to model many different types of systems. They are used to study social interactions among individuals, the spread of disease through populations, the scheduling and efficiency of factory processes, and how cells react to drug treatments, to name a few. There are also many computational resources for the implementation of agent-based models that are currently used in the scientific community; one such free software program is NetLogo [**25**]. In addition, many articles in which agent-based models are used make the models freely available.

Finally, it is worth noting that many of the current issues in the scientific community are interdisciplinary. Finding a cure for cancer will involve geneticists, biologists, mathematicians, chemists, and perhaps many other specialists. While differential equations are suitable for use by mathematicians, they do not translate very well in an interdisciplinary setting. Agent-based models, on the other hand, have an intuitive formulation and can be examined via a graphical interface. Thus they are a natural tool for promoting interdisciplinary research, as the mathematics underlying the models is hidden in the programming; in other words, it is possible for biologists and chemists to make use of agent-based models without a full background in the mathematics that are involved in creating the model.

3.2 Current Issues in Agent-Based Modeling

While agent-based models provide a convenient and natural setting for studying complex systems, there are several issues within the mathematical research community that are currently unresolved. A 2006 paper [**26**] written by a large group of researchers identify two main obstacles: the first is the lack of standardization of the description of agent-based models, and the second is a lack of rigorous mathematical formulation of the system itself.

Descriptions of agent-based models can vary in different settings, and as of yet there is no standard definition that is agreed upon. Some models are developed to simulate physical processes, and others are developed in the framework of graph theory. In some cases models are developed in order to study only a certain aspect of the system; thus the model may have more variables pertaining to certain processes than others. An article may begin with the statement of an objective, or it may begin with a description of the model itself. In some cases, the rules that govern the updating of the agents' state variables are deterministic, and in other cases they are stochastic. All of these issues would be resolved if there was a standard protocol for describing agent-based models; Grimm et al [**26**] proposes one such protocol. The need is stressed for a structure to be followed in the presentation of the model; in particular, the layout of the model ought to be standardized so that when an article is read, it is evident to the reader what the model describes, what its rules are, and where in the description they can expect to find the same type of information. The current literature presents models in myriad forms, and the descriptions of the agents, environments, and rules come in no particular order. Thus a reader is required to scour the paper for pertinent details that might otherwise be presented in some standard way.

The second major issue concerns the lack of rigor in the formulation of a model. In many cases, the description of a model is given in several paragraphs, describing in a vague way what the agents are and how they interact with each other. In fact, in order for agent-based models to be implemented on computer software, there are intricate rules embedded in the programming of the models. These rules and equations are often glossed over if not entirely omitted; if they are given, it is typically through a verbal description rather than a strict mathematical formulation. For instance, an author may spend several paragraphs explaining how the model was formulated that could just have easily been given in one or two equations. Short and precise definitions can save time for the reader and also make the author's intentions explicit. In an effort to overcome

this lack of formalization, a publication by Hinkelmann et al [**27**] proposes a rigorous mathematical representation for agent-based models as polynomial dynamical systems; further description of such systems can be found in [**28**].

3.3 Simulation Optimization

Simulation optimization has been an area of study as long as computers have been in use. It is the study of optimization of any type of simulation process, and thus includes the study of agent-based models as one method of simulation. For a brief tutorial on the subject, see [**29**]. Both continuous and discrete models can be simulated by computer, and thus either type of model may be encountered in the broad arena of simulation optimization. In general, the goal of simulation optimization is to find a set of parameters, variables, or coefficients that, when implemented in the model, can be expected to produce the best results. Of course, what is meant by 'best' is defined by the user: it may be that in a simulation of a factory and the machines that operate within it, we wish to determine a work schedule that maximizes production. On the other hand, we may want to know what the optimal schedule is in order to maximize the profit resulting from selling the goods. It may be true that the best schedule is the same in either case, but not necessarily (for example, maximizing production may necessitate more repairs, which would lessen overall profit).

3.4 Optimal Control of Agent-Based Models

Agent-based models provide a framework for analyzing complex systems and allow researchers to simulate processes that might otherwise be very expensive or unfeasible to perform. But by themselves, the models are simply simulations of the system and nothing more; it is not until they are used to answer specific questions that the importance of agent-based models is fully realized. Just as optimal control theory has been developed in order to answer specific questions about continuous and discrete systems, the same principles can be applied to agent-based models.

Here *optimization* refers to obtaining the best parameter values in light of the optimization objective. For example, a scientist may begin with a question such as the following:

given a patient with a certain type of cancer, there are several possible drugs that can be administered as treatment. In order to maximize the patient's life expectancy and minimize the tissue damage caused by the drug, which treatment should be administered? With each such question there is an associated cost function, as described in Chapter 1. The cost function is an equation that takes data from the simulation of an agent-based model as input and returns as output a value that corresponds to the aspects of the model that are of interest. In the preceding example, the cost function would be a function of two variables: the life expectancy of a patient and the collateral tissue damage. If we then chose to ask about the monetary cost of producing the drug and wished to minimize that cost in addition to controlling the two above-mentioned variables, we would have to reformulate the cost function to incorporate monetary cost as a variable. Note too that the drug that is the cheapest to produce may not correspond to the longest possible life expectancy; thus we may attach weights to the variables to make one variable more important in the evaluation of the cost function than the other variables.

Thus a single agent-based model can be used to answer many questions, and the optimal control will depend on the goal of the study. A model may be developed with certain control objectives in mind, or we may simply wish to use an existing model to ask questions that arise naturally in the course of studying the model.

3.4.1 History of Optimal Control of Agent-Based Models

The history of agent-based models goes as far back as the 1940s, when John Von Neumann first introduced the concept of cellular automata; however, it wasn't until nearly fifty years later that agent-based models came to be used widely in the scientific community. While the rules that govern how an agent moves and interacts can be simple, the dynamics generated by them are complex; as the number of agents increase in a model, the computational resources required grow exponentially. Thus the implementation of agent-based models is mainly an issue of computing power; the resources that are required to run even relatively simple models were simply not available to the general public until the 1990s. In fact, one of the earliest modern-day uses of an agent-based model was introduced by Craig Reynolds in 1987 [**30**], wherein the modeling technique was described and subsequently became the basis of computer graphics as used in cinema. It is only in the last ten years or so that optimal control has been used with agent-based models, due largely in part to the computational resources involved.

3.4.2 Disambiguity of Optimization as Applied to Agent-Based Models

There are two basic goals that may both be described as optimization and its relation to agent-based modeling. The first is that we have discussed above: obtaining a solution or a sequence of control variable values that return the most favorable results based on some pre-defined criteria. However, there is another aspect of agent-based modeling that has attracted some attention recently as well: the optimization of the model itself. The models that scientists use are, almost without exception, meant to reproduce some process that corresponds to a real-world scenario. For example, the process we wish to simulate may be the spread of an epidemic through a population or the assignment of different tasks to a number of machines. In either case, we typically wish to use the results obtained from the optimization process in order to implement those results in the corresponding real-world scenario. Thus our results are only as useful as the model itself. Given a model that does not accurately reproduce the dynamics of the actual system, we cannot expect our results to be very informative. Thus some scientists have studied means by which *the model itself* can be optimized; that is, how to construct the model so that it more faithfully reproduces the dynamics of the system we wish to study (or at least reproduces the dynamics that we are interested in). For an example of such a study, see [**31**] and [**32**]. On the other hand, see [**33**] and [**34**] for examples of studies in which agent-based modeling is used as a means of answering a specific question (in other words, the agent-based modeling is implemented in the algorithm itself; once constructed, the result obtained *is* the answer).

For the purposes of this paper, optimization and optimal control of agent-based models refers to the goal stated earlier in this chapter: given an agent-based model and an associated cost function (based on our objective), what choice of control inputs leads to the minimization of the cost function? The issue of optimizing the models themselves or using agent-based models as a means of constructing one-off solutions is not examined in this paper.

4. Case Studies

In this chapter we examine literature on several agent-based models; in particular we examine methods of optimization and optimal control as applied to these models and discuss the results that were obtained. Recall that in optimization problems, we search for the best solution without a particular idea of the final state of the system, whereas in optimal control, a particular state is sought and we search for the solution that leads to that state in the most efficient way. In many studies, the problem is formulated as an optimization problem because the best state the system can achieve is not known *a priori*. However, optimal control is often used as an intermediate step in this process. From empirical data, we know that certain states are achievable and can thus formulate optimal control problems and use cost functions to determine the optimal control; we might then use that solution to inform the optimization process. The following studies use this hybridization of optimization and optimal control.

4.1 Immune system response to HIV infection

In a 2007 paper, Castiglione et al [**35**] discuss results obtained from an agent-based model of the human immune system. The model is formulated to simulate the immune systems response upon being infected with HIV; prior to control, the model dynamics of an infected virtual patient can be observed over a period of time. At the end of the simulation, certain measurements are taken. These include the viral count (that is, the number of infected cells) and the number of CD 4 T helper cells, among many others. For such infections, standard practice is to administer a Highly Active AntiRetroviral Therapy (HAART) in order to control the damage caused by the infection. While it is not yet possible for these treatments to eradicate HIV, they have the effect of prolonging the life expectancy of an infected patient. The model is a three-dimensional stochastic cellular automata in which the agents are the major classes of cells of lymphoid lineage and some of the major classes of cells of myeloid lineage.

Virtual patients are infected with HIV and the effect on the immune system is simulated for a period of over seven years, in order for the infection to reach the chronic phase. During this time, no treatment is administered. Once the chronic phase is reached, the patient undergoes treatment for a period of six months. Empirical data from real patients were used to implement

the effects of HAART on the immune system model; in particular, HAART modifies the life cycle of HIV in the model by preventing spread of infection at certain measured rates. These observed rates can then be incorporated into the simulation of the model by altering the HIV life cycle during time periods when HAART is administered. In actual practice, HAART is a method of administering a drug cocktail on a daily basis; the authors argue that daily administration is an unrealistic expectation for a patient and thus wish to find a treatment schedule that obtains similar results but requires the patient to take fewer dosages of the cocktail. Thus the optimization goal is to determine a treatment schedule that minimizes the viral count, maximizes the immune restoration (as measured by the number of CD 4 T helper cell lymphocytes), and minimizes the number of times the treatment must be taken.

The six-month period is split into time increments of one week, for a total treatment time of 24 weeks. During each week, the patient either takes the HAART or the patient does not (taking the HAART indicates that the cocktail is taken each day of that week). The control variable in this situation is the treatment schedule; the only aspect of the model that the user has direct control over. Recall that a solution is a sequence of inputs to the control variable; so in this case, a solution is a vector of length 24, of which each entry is either a 0 (indicating treatment was not administered that week) or a 1 (indicating that treatment was administered that week). The basic process of simulating this agent-based model and its response to various treatments is to select a solution, implement it in the model, and run the simulation. In order to account for stochastic variability, each solution may be simulated ten or even twenty times. The cost function is then evaluated, using the number of times treatment was administered (i.e., the number of 1's in that solution), the average viral count observed over the total number of simulations, and the average number of CD4 T helper cell lymphocytes observed over the total number of simulations. In particular, the cost function is a sum of three terms: the HIV load (i.e., the sum of the free and proviral load divided by the initial viral point at initiation of treatment), the CD4 fitness (i.e., the ratio of CD4 level after treatment to the CD4 level just prior to treatment), and the amount of therapy (i.e., the number of active therapy days divided by the total number of days in the simulation).

In order to find an optimal control (in this case, an optimal treatment schedule) the authors use a genetic algorithm. Each solution is viewed as a chromosome consisting of 24 genes, each of which is either expressed (i.e., given the value '1') or not expressed (given the value '0').

A population consists of 32 different treatment schedules (a subset of the 2^{24} possible solutions); this population size was selected based on computational considerations. The initial population of solutions was generated using tournament selection. Each solution in the population is then simulated 16 times in order to account for variability (we can also think of this as applying the same treatment schedule to 16 different individuals). The cost of each solution is then evaluated. Uniform crossover is used as the breeding protocol within the genetic algorithm, and elitism is used on the two best solutions from the population. Subsequent populations are generated by using a certain percentage of solutions from the current population and then filling out the rest of the population via the breeding process, as described in Chapter 1. As the specific algorithm used in this paper is proprietary information, full detail is not available as to the specifics of the algorithm. However, the authors do indicate that a total of 60 generations were bred using the algorithm which means that altogether, $32 \times 16 \times 60 = 30720$ simulations were run.

Upon completion of the treatment period, virtual bacteria injections were given to the patients in order to test the efficacy of the treatment. This was done in order to simulate the ability of the patient's immune system to kill bacteria after the six-month treatment period. The virtual patients were then followed for three months after cessation of treatment and the survival rates were tabulated. The authors found that of the three factors used in the cost function, the amount of therapy was the best indicator of the survival expectancy rate.

The use of heuristic algorithms can expedite the search for a good control sequence for a given problem, but a question always remains: how can we be sure that the best solution we have found is actually the optimal solution, and not just the best that we were able to find? For example, in this situation, how can the authors be sure that the best solution they obtained by using the genetic algorithm did not depend on the initial population, or that there might not be a better solution elsewhere in the solution space that was not used in simulation? In order to answer that question, the authors compare the results that they obtained with results obtained from three different control groups: a *void* treatment schedule in which no treatment was administered, a *full* treatment schedule in which treatment was administered during each of the 24 possible weeks, and a *random* treatment schedule in which the number of treatments was equal to the number of treatments in the algorithms best solution, but the distribution of treatments throughout the six-month period was random. The best solution as determined by the genetic algorithm and the three solutions just mentioned were run an additional 200 times in the

simulation and the survival rates were tabulated. The authors observed the following survival rates: full treatment – 34.l1%; solution obtained from genetic algorithm – 30.50%; random treatment – 21.86%; void treatment – 20.25%.

In terms of expected survival rate, it is reasonable to assume that full treatment is the best possible treatment; however, note the similarity between the rates of the solution obtained by the genetic algorithm and that obtained by full treatment. Furthermore, full treatment requires patients to take the HAART cocktail every week for 25 weeks, while the genetic algorithm solution only involves treatment during 15 of the 25 weeks. A treatment schedule with 15 randomly selected weeks for treatment, on the other hand, leads to a survival rate that is very close to the expected survival rate for void treatment. This can be interpreted to mean that not only is the number of treatment weeks important, it is important which weeks they are. Randomly selecting 15 weeks for treatment is not much better than foregoing treatment altogether; this is a convincing argument that the genetic algorithm is truly finding an optimal solution rather than just producing random results via repeated breeding and mutation of solutions.

Note too that it is possible that there is an even better solution in the solution space that simply was not encountered by the algorithm; in order to be absolutely sure, each of the 2^{24} solutions would have to be run (in fact, they would have to be run a number of times in order to eliminate variability due to the stochastic nature of the model). Since each simulation was run 16 times to eliminate variability, this means that in actuality only $32 \times 60 = 1920$ different solutions were considered out of 2^{24} total solutions. Thus only

$$100 \times \left(\frac{1920}{2^{24}} \right) \approx 100 \times \left(\frac{1}{2^{13}} \right) \approx 0.012\%$$

of the solution space was examined in order to achieve these results, which shows that that this heuristic algorithm was able to find a good solution by examining only a small fraction of the solution space. Furthermore, while the full treatment schedule shows a better survival expectancy rate, it is also less realistic in practice. For example, the monetary cost of producing the drugs (or the cost to the patient of buying the drugs) was not included at all in the cost function. In reality, these costs may lower the feasibility of full treatment as a viable option. It is possible, as well,

that other factors than the three used in the cost function are better predictors of a patients expected survival.

4.2 Determining a vaccination protocol for pandemic influenza

A model of a small town in the United States has been studied by Longini since 1978 [**36**] and has been in use since that time. The current incarnation is an updated version of the model described by Halloran in [**37**]. This version of the model can be described as an agent-based model consisting of 10,000 agents, each of which represents one person. The 2000 U.S. Census was used to determine the current age distributions and the family structure of the individuals; five communities were delineated, consisting of four neighborhoods each. In order to more accurately model the interaction between people in the virtual population, each community contained two elementary schools, a middle school, and a high school. Younger children were also placed into play groups based on their neighborhoods. The agents in the model travel back and forth to the locations that are associated with their age group (schools, hospitals, etc.) according to rules described in the model. This structure is well-suited for the task of studying the spread of an influenza pandemic because it models realistic interactions of different age groups in the population, each of which have different rates of susceptibility, infection, and recovery in response to an influenza attack.

Patel et al published a study in 2005 [**38**] focusing on two particular influenza pandemics: the Asian flu (a 1957 – 1958 pandemic) and the Hong Kong flu (a 1968 – 1969 pandemic). The population was divided into five age groups. Data from these particular strains of the flu were used to determine the attack rates of each strain on each of the five age groups. The optimization problem is stated as follows: “given a limited quantity of influenza vaccine and a particular population structure and illness attack rate pattern for a single wave of pandemic influenza, what proportion of each age group should be vaccinated to minimize the impact of the epidemic?” ([**39**], p.203).

Two different interpretations are used to determine the impact of the epidemic; in one case the goal is to minimize the number of illnesses that occur, and in the other case the goal is to minimize the number of resultant deaths. Since two interpretations were used, two cost functions were constructed; then since two different strains of influenza attack rate data were used, a

number of problems are being examined at once: for a given strain, and a given goal (i.e., minimizing illness versus minimizing number of deaths), what is the optimal distribution of the available vaccines among the population?

In this case, since there are five age groups, a solution is a vector of length 5; the entries of this vector are the *proportion* of the corresponding age group that is given a vaccine prior to the start of the influenza outbreak. It is assumed that all available vaccines will be administered, and that each person who receives a vaccine receives only one dose. The authors also note that it is more dangerous for an infant (or an elderly citizen) to become infected than it is for a young adult; thus *weights* are attached to the age groups in order to signify these various risk levels. The *average illness attack rate* is defined to be the proportion of people in each age group that become infected over the course of the simulation (note that some of these people have been vaccinated but may become infected regardless). Then to state the optimization goal more explicitly, it is the attempt to minimize the sum over all age groups of the product of each group's *proportion*, *weight*, and *average illness attack rate*. Note that the average illness attack rate is observed upon simulation and cannot be chosen directly, and the proportion of each age group that receives a vaccine is dependent upon the number of available vaccines. Thus the optimal solution depends on two factors: the number of available vaccines and the weights that are given to each age group; changing either of these factors will affect the optimal solution.

As in the previous example, a genetic algorithm is used to determine the optimal solution. In this case a solution is viewed as a chromosome consisting of five genes, each of which can take any real value between 0 and 1 (representing the proportion of each age group that is given the vaccine). Note that if the number of vaccines exceeds the population size, then everyone can receive a vaccine. However, if the number of vaccines is less than the population size then it may not be possible to vaccinate every member of even one age group, let alone all five. Thus not all vectors of length five with real entries between 0 and 1 are admissible; an admissible vector is determined by the number of vaccines (all of which must be used).

A random set of 50 solutions is generated as the initial population. Each is implemented in the model and the model is run 20 times; the average value of the cost function is calculated over these 20 simulations. The solutions are then sorted in terms of fitness; the best 25 solutions are kept in the next generation. The remaining 25 solutions in the next generation are determined by breeding the top 25 from the previous generation. The breeding process is as follows: 10 of

the 25 solutions are selected at random. Of those 10, the best solution is selected as the 'father' in the breeding process. This process is repeated in order to select a 'mother', with the added condition that the father and mother cannot be the same solution. A new 'child' solution is then bred – for each gene in the solution, there is an 80% chance that the child will inherit the value of the father's gene and a 20% chance that the child will inherit the mother's gene. Recall that the proportions of each age group to be vaccinated were dependent on the number of available vaccines, and thus not all solutions are admissible (because they may require more people to be vaccinated than is possible). Thus the 'child' solution that is bred may not be admissible; as such, it may be necessary to adjust the proportions in order for the solution to become admissible. This evening out process is referred to in the general discussion of genetic algorithms in Chapter 2 as the mutation step; the new solution begins with values from the parent solutions, then undergoes a certain randomization process, which in this case involves altering the values slightly so as to become an admissible solution. Twenty-four other solutions are bred in the same manner (using the initial pool of 25 as parents) so that the second generation now consists of the top 25 solutions from the first generation and 25 that have been bred from those. This process is repeated until four consecutive generations fail to produce a better solution; that is, until the best solution remains the best solution for four generations in a row.

The authors compared the optimal solution found by the genetic algorithm with the optimal solution found by another method, referred to as random mutation hill climbing (RMHC). In the RMHC algorithm, a solution is generated randomly. A subsequent solution is generated by randomly altering the value of each gene, and then normalizing the resultant values so that the produced solution is admissible. If the solution produced is better than the original, it replaces the original as the optimal solution and the procedure continues. Otherwise, the procedure continues with the current solution as the optimal solution. This process was repeated until the optimal solution was not improved upon for 100 consecutive attempts.

The algorithms were run for two different strains of influenza, the Asian flu and the Hong Kong flu. Optimal solutions were obtained for nine different vaccine availability levels (from 10% to 90% of the population). The genetic algorithm was not shown to be better than RMHC at a statistically significant level, and for this reason only results obtained from the genetic algorithm are given in the paper. Since the two strains of influenza have different attack rates for each of the age groups, the optimal solution differs in each case. The Asian flu has a higher attack rate

for school-aged children, thus when the goal was to minimize illnesses the optimal solution tended towards vaccinating mostly that age group, and for the Hong Kong flu the optimal solution favors a balance of vaccinating school-aged children and young adults. On the other hand, when the goal was to minimize the number of deaths, the optimal solution found for the Asian flu suggested higher vaccination levels for the elderly, which shifted to a balance between elderly and school-aged children at higher vaccination availability levels. For the Hong Kong flu, the optimal solution for minimizing deaths followed this same trend of vaccinating the elderly and school-aged children. The proportions of which age group to vaccinate varied quite a bit as the vaccination availability levels change; in particular, these proportions do not seem to follow a very predictable pattern (particularly when the goal was to minimize the number of resultant deaths). It is noted that for the Hong Kong flu, it is not possible to stop an epidemic when the vaccine availability level is below 40%, but for the Asian flu it is possible with only 20% availability.

Once their results were obtained, the authors implemented random vaccinations at each availability level in order to ensure that the results obtained were actually better than those that might have been obtained by simply vaccinating a random proportion of each age group. For each of the nine availability levels (10 – 90%), the results obtained by the genetic algorithm were better than those obtained by random vaccination (both in terms of minimizing illness and in terms of minimizing deaths). In particular, the results were markedly better at availability levels of 30 – 40%. This may seem odd at first but with a bit of thought it is not so strange: if only 10% of the population can be given a vaccination, there is not a lot of freedom in deciding who to give the vaccinations to; thus the results obtained by randomly assigning vaccinations is probably nearly as good as the optimal solution obtained by a search algorithm. At 90% availability, the same is true: nearly everybody can be given a vaccine, so the exact distribution of vaccines is not as important and we can achieve similar results by administering the vaccines randomly. However, in the mid-ranges there is a lot of flexibility in terms of how the vaccines can be administered; thus a random administration has a higher chance of not working out as well as the carefully obtained solution that comes from the genetic algorithm.

In order to make a connection to real-life practices during influenza outbreaks, the policies of the Center for Disease Control and Prevention (CDC) were examined; it was found that in most cases the protocol is to administer vaccinations as quickly as possible to as many

people as possible. At first glance it may appear that this is more or less a policy of random administration, but in fact it is not quite so. This is because not all of the age groups will appear at the vaccination sites in equal measure; adults, for example, are perhaps more likely to show up and receive a vaccination than infants or the elderly because of their increased mobility.

In light of these findings, the results found in this study are highly informative and useful. If an influenza outbreak occurs, a similar simulation to that performed here can be implemented almost immediately once the attack rates and vaccine availability levels are determined for that strain; the optimal solution that is found can then be used to inform the vaccination policy, which was shown in this case to be significantly more effective than random vaccinations in preventing an epidemic. On the other hand, we must consider the feasibility of implementing a chosen vaccination policy. For example, our simulation may tell us that we ought to vaccinate 80% of school-aged children, but in fact it will be up to the parents to make sure that the children are vaccinated. In general, the policy can be decided upon but it will be up to the citizens to show up and receive the vaccination; in this sense, it may be difficult to implement any such policy.

Another issue is the generalization of these results to a larger population. After all, the model used in this study was made up of only 10,000 people. The authors concede this point but note that due to the regional climate differences in the U.S., influenza outbreaks occur during different times of the year in different areas and thus are in fact somewhat localized. In addition, they argue that the dynamics involved in the model are not based in any specific way on the population size; in fact, the results may well be generalized simply by viewing each of the 10,000 agents as entire families instead of individuals, or even neighborhoods. A modification of the model may be necessary in order to justify this viewpoint, but it is reasonable to suppose that such modifications are possible. For a different treatment of solving a similar problem, see [**40**].

4.3 Vaccination protocol for tumor prevention in mice

The final case study for this chapter was first introduced by Pappalardo et al in 2005 [**41**]. The authors developed a model entitled SimTriplex that simulates the immune system of mice; in particular, the model is designed to study the response of the immune system when tumor cells are introduced. The SimTriplex model was derived from C-ImmSim [**42, 43**], a model of the human immune system. In this model the agents are cells, and the interaction of the cells is

determined by rules that were implemented according to empirical data. Triplex is a specific vaccine designed to prevent cancer; SimTriplex is used to determine the effectiveness of Triplex administration in preventing tumor growth in virtual mice. This case study evolves over the course of several papers, all of which discuss different methods of achieving the same goal: developing a vaccination protocol (i.e., a vaccination treatment schedule) that is most effective at preventing tumor development. Throughout the literature, references and comparisons are made to the Chronic treatment – this treatment consists of administering a vaccine at every possible time step. As such, it is used as the control group; in particular, the optimal control problem is to determine a vaccination protocol that produces results comparable to that of Chronic treatment while minimizing the number of dosages of the vaccine that must be taken. Since the agents in this model are individual cells, a tumor is considered to have formed if the number of cancer cells exceeds 10,000 at any time during the simulation.

In 2006 Lollini et al [**44**] attempted the use of a genetic algorithm to determine a vaccination protocol with SimTriplex. The virtual mice were treated for 400 days with the possibility of vaccination occurring once every 8 hours. Thus altogether there are 1200 treatments possible; the Chronic treatment is the administration of the vaccine during each of these 1200 time steps. The cost function depends on two factors: the number of injections and the survival time of the mouse. The genetic algorithm used is not given explicitly in this paper, but the authors did report that tournament selection was used to generate solutions, uniform crossover was employed as the breeding method, and mutation and elitism were also incorporated (elitism refers to giving preferential status to solutions that produce better results). While a threshold is set for the formal formation of a tumor, it is noted that if the number of cells approaches the threshold without necessarily crossing it, a tumor may form in later stages and other serious complications are introduced in the immune system (i.e., an increased risk of carcinogenesis). Thus a third factor was introduced into the cost function to modify the threshold level of tumor formation. In this case, the algorithm found a solution that met the cost function threshold level requirements, but the solution was found by examining only one virtual mouse. When this solution was applied to a larger population, it was shown to prevent tumor growth in only 20% of the mice. Elsewhere in the paper, results are obtained in which 88% of the mice remain tumor-free for 400 days, but the details of the modifications to the algorithm are not given. The Chronic treatment schedule has the same expected survival rate; thus the authors

conclude that it is possible to determine a treatment schedule that is as effective as Chronic treatment but with fewer injections. It is worth noting that the run-time of this algorithm is given as 72 hours (using 32 nodes on a high-performing computer cluster).

Pennisi et al revisited this problem in a 2008 paper [**45**]; they attempted to improve upon previous results by the use of a simulated annealing algorithm. The goal remains the same: to determine a vaccination schedule that is as effective as Chronic treatment but minimizes the number of injections. A population of 200 virtual mice was examined, with 8 mice selected as the simple random sample. Recall that simulated annealing algorithm (see Chapter 2) is a simulation of the cooling of a physical system and the corresponding lowering of the overall energy of the system. In this case, the authors also wish to follow the mice for 400 days (i.e., 57 weeks) and allow for two vaccinations to be administered per week; thus in this setting the Chronic treatment involves administration of 114 injections. In simulated annealing, the temperature corresponds to the control variable; thus in this case, the temperature of the system is the number of injections. At each temperature level, a minimal energy level is attained; once attained, the temperature is lowered and the process repeats. This continues until an absolute minimal energy level is obtained. Since we wish to maximize the survival time of the mice, the energy level is viewed as the reciprocal of the survival time; thus in order to maximize survival we wish to minimize this energy level. Then the process is as follows: beginning with an initial temperature of 114, the optimal vaccination protocol is searched out (at this temperature level, it will be the Chronic treatment). Then the temperature is lowered, say, to 113. A new minimum-energy equilibrium is then sought, and the process continues. The algorithm stops when it is not possible to attain energy equilibrium at that temperature (i.e., when the temperature becomes so low that the threshold conditions for survival time cannot be satisfied). As in the genetic algorithm search, safety thresholds are implemented in the cost function that determine the survival time of the mice based on the number of tumor cells. Once the threshold is reached, the mouse is considered to have died.

As noted above, the genetic algorithm was able to obtain a solution that resulted in 88% of the mice surviving for 400 days. In this paper, the simulated annealing approach obtained similar results (a survival rate of 86.5%). The methods used were similar; each approach involved determining the solution using a sample size of 8 mice, and then applying that protocol to the population of 200 mice. The emphasis in the simulated annealing paper is not on the

effectiveness of the results (since they are comparable to solutions that were previously known) but on the computational resources required to obtain these results. The simulated annealing approach took only 2 hours to run on an 8 processor unit to obtain the same results as the genetic algorithm; thus the computing time is on the order of 10^2 faster than the genetic algorithm. Finally, it is noted that while both algorithms obtain similar results, the treatment schedules are not the same. This shows that there may be more than one optimal solution, or that there may well be another solution that neither algorithm found that is even better than those discussed. In other words, it is possible that both algorithms found solutions that were only locally optimal but that the solution space may contain other better solutions elsewhere.

A review of this problem was presented by Pappalardo et al in 2009 [**46**]. Both the genetic algorithm and the simulated annealing algorithm are discussed, in addition to a continuous-model approach. While the results from the heuristic algorithms seem to be as effective as the Chronic treatment, the authors argue that they remain unsatisfactory. This is because of the cancer-cell threshold level; the complications caused by cancer cell levels rising too closely to the threshold of tumor formation in actuality ought to reduce the expected survival time of the mice.

In order to test the results of the computer simulations, the treatment schedules were implemented on actual mice in a wet lab. The in vivo experiments resulted in an 81% survival rate, notably lower than the rate predicted by the algorithms. However, it is noted that the final five injections prescribed by the computer simulations were not actually given because the biologists determined that they would not affect the survival rate.

The results of this series of papers seems to show that not all heuristic algorithms are equal; in other words, the fact that the simulated annealing approach achieved similar results to the genetic algorithm in only a fraction of the computing time seems to indicate that in this case, simulated annealing is a better approach. Of course, this does not indicate that simulated annealing is better than genetic algorithms in general; it is likely problem dependent, and may also be impacted by the setup and implementation of the algorithm. For example, the genetic algorithm involved solutions of length 1200, while in the simulated annealing approach there was a maximum of 114 injections; thus the solution space is much smaller and so it is reasonable that the search would take less time.

5. Conclusions

The aim of this paper has been to examine optimal control and optimization of agent-based models. We have reviewed what agent-based models are, how and why they are used, and discussed several methods. Some of the methods are currently in use by the mathematical and scientific community, and other methods have not been used but could adapt readily to fit into this framework. In this section, we give a brief overview of the methods in use and then discuss possible future directions for further research. In particular, some detail is given on how an agent-based model can be formulated as a polynomial dynamical system (PDS). Pseudo-code is given for two heuristic algorithms discussed in chapter 2, adapted for use with a general PDS.

5.1 Current methods

Heuristic algorithms are the preferred means for performing optimal control and optimization of agent-based models. Typically, agent-based models are designed to simulate some physical process or interaction, and as such, they tend to be complex models with many variables; in addition, the rules that govern how the agents interact are frequently complicated as well. As more and more biological data is collected and used to create the models, the corresponding validity of the model increases and the simulations become more realistic. But the cost of this increased realism is in computation time, and as the models grow more complicated so too do they increase in their resource requirements. Thus in a typical optimization or control problem, it is impossible to enumerate the solution space to find optimal solutions. Heuristic algorithms provide a means for searching the solution space in an effective manner in order to produce near-optimal results in a reasonable timeframe. Thus the attention given to such algorithms is understandable; the more precise and effective the algorithms we are able to develop, the quicker we will be able to search for an optimal solution.

However, heuristic algorithms are not the only methods being used for optimal control of agent-based models. Several publications by Peter Kim explore another method of optimal control. In [**47**], Kim begins with a complex biological model, approximates the model with a series of difference equations, and then formulates a much smaller deterministic model. Since deterministic models can be solved outright by computer algorithms, Kim obtains a solution

from the deterministic model and shows that it is comparable to the solution obtained by a heuristic search of the original stochastic model. In [**48**], Kim approximates the original stochastic model by a series of partial differential equations and then uses theory from mathematics to obtain an optimal solution of the resulting continuous model. Thus two different methods are used to reduce and transform the form of the model without compromising the integrity of the model.

This shows that it while it is possible to build ever more complicated models, it is not necessarily true that a high level of complexity is required in order to solve specific control problems. It may be possible in many cases to reduce models by approximating variables that do not play a large role in the process we wish to control; in such cases simpler models can be constructed that maintain the relevant model dynamics. A general framework for such model reductions would be greatly beneficial to the study of agent-based models and is currently not a well-developed area in the study of discrete systems in general and agent-based models in particular.

In Chapter 3, it was noted that Hinkelmann et al [**27**] have devised a means of translating agent-based models into polynomial dynamical systems. In that paper, several examples are given of how that is accomplished; in section 5.3 we present a brief overview. The benefit of such a representation is that years of algebraic theory become immediately available for the study of such systems. By grounding the models in a form that has been studied for many years, analytical tools can be used that would be otherwise unavailable.

5.2 Future research

One of the issues prevalent in the literature on agent-based models is deciding which type of control method should be used for a given problem. For example, in the case studies in Chapter 4, genetic algorithms and simulated annealing are used, but little justification is given as to why. Presumably it is because these algorithms are well-documented and software already exists (in Matlab, for example) that implements these algorithms. But it may well be that another heuristic algorithm would be even better suited for a given problem; as noted in the final case study, the simulated annealing approach was shown to be as effective as the genetic algorithm in only a fraction of the computing time. As yet, there are no established criteria for deciding which

method to use to solve a given optimal control or optimization problem of an agent-based model. Part of the problem is that, as noted earlier, a formal representation of agent-based models is lacking in the mathematical community. Without a formal representation, it is impossible to delineate models for the application of different control methods; hence such a formalization is critical to the future study of agent-based models.

Kim's method (discussed in 5.1) shows that model reduction and model approximation are possible in certain instances. However, there is no proof that such methods would work for all models and for all control problems. It would serve the study of agent-based models well to know exactly how, and under what conditions, a model can be approximated either by a smaller deterministic model or in some other way. As yet, there are no formal criteria for answering this question.

Another need in this area is increased software development. If an agent-based model is encoded as a polynomial dynamical system (PDS), for example, then it would be useful to have software readily available that could perform any number of heuristic algorithms on the PDS in search of an optimal solution. Such software would be even more useful in light of the fact that it would only need the PDS and the goal of the optimal control problem as input; thus it would not have to be altered for every individual problem. It could be used with any agent-based model so long as the model is represented as a PDS.

5.3 Polynomial Dynamical Systems (PDS)

We end this paper by giving an example of the form an agent-based model takes when represented as a PDS and then presenting pseudo-code adapting two of the algorithms from chapter 2 for use with a general PDS. A tutorial on how an agent-based model can be represented as a PDS is given in [**27**]. In short, a polynomial dynamical system is a dynamical system in which the rules describing agent interactions take the form of polynomials in multiple variables over a finite field. The agents are described by collections of variables; state variables have values that correspond to different aspects of the agents and spatial variables describe some aspect of the environment. The dynamics of the model are generated by iteration of the polynomials. An agent-based model consisting of n agents is represented as a PDS consisting of

n polynomials. Each polynomial f_i defines how the variable x_i is updated at each time step; since a state variable's value often depends on complicated interactions of the other state and spatial variables, these polynomials generally involve many variables (though not always).

Recall that each variable (and the coefficients of the variables) are elements of finite fields, though the fields may be different for different types of variables. When forming an agent-based model from the literature, we may begin by creating a truth table describing how the variables interact. This is our first step in creating a PDS: from the truth table, we can use an interpolation formula to generate polynomials that produce the desired output.

We illustrate this interpolation formula with a simple example: suppose that we have four state variables $x_1,...x_4$, we know that x_1 has values based entirely on the values of the other three variables, and we wish to study the effect of x_2, x_3, x_4 on x_1. In this example, we will suppose that we are working in $\mathbb{F}_2$; that is, the Boolean field consisting of two elements. Then suppose that our truth table contains the following information:

x_1	x_2	x_3	x_4
1	0	0	0
0	0	0	1
0	0	1	0
1	0	1	1
1	1	0	0
1	1	0	1
0	1	1	0
1	1	1	1

In light of this information, we wish to obtain $f_1(x_2, x_3, x_4)$; that is, we wish to find a polynomial over $\mathbb{F}_2$ such that for the given values of x_2, x_3, x_4, the result is the value of x_1 given by the table. We do this by adding eight terms; one for each row of the table. For example, note that the term $(1-x_2)(1-x_3)(1-x_4)$ has value 1 when $x_2 = x_3 = x_4 = 0$ and has the value 0 in all other cases; this corresponds precisely to the first row of our table. In fact, we can construct

similar terms for all rows in which x_1 has the value 1: for rows 4, 5, 6, and 8 we obtain the respective terms $(1-x_2)x_3x_4$, $x_2(1-x_3)(1-x_4)$, $x_2(1-x_3)x_4$, and $x_2x_3x_4$. Since each term evaluates to 1 precisely for the row values given in the table, they will all evaluate to 0 in all other cases and thus other terms need not be considered; in particular, this truth table is represented by the polynomial

$$f_1 = (1-x_2)(1-x_3)(1-x_4) + (1-x_2)x_3x_4 + x_2(1-x_3)(1-x_4) + x_2(1-x_3)x_4 + x_2x_3x_4,$$

which can be simplified to

$$f_1 = 1 + x_3 + x_4 + x_2x_4 + x_2x_3x_4.$$

Similar constructions can be repeated for each of the variables – in this way a PDS is obtained from the truth table. In fact, such polynomials can be created for any (finite) number of variables over any finite field, not just $\mathbb{F}_2$; for a detailed treatment of such constructions see [**28**].

Formulating agent-based models as polynomial dynamical systems allows for a consistent presentation and for standardization of analysis techniques. In the present myriad forms, analysis must be done on a case-by-case basis, which in many cases means that new software and new code must be written for each model (and updated for each change to a model). Converting agent-based models to PDS form has the benefit of ease of implementation: software that is written to handle a PDS need not be re-written for a different agent-based model; the equations and the control objective are the only necessary input. Once a PDS is obtained from the agent-based model, more specified versions of the algorithms discussed above can be implemented. We conclude this paper with pseudo-code for two algorithms: the first is a genetic algorithm and the second is simulated annealing. We hope the advantages of these methods have been made clear in the course of this paper and in forthcoming research further implementations will be carried out.

GENETIC ALGORITHM – PSEUDO-CODE FOR PDS

Input: Mobile state variables $x_1,...,x_k$; spatial state variables $y_{k+1},...,y_n$.

Initial values of each state variable $x_1,...,x_k, y_{k+1},...,y_n$.

Control variables $c_1,...,c_d$ (each c_i corresponds to some state variable).

Finite fields $\mathbb{F}x, \mathbb{F}y, \mathbb{F}c$ indicating state field for each type of variable.

Polynomials $f_1,..., f_n$ describing the update rule for each state variable.

Number of time steps *t*.

Cost function *Cost*( ) (e.g. sum of all/some of the state variable values after *t* time steps).

Size *n* of each solution population, a positive even integer.

Inheritance rate I_p for use in breeding step.

Mutation rate *mu*.

Number of simulations *s* to run each solution.

Optional: update schedule *P* for state variables.

(*continued on next page*)

0. Generate a set of n initial solutions as P_1, the initial population. P_1 is a set of vectors of length $d \cdot t$ with entries in $\mathbb{F}c$ (each block of t consecutive entries representing a sequence of inputs to one control variable).

1. i = 1; *numTimesBest* = 0; **Sort**(P_1). % See pseudo-code below.

2. While($numTimesBest < 10$) do % Continue until best
   P_t = {}; % solution is not improved
   for j from 1 to $\frac{n}{2}$ do % upon for 10 consecutive
      $P_t = P_t \cup P_i[j]$; % generations.
   end
   $P_{i+1} = P_t$;
   for k from 1 to $\frac{n}{2}$ do
      P_r = **ChooseTenRandom**(P_t);
      Sort(P_r);
      $P_f = P_r[1]$;
      $P_m = P_r[2]$;
      P_c = **Breed**(P_f, P_m); % See pseudo-code below.
      Mutate(P_c); % See pseudo-code below.
      $P_{i+1} = P_{i+1} \cup P_c$;
   end
   Sort(P_{i+1});
   if ($P_{i+1}[1] = P_i[1]$) then
      $numTimesBest = numTimesBest + 1$;
   else $numTimesBest = 0$;
   end
   $i = i + 1$;
   end

3. **Return**($P_i[1]$).

Sort( ) FUNCTION PSEUDO-CODE

Input: Solution population *P*.

Global variables: cost function *Cost*( ), number of simulations *s*, initial values of state variables.

0. $C = \{\ \}$.
1. for *i* from 1 to size(*P*) do
   $tempCost = 0$;
   for *j* from 1 to *s* do
   tempCost = *tempCost* + *Cost*(**Simulate**($P[i]$));
   end
   $C[i] = tempCost / s$; % Determine average cost of solution.
   end
2. $P_t = \{\ \}$;
3. for *i* from 1 to size(*P*) do
   $k = address(\min(C))$; % Find index of minimum cost value.
   $P_t[i] = P[k]$; % Store solution as first entry of P_t.
   $C[k] = \max(C) + 1$; % Ensure this entry will not be minimum
   end % again by making it the new maximum.
4. **Return**(P_t).

Breed( ) FUNCTION PSEUDO-CODE

Input: Two solutions P_f and P_m.

Global variables: P_f probability inheritance, I_p.

0. $P_c = \{\ \}$;
1. for *i* from 1 to size(P_f) do
   if ($random(0,1) < I_p$) then
      $P_c[i] = P_f[i]$;
   else
      $P_c[i] = P_m[i]$;
   end
   end
2. **Return**(P_c).

Mutate( ) FUNCTION PSEUDO-CODE

Input: Solution P.

Global variables: Mutation rate *mu* , control variable field $\mathbb{F}c$.

0. for *i* from 1 to size(P) do
   if ($random(0,1) < mu$) then
      $P[i] = random(\mathbb{F}c)$;
   end
   end
1. **Return**(P_c).

SIMULATED ANNEALING – PSEUDO-CODE FOR PDS

Input: Mobile state variables $x_1,...,x_k$; spatial state variables $y_{k+1},...,y_n$.

Initial values of each state variable $x_1,...,x_k, y_{k+1},...,y_n$.

Control variables $c_1,...,c_d$ (each c_i corresponds to some state variable).

Finite fields $\mathbb{F}x, \mathbb{F}y, \mathbb{F}c$ indicating state field for each type of variable.

Polynomials $f_1,..., f_n$ describing the update rule for each state variable.

Number of time steps *t*.

Initial variability level *T* (i.e., temperature).

Rate *d* at which temperature is decreased.

Minimum temperature *minTemp*.

Cost function *Cost*( ).

Mutation rate *mu* for use in neighbor selection.

0. Generate s_1, random initial solution.
1. Set current optimal solution, $s_{opt} = s_1$.
2. While($T \geq minTemp$) do

   numTimesBest = 0;

   While($numTimesBest < 10$) do % Equilibrium condition.

      s = **Mutate**(s_{opt}); % See **Mutate**( ) code above.

      $energyChange = Cost(s) - Cost(s_{opt})$;

      if ($energyChange < 0$ or $random(0,1) < e^{(-energyChange/T)}$ then

         $s = s_{opt}$;

      else $numTimesBest = numTimesBest + 1$;

   end

   $T = d \cdot T$; % Lower the temperature.

   end
3. **Return**(s_{opt}).

5.4 Summary

Agent-based models are used in many areas of research and have applications to many real-world problems and situations. They provide a natural framework for investigating the complicated dynamics and relationships that exist in large systems. These models are easy to implement on computers and can be modified to simulate many different processes. Using agent-based models as tools for simulation, time and money can be saved and questions of critical importance can be answered.

In this paper, we have established what agent-based models are and how they are used. In particular, we have defined and examined optimization and optimal control in this framework. We have examined the terminology used in the field and several of the methods that are currently being used for optimization and optimal control. We have discussed case studies in which various methods have been implemented and the results that were obtained.

While these methods are capable of producing good results, several drawbacks still remain. The lack of a standardized presentation of agent-based models is perhaps the biggest issue in the field; without a rigorous formulation it is difficult to apply mathematical theory to such problems without applying arbitrary limitations or assumptions. In order to further the study of optimal control of agent-based models, it is necessary that the model format be standardized so that the mathematics can be uniformly applied. One such presentation, that of polynomial dynamical systems, has been discussed. We have given a brief example of how one can convert an agent-based model into a PDS, and discussed the advantages of doing so.

It is imperative that these methods be more fully developed so that the theory behind the mathematics can be brought to bear on problems that have not yet been examined with full mathematical rigor. Once this goal is accomplished, the analysis of agent-based models (particularly in the framework of optimization and optimal control) will advance. This will allow for these real-world problems to be addressed in a new light, offering insight and solutions that are currently unobtainable.

6. Bibliography

1. A. H. Land and A. G. Doig, *An automatic method of solving discrete programming problems*, Econometrica **28** (1960), no. 3, 497 - 520.

2. D. E. Goldberg, *Genetic algorithms in search, optimization, and machine learning*, Addison-Wesley, Reading, MA, 1989.

3. J. H. Holland, *Adaptation in natural and artificial systems*, University of Michigan Press, Ann Arbor, MI, 1975.

4. J. R. Koza, "Evolution and co-evolution of computer programs to control independently-acting agents," *Proceedings of the first international conference on simulation of adaptive behavior on From animals to animats*, MIT Press, Paris, France, 1990.

5. S. Kirkpatrick, C. D. Gelatt and M. P. Vecchi, *Optimization by simulated annealing*, Science **220** (1983), no. 4598, 671-680.

6. N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller and E. Teller, *Equations of state calculations by fast computing machines*, Journal of Chemical Physics **21** (1953), no. 6, 1087-1092.

7. S. Y. Kim, S. B. Lee and J. Lee, *Structure optimization by conformational space annealing in an off-lattice protein model*, Phys Rev E Stat Nonlin Soft Matter Phys **72** (2005), no. 1 Pt 1, 011916.

8. K. Bryan, P. Cunningham and N. Bolshakova, *Application of simulated annealing to the biclustering of gene expression data*, IEEE Trans Inf Technol Biomed **10** (2006), no. 3, 519-525.

9. F. Glover, *Tabu search: A tutorial*, INTERFACES **20** (1990), no. 4, 74-94.

10. ---, *Tabu search--part i*, INFORMS JOURNAL ON COMPUTING **1** (1989), no. 3, 190-206.

11. ---, *Tabu search--part ii*, INFORMS JOURNAL ON COMPUTING **2** (1990), no. 1, 4-32.

12. X. Zhang, T. Wang, H. Luo, J. Y. Yang, Y. Deng, J. Tang and M. Q. Yang, *3D protein structure prediction with genetic tabu search algorithm*, BMC Syst Biol **4 Suppl 1** (2009), S6.

13. M. Dorigo, V. Maniezzo and A. Colorni, *The ant system: Optimization by a colony of cooperating agents*, IEEE Transactions on Systems, Man, and Cybernetics Part B: Cybernetics **26** (1996), no. 1, 29-41.

14. M. Dorigo, G. Di Caro and L. M. Gambardella, *Ant algorithms for discrete optimization*, Artif Life **5** (1999), no. 2, 137-172.

15. A. Shmygelska and H. H. Hoos, *An ant colony optimisation algorithm for the 2D and 3D hydrophobic polar protein folding problem*, BMC Bioinformatics **6** (2005), 30.

16. D. Merkle and M. Middendorf, *Modeling the dynamics of ant colony optimization*, Evolutionary Computation **10** (2002), no. 3, 235-262.

17. T. A. Feo and M. Resende, *Greedy randomized adaptive search procedures*, Journal of Global Optimization **6** (1995), 109 - 134.

18. S. Dharan and A. S. Nair, *Biclustering of gene expression data using reactive greedy randomized adaptive search procedure*, BMC Bioinformatics **10 Suppl 1** (2009), S27.

19. D. E. Joslin and D. P. Clements, *"Squeaky wheel" Optimization*, Journal of Artificial Intelligence Research **10** (1999), 353 - 373.

20. J. Li, A. J. Parkes and E. K. Burke, *Evolutionary squeaky wheel optimization: A new analysis framework*, Evol Comput (2011).

21. P. Hansen and N. Mladenovic, *Variable neighborhood search: Principles and applications*, European Journal of Operational Research **130** (2001), no. 3, 449-467.

22. T. Allen and D. Vuckovich, *An open-source population indifference zone-based algorithm for simulation optimization*, Winter Simulation Conference, Winter Simulation Conference, 2010.

23. R. Laubenbacher, A. S. Jarrah, H. S. Mortveit and S. S. Ravi, "Mathematical formalism for agent based modeling," *Encyclopedia of Complexity and Systems Science*, R. A. Meyers (Editor), Springer, 2009, pp. 160 - 176.

24. C. M. Macal and M. J. North, *Tutorial on agent-based modelling and simulation*, J of Sim **4** (2010), no. 3, 151-162.

25. U. Wilensky, "Netlogo. Http://ccl.Northwestern.Edu/netlogo/.", Center for Connected Learning and Computer-Based Modeling, Northwestern University, Evanston, IL, 2009.

26. V. Grimm, U. Berger, F. Bastiansen, S. Eliassen, V. Ginot, J. Giske, J. Goss-Custard, T. Grand, S. K. Heinz, G. Huse, A. Huth, J. U. Jepsen, C. Jørgensen, W. M. Mooij, B. Müller, G. Pe'er, C. Piou, S. F. Railsback, A. M. Robbins, M. M. Robbins, E. Rossmanith, N. Rüger, E. Strand, S. Souissi, R. A. Stillman, R. Vabø, U. Visser and D. L. DeAngelis, *A standard protocol for describing individual-based and agent-based models*, Ecological Modelling **198** (2006), no. 1-2, 115-126.

27. F. Hinkelmann, D. Murrugarra, A. S. Jarrah and R. Laubenbacher, *A mathematical framework for agent based models of complex biological networks*, Bull Math Biol (2010).

28. A. Veliz-Cuba, A. S. Jarrah and R. Laubenbacher, *Polynomial algebra of discrete models in systems biology*, Bioinformatics **26** (2010), no. 13, 1637-1643.

29. M. Fu, *A tutorial review of techniques for simulation optimization*, Winter Simulation Conference, Winter Simulation Conference, 1994, pp. 149 - 156.

30. C. W. Reynolds, *Flocks, herds and schools: A distributed behavioral model*, SIGGRAPH Comput. Graph. **21** (1987), no. 4, 25-34.

31. M. Gilli and P. Winker, *A global optimization heuristic for estimating agent based models*, Computational Statistics & Data Analysis **42** (2003), no. 3, 299-312.

32. S.-B. Shahab and C. A. Hunt, "Prediction of in vitro hepatic biliary excretion using stochastic agent-based modeling and fuzzy clustering," *Proceedings of the 38th conference on Winter simulation*, Winter Simulation Conference, Monterey, California, 2006.

33. A. Troisi, V. Wong and M. A. Ratner, *An agent-based approach for modeling molecular self-organization*, Proc Natl Acad Sci U S A **102** (2005), no. 2, 255-260.

34. S. Fortuna and A. Troisi, *Agent-based modeling for the 2D molecular self-organization of realistic molecules*, J Phys Chem B **114** (2010), no. 31, 10151-10159.

35. F. Castiglione, F. Pappalardo, M. Bernaschi and S. Motta, *Optimization of HAART with genetic algorithms and agent-based models of HIV infection*, Bioinformatics **23** (2007), no. 24, 3350-3355.

36. I. M. Longini, E. Ackerman and L. R. Elveback, *An optimization model for influenza A epidemics*, Mathematical Biosciences **38** (1978), no. 1-2, 141-157.

37. E. Halloran, I. M. Longini, D. M. Cowart and A. Nizam, *Community interventions and the epidemic prevention potential*, Vaccine **20** (2002), no. 27-28, 3254-3262.

38. R. Patel, I. M. Longini, Jr. and M. E. Halloran, *Finding optimal vaccination strategies for pandemic influenza using genetic algorithms*, J Theor Biol **234** (2005), no. 2, 201-212.

39. I. M. Longini, M. E. Halloran, A. Nizam and Y. Yang, *Containing pandemic influenza with antiviral agents*, American Journal of Epidemiology **159** (2004), no. 7, 623-633.

40. P. Kasaie, W. D. Kelton, A. Vaghefi and S. G. R. J. Naini, *Toward optimal resource-allocation for control of epidemics: An agent-based-simulation approach*, Winter Simulation Conference, Winter Simulation Conference, 2010.

41. F. Pappalardo, P. L. Lollini, F. Castiglione and S. Motta, *Modeling and simulation of cancer immunoprevention vaccine*, Bioinformatics **21** (2005), no. 12, 2891-2897.

42. F. Castiglione, M. Bernaschi and S. Succi, *Simulating the immune response on a distributed parallel computer*, Int J Mod Phys C **8** (1997), no. 3, 527-545.

43. M. Bernaschi and F. Castiglione, *Design and implementation of an immune system simulator*, Computers in Biology and Medicine **31** (2001), no. 5, 303-331.

44. P. L. Lollini, S. Motta and F. Pappalardo, *Discovery of cancer vaccination protocols with a genetic algorithm driving an agent based simulator*, BMC Bioinformatics **7** (2006), 352.

45. M. Pennisi, R. Catanuto, F. Pappalardo and S. Motta, *Optimal vaccination schedules using simulated annealing*, Bioinformatics **24** (2008), no. 15, 1740-1742.

46. F. Pappalardo, M. Pennisi, F. Castiglione and S. Motta, *Vaccine protocols optimization: In silico experiences*, Biotechnol Adv **28** (2009), no. 1, 82-93.

47. P. S. Kim, P. P. Lee and D. Levy, *Modeling imatinib-treated chronic myelogenous leukemia: Reducing the complexity of agent-based models.*, Bulletin of Mathematical Biology **70** (2008), 728 - 744.

48. ---, *A PDE model for imatinib-treated chronic myelogenous leukemia*, Bulletin of Mathematical Biology **70** (2008), 1994-2016.