UNIVERSITY OF WISCONSIN — MADISON
DEPARTMENT OF STATISTICS

Practical Model Building for Quantitative Population Ethology with Event-Driven Competing Risks

Bland Ewing[1]

Brian S. Yandell[2]

James F. Barbieri[3]

Robert F. Luck[4]

[1] Department of Entomology (Ret), University of California, Berkeley, CA
[2] Departments of Statistics and Horticulture, University of Wisconsin-Madison, WI
[3] Department of Entomology, University of California, Riverside, CA, and
Advanced Systems Development, NAWC-WPNS, China Lake, CA
[4] Department of Entomology, University of California, Riverside, CA

Technical Report # 1034

16 January 2001

Department of Statistics
University of Wisconsin-Madison
1210 West Dayton Street
Madison, WI 53706-1685

Phone: 608/262-2598
Fax: 608/262-0032
Internet: yandell@stat.wisc.edu

Practical Model Building for Quantitative Population Ethology with Event-Driven Competing Risks

Bland Ewing[1], Brian S. Yandell[2], James F. Barbieri[3], Robert F. Luck[4]

Abstract

Ideas have been presented elsewhere on a framework for quantitative population ethology using event-driven competing risks. In today's climate, it is not enough to have an idea--one must have an implementation of the idea. To that end, we present the beginnings of such a system built on a public domain, statistical system called R. We use spline-based graphical tools to craft the mean value functions of the competing risk structure, based either on data or on prior belief. Spatial location of individuals is developed on a hexagonal grid at the resolution of the model, with a triangular coordinate system to reduce computation of distances. The components can be quickly developed and tested in R, and later translated to C for more efficient coding of the most highly used loops. We also discuss some of the design requirements that affect the development of an efficient competing risk model of large problems.

[1] Department of Entomology (Ret), University of California, Berkeley, CA
[2] Departments of Statistics and Horticulture, University of Wisconsin-Madison, WI
[3] Department of Entomology, University of California, Riverside, CA and
Advanced Systems Development, NAWC-WPNS, China Lake, CA
[4] Department of Entomology, University of California, Riverside, CA

1 Introduction

In a previous paper (Ewing et al. 2001) we described an innovative modeling technique that allows one to model population dynamics at the level of resolution of individual members of a population. In that paper we describe the stochastic framework for such a model, but the simulation techniques were not discussed. The purpose of this paper is to discuss a prototype model simulation system and to outline a plan for more advanced model development.

Section 2 discusses the choice of software modeling system, namely R. Section 3 presents our scheme for conveying information about organisms, including species features, life stage events, movement and interactions among organisms. Future event scheduling is presented in Section 4, showing the competing risk structure and a sample simulation run as well as a graphical tool to design curves for scheduling future events. Span and resolution in space and time are discussed in Section 5. Here triangles form the building blocks for spatial relationships, leading to great time savings with little loss of precision. Further, the graphical mapping of hours to degree-days is presented. Further considerations, including some technical details and future plans, are sketched in Section 6.

2 Choice of Modeling System

The model is being developed in R, one of the S family of statistical languages (Venables and Ripley 2000). This system is in the public domain, is easy to learn, and includes standard statistical tools. In addition, it was designed from the start as an object-oriented language with device-independent graphical engines. Its chief drawback in this venture is that execution of loops can be slow. This is important since our model is executed serially rather than in parallel. This drawback can be overcome by "hardening" the code--that is, rewriting the inner loops in C or Fortran--once the basic fabric of the operations has stabilized.

We are developing a library of routines specific to the Event-Driven Competing Risk modeling effort (Ewing et al. 2001). Assuming one has installed the R system and has it running, the library is invoked by typing (after the ">" prompt):

- `library(ewing)`

This attaches a library of objects--data, functions, and possibly C or Fortran dynamically linked libraries.

The S system is a flexible and powerful environment for implementing statistical ideas that was developed at AT&T Bell Laboratories by Rick Becker, John Chambers and Allan Wilks. This system allows quick implementation of ideas, with easy-to-use graphical display tools that provide publication quality figures. The S system is available in a commercial form as S-plus from MathSoft (www.mathsoft.com) with GUI pull-down menus and dialogs. The book by Venables and Ripley (1999) is an excellent introduction to S, with comprehensive overview of basic and advanced features.

Recently, R has emerged as a public domain implementation "not entirely unlike" S, initially developed by Ross Ihaka and Robert Gentleman and now maintained by an informal, international team of volunteers known as the R Project (www.r-project.org). While it does not have all the features of S or S-plus, such as Trellis graphics and pull-down menus, it has a solid base and a rapidly growing library of contributed routines. Venables and Ripley (2000) has more technical detail on programming in the various flavors of S and R.

R is closely aligned with a large effort known as the Omega Project (www.omegahat.org) which will eventually provide an open-source development framework for statistical applications, interfacing with statistical manipulation environments, database systems and graphical display technologies using CORBA. This system has been under development since July 1998 using Java in a web-based setting. We anticipate migrating to Omega in the future, which should be easy since the R group are heavily involved in this development.

3 Organism Information Structure

Biologists want intimate control of simulation details to incorporate realism. Typically this involves events, that is significant biological changes that can be marked and counted. The events themselves can often be laid out in tables, which we can then use as raw inputs to simulations. These tables not only describe the life events, but they also identify the organisms, units, and other features, allowing the simulation software itself to be free of specific reference to any one system. At present these files all reside in library(ewing) in the data folder.

3.1 Organism Features

The simulation code has no organism information per se. All organism information is designed by the scientist in tables as ordinary text files that can be incorporated into our internal structure, which is appropriately called `Organism`. This internal structure contains simulation features, schedules for future events, and process information for interactions among organisms. For our prototype simulation, the primary organisms are Red Scale, *Aphytis*, *Encarsia* and a greatly simplified orange tree as a substrate. Here, orange is actually a static substrate with certain properties that are important for the life histories of the other organisms. The "master" information on these organisms is in the `organism.features.txt` file:

```
          units offspring attack   substrate deplete subclass  parasite  move
redscale  DD    10        NA       orange    100     host      NA        crawler
aphytis   hr    redscale  redscale orange    24      adult     ecto      adult
encarsia  hr    redscale  redscale orange    12      adult     endo      adult
orange    NA    NA        NA       NA        NA      NA        NA        NA
```

Notice that the orange "organism" has missing values (`NA`) for the features since it is considered static. In addition, redscale (Californai Red Scale) has no attack feature, as it is only a host for aphytis (*Aphytis* sp.), *Encarsia* (*Encarsia* sp.) and other parasitoids. Some features such as offspring may have either a number or the name another organism. The number signifies a mean value, while the name of an organism signifies that there is some interaction for this feature. For instance, Red Scale have on average 10 offspring, but the number of *Aphytis* offspring depends on the condition of its Red Scale host. Units of "biological time" may differ among organisms; Red Scale is temperature sensitive and grows by degree-days (DD), while *Aphytis* is diurnal, dependent on hours (hr). Other columns have to do with other features of the simulation:

units	biological time units (DD=degree-days, hr=hours)
offspring	mean number of offspring, or name of host
attack	name of host to be attacked
substrate	substrate on which organism lives
deplete	time in units to energy depletion worth 1 offspring
subclass	age class of life stages to be plotted by substrate
parasite	type of parasite or parasitoid
move	life stage that can move

3.2 Organism Future Events

An organism may have many potential future events, many of which correspond to stages of its life history. Other events, such as sex determination, feeding, starvation and death can be included as well. For instance, here are the future event schedules and other information by life stages for Red Scale, in object `future.redscale.txt`:

```
        current       future fid DD pch     color ageclass  event init
1       crawler first.instar   2 55   0     brown  crawler future   10
2  first.instar   first.molt   3 92   1     green     host future    4
3    first.molt     second.1   4 48   1     green     host future    4
4      second.1     second.2   5 30   2 turquoise     host future    1
```

```
5      second.2     second.3   6 30   2 turquoise     host future    1
6      second.3       female   8 52   2 turquoise     host future    1
7      second.3         male   9 52   2 turquoise     host future    1
8        female  second.molt  10  0   2 turquoise     host future    0
9          male        death  16  0   2 turquoise     host  death    0
10  second.molt      third.1  11 48   2 turquoise     host future    1
11     third.1       third.2  12 55   3      blue     host future    1
12     third.2       third.3  13 55   3      blue     host future    1
13     third.3        virgin  14 95   3      blue     host future    1
14      virgin        gravid  15 90   V    violet     host future    1
15      gravid        gravid  15 10   G     black   gravid  birth    0
16       death         death  16  0   D       red       NA  death    0
17     starved         death  16  0   D       red   gravid future    0
```

The times here are the mean number of degree-days (DD) to the event. The time to a scheduled future event, say from third.1 to third.2 would be a random draw T from the future event distribution, in this case $F(t) = 1-\exp(-M(t/55))$. The mean value function $M(t)$ is assumed to be the identity unless it is otherwise provided (see the next section). The future events for *Aphytis*, `future.aphytis.txt`, are similar with some notable exceptions:

```
    current   future fid  hr pch     color ageclass   event init
1       egg   larvae   2  48   E     brown    young  future   10
2    larvae prepupae   3  60   L     green    young  future    5
3  prepupae    pupae   4  24   p turquoise    young  future    3
4     pupae    adult   5 144   P      blue    young  future    2
5     adult     feed   7  12   F    orange    adult  future    1
6     adult     ovip   8  12   H    purple    adult  future    0
7      feed    adult   5  12   F    orange    adult  attack    0
8      ovip    adult   5  12   H    purple    adult  attack    0
9     death    death   9   0   D       red       NA   death    0
10     male    death   9   0   D       red    adult   death    0
11  starved    death   9   0   D       red    adult   death    0
```

Here the adult *Aphytis* may either feed or oviposit (ovip) on its host. Future events 5 and 6 are competing risks, which in this case are sorted out in the simulation code based on the size of the selected host. [This is one part that is still particular to this host-parasitoid system.] The event column identifies the type of event; either future event, birth or death (both immediate events); or attack (another immediate event that involves interaction with other individuals). The "fid" column simplifies some coding when relating current to future stages. The last column specifies the initial relative abundance of life stages: both species are weighted heavily toward young organisms.

The other columns for both Red Scale and *Aphytis* are used for plotting. The pch and color are used when plotting the spatial arrangement (temporarily disconnected), while the ageclass specifies how the life events are to be summarized. Recall that Red Scale "host" and *Aphytis* "adult" ageclasses are to be plotted by substrate according to the organism feature file.

3.3 Interactions among Organisms

The third type of spreadsheet concerns interactions among organisms. While this could be encoded directly into subroutines, such a table can provide more generality to the simulation modeling system. Here is `redscale.aphytis.txt`, including information about *Aphytis* can use various stages of Red Scale:

	ovip	feed	offspring	male
crawler	0	0	0.0	NA
first.instar	0	5	0.0	5
first.molt	0	2	0.0	5
second.1	0	5	0.0	5
second.2	3	2	0.0	5
second.3	4	1	0.0	5
second.molt	2	0	0.0	4
third.1	3	0	1.3	2
third.2	5	0	1.3	1
third.3	5	0	1.3	1
virgin	2	0	2.7	0
gravid	0	0	0.0	NA
death	0	0	0.0	NA
parasite	0	0	0.0	NA
male	0	0	0.0	NA
female	0	0	0.0	NA

Most columns have a 0-5 scale which serve as individual host weights during sampling. That is, if there are 10 hosts in the area, *Aphytis* will chose one based on the weights associated with their life stage. Thus, Red Scale prefers to oviposit on second and third instars, and is more likely to produce male offspring for smaller Red Scale. It will feed if the Red Scale is very small. The mean number of *Aphytis* eggs that an emerging *Aphytis* would have ready to lay is dependent on Red Scale stage as well.

The following type of interaction, `orange.aphytis.txt`, indicates how a species interacts with its substrate. In this case, *Aphytis* can move around the orange. Our prototype orange consists of a two-sided leaf, a 20-sided orange (icosohedron), and a twig connecting the two. Again, a 0-5 scale is employed. This file really only has information about movement among fruit-twig-leaf, while a second file, not shown, has topological information about movement within each of these (e.g. movement on icosohedron is restricted to the three adjacent triangles).

	substrate	side	find	move	fruit	twig	leaf	init
fr1	fruit	1	5	1	3	2	1	1
fr2	fruit	2	5	1	3	2	1	1
fr3	fruit	3	5	1	3	2	1	1
fr4	fruit	4	5	1	3	2	1	1
fr5	fruit	5	5	1	3	2	1	1
fr6	fruit	6	5	1	3	2	1	1
fr7	fruit	7	5	1	3	2	1	1
fr8	fruit	8	5	1	3	2	1	1
fr9	fruit	9	5	1	3	2	1	1
fr10	fruit	10	5	1	3	2	1	1
fr11	fruit	11	5	1	3	2	1	1
fr12	fruit	12	5	1	3	2	1	1
fr13	fruit	13	5	1	3	2	1	1
fr14	fruit	14	5	1	3	2	1	1
fr15	fruit	15	5	1	3	2	1	1
fr16	fruit	16	5	1	3	2	1	1
fr17	fruit	17	5	1	3	2	1	1
fr18	fruit	18	5	1	3	2	1	1
fr19	fruit	19	5	1	3	2	1	1
fr20	fruit	20	5	1	3	2	1	1
twig	twig	NA	1	5	3	1	2	1
lftop	leaf	top	3	3	3	2	1	5
lfbot	leaf	bottom	2	4	3	1	2	0

Notice that *Aphytis* is more likely to `find` fruit than leaf top, and twig has the lowest probability. Also, *Aphytis* tends to move short distances if on a fruit, and long distances if on a twig. Perhaps we need only rely on probabilities to transfer between orange components. A similar file exists for `orange.redscale.txt`, with perhaps slightly different values.

We are not completely settled on this system, but it provides a convenient starting point for spatial simulation. Fine detail movement is based on triangular grid system and is introduced in Section 5.

4 Future Event Scheduling

Once the various organism files discussed in the previous section have been constructed and placed in the appropriate location, the simulation is ready to be initialized. The simulation is initialized with the command:

- `init.simulation ( myrun, c("redscale","aphytis") )`

In this example, this command reads in the appropriate information for both Red Scale and *Aphytis*, and any dependent organisms identified in the `organism.features.txt` file (in this case, orange), and it prompts the user for the number of organisms, and initializes the populations. It also initializes other simulation features, such as the temperature regime (see Section 5). Once the simulation is initialized, the simulation is executed by using the command

- `step.future(myrun,1000)`

This command executes 1000 future events of reproduction, death, parasitization and aging through the various life stages. The summary plot changes with each 50 events. In this de novo use, the time to next event is drawn from an exponential with the mean time to event given by the "`future.redscale.txt` " file as explained in the previous section. During a simulation, as currently implemented, there are two graphical time summaries for each species. These show number of organisms by age class (left) and number of organisms by spatial location (right). For instance, Red Scale is counted by crawler, host, and gravid (host is all stages that can be parasitized by *Aphytis*), while *Aphytis* is counted by young and adult. The three spatial locations are leaf, twig and fruit.

<<Figure 1 here>>

Summary tables are provided at the end of the run indicating timing results and, if desired, age distributions before, during and after the simulation.

Changing the number of either species, or the relative timing of future events, can lead to an exponential growth or decay in the Red Scale population. The simulation displayed in Figure 1 is far from equilibrium. Because the system is inherently nonlinear, it is extremely hard to predict behavior. It is challenging to adjust the number of each species, time to events, movement and number of offspring to balance the population.

4.1 Competing Risk Structure

The simulation sorts all events such that the next future event has minimal time. This sort technique typically only involves rearranging one or two events, and is very fast. Essentially we are reordering a priority queue in such a way that the top of queue contains the next event to be processed (see Section 6.2 for details). That event is processed by setting the current event to the future event, processing any immediate events (e.g. scheduling birth of crawlers for gravid females), and then proceeding to the next event. Death is treated in this way as an immediate event. At the time of death, the individual is removed from the simulation. Competing risks arise when there are more than one possible future events following a given current event.

The competing risk structure has been described in a companion paper (Ewing et al. 2001) and is only briefly referred to here. The important steps in the current implementation are

(1) efficient computation of $M(\cdot)$ and $M^{-1}(\cdot)$ using forward and backward splines
(2) static lists of individual characteristics that do not change over the span
(3) interaction among species

In a future development of the simulation, we plan to incorporate more realistic settings and feedbacks. Currently the competing risk structure for *Aphytis* allows for a choice between feeding and ovipositing, as shown in the future event structure of the previous section. *Aphytis* proceeds from egg to larvae to prepupae to pupae to adult with mean future event times in DD as indicated earlier. However, in the adult stage, *Aphytis* may either feed or oviposit. After each feed or oviposit, the adult has a number of choices. Feeding sustains the insect, allowing it to potentially oviposit more offspring at a later time, while ovipositing depletes resources, but contributes to survival of the species. The choice *Aphytis* makes depends on a number of environmental factors including the number of eggs it has in reserve, the last time ovipositing took place, the availability of hosts, and the condition of the *Aphytis* in question. The decision to oviposit is also determined by the size of the scale and the number of *Aphytis* eggs previously oviposited. Many but not all of these features have been implemented.

4.2 User Friendly Graphical Design of Biological Processes

The major theme of this work in quantitative population ethology is the development of a modeling system that captures the essential features observed by field ecologists within a sound mathematical framework, presented with intuitive graphical tools. To accomplish this, it is essential to have graphical design of the mean value functions $M(\cdot)$ that are used throughout the modeling process to schedule future events. At present, we use the R implementation (due to Bates and Venables) of forward and backward cubic splines in a graphical setting where the user can select knots to create the shape needed.

One of the beauties of quantitative population ethology is the ability for a scientist to design mean value curves for each future event based on data and experience. In order to do this, we have developed an interactive system to craft such curves, which are then made available to the simulation. The underpinning of this system is a set of forward and backward interpolating

splines as implemented in R by Bates and Venables in `library(splines)`. The back-spline is not actually a spline, but it functions much the same. The routines are simple to use:

- `library(splines)`
- `meanvalue <- interpSpline( data$x, data$y )`
- `invmvalue <- backSpline( meanvalue )`

Here the data are columns "`x`" and "`y`" of the data frame "`data`". These resultant structures include the spline knots and coefficients, and are readily understood by R for plotting. One could for instance design the mean value curve for the future event of "virgin" by invoking the `ewing` library routine:

- `init.meanvalue( redscale, "virgin" )`

This opens a graphics window with a side bar. The default is a constant rate of 1, leading to a linear mean value curve with slope 1. In this example, the mean value is plotted from 1 to 5 time units, as the probability for an event beyond 5 units with this rate is less than .01. Notice that "replace" and "mean value" are both active, as indicated by their green color. That is, the shape of the curve can be modified using the cursor and clicking to move one or more nodes.

<<Figure 2 here>>

Figure 2(a) has three nodes moved upward to yield a slight curve. The blue line is the forward spline, while the red line is the backward spline. Notice that in this case they do not quite agree at the upper end. Further, if the forward spline is not monotonic, the backspline will not work and is not plotted. There is a certain art in designing cubic spline curves, especially when using back-splines. Here it might be useful to add a point at the upper end to bring the blue and red lines together.

The "rescale" button allows you to enter new limits for "time" and "mean value" (you have to switch over to the command screen). The "shrink to 1" button is important. This command attempts to shrink the curve toward a mean of 1 event over $[0,\infty)$. This standardization of the mean value curve is needed for the future event simulations. Note that the curve is extrapolated linearly past the upper time limit.

It is also possible to work on the curve as a cumulative probability by clicking the word "probability" (Figure 2(b)). The vertical axes switch sides, but you can still move nodes around. Here the math is operating on the mean value curve, although you are viewing the cumulative probability. Here the discrepancy at the upper tail between the forward and backward spline is not noticeable. Hence, one might choose to ignore it for design purposes.

The rate, or derivative of the mean value function, can be plotted (Figure 3), but one cannot redesign this curve at present. The rate is a quadratic spline, an exact derivative of the mean value function. Similar comments apply to the density, and to the derivative of probability.

<<Figure 3 here>>

The Kolmogorov-Smirov statistic can provide error bounds for data driven cumulative probability and cumulative hazard curves. For instance, one can put curves above and below for realizations that almost always deviate from the true (10% in blue), exceed half the time (50% in

green), and almost never are crossed (90% in red). The width of these bands depends on the inverse of square root of sample size. This could be done as bands around an empirical curve or as bands around estimated curves. For instance, one can notice that several different curves would fit any particular set of data. This is implemented with a new "data" button, but is not shown here.

The advantage of such a system is that one can in some situations begin with data and then adjust the curve based on other information. If there are many data, then the confidence limits would be tight and the curve well determined from the start. However, the data may be obtained from an indirect measure of the process, or may be rather imprecise. Small data sets would have wide confidence limits, highlighting how little one really knows about their shape. One might then feel justified in playing with the shape, which would allow an investigation of sensitivity.

5 Span and Resolution: Space and Time

5.1 Span and Resolution for Model

Orange fruit are mature for about 8 months. Luck (see Yu and Luck 1988 and references in Ewing et al. 2001) suggests that the time in days from egg to gravid adult for the insect species involved are roughly 2-4 weeks (red scale 24-25 days; *Aphytis* 13-14 days; *Encarsia* 19-28 days). Thus, 8 months corresponds to 10 or more generations for each species. This gives us our span of roughly 240 days or 6400 degree-days (DD), using 1 day = approximately 26.7 DD on average.

Resolution is more difficult to determine. Luck (cf. Yu and Luck 1988) often recorded red scale life histories to the precision of 1-3 DD, with a recognition of inherent variability. Information on *Aphytis* and *Encarsia* is often recorded to the precision of 1 day. However, multiple events of feeding and ovipositing probably occur on any given day for adults. For simplicity, we consider the resolutions of approximately 1 DD and 1 hour. These are roughly equivalent on average.

Given a particular span and resolution, what are the implications for the size of the simulation? Consider roughly 500 individuals at any one time, with on average 20 events per individual. We are running 10 generations, which makes 100,000 events. Version 3 of the software can handle about 3000 events per hour on a Pentium II 150Mhz system. Thus one simulation could be completed in a day or two. There are some issues about space management that need to be redesigned to make this happen, however, but they are solvable.

Suppose one were to use classical modeling techniques, running each individual a DD at a time. Assuming there are 500 individuals, there would be 3,200,000 steps (500 individuals times 6400 DD), or if you consider cycling through say 10 types of events per individual, 32,000,000 operations. This is a considerably more costly and time consuming operation.

Notice also that we do not explicitly need to set things up rounded to the smallest resolution. In fact, it can be useful to have a smaller precision to resolve ties. The important thing is to have a resolution that is relevant to the events under study, and vice versa. Note further that the choice of future events implicitly determines the resolution; coarser events correspond to coarser time resolution.

5.2 Quadratic Spline via Ramped Steps to Relate Hours to Degree-Days

How do we relate time to degree-days? Computationally, trigonometric functions are expensive and rather crude. In this simulation we model temperature (degrees) as a ramped step function. Below is a sketch where the nighttime temp falls below the threshold of biological activity, but exceeds it as the temperature increases during the daylight hours. The beauty of this technique is three-fold. First, it is simple to construct. Second, we can use the same spline functions that are already in place for designing probability to event curves. Notice that the degree-day curve is the integral of the temperature, which in this case is a quadratic spline. Finally, we can replace artificial constructs such as the one above with real data by producing a spline fit to the actual data.

<<Figure 4 here>>

The temperature/degree-day translation is incorporated into the simulation now. However, the tools to design them are still primitive. The routine initTemp() creates an arbitrary scenario that can be edited by hand for he changes over a day (Figure 4). The routine spline.temp() is a graphical tool much like the mean-value curve that allows one to modify the daily high and low temperatures (Figure 5). Below are the commands

- `initTemp()` `# set up Figure 4`
- `spline.temp()` `# modify Figure 5`
- `temp.plot()` `# show Figure 6`
- `temp.plot(derivative = TRUE)`

<<Figure 5 here>>

In the current simulation, degree-days are the units ultimately used to calibrate the development of the various individuals within the simulation. However, parasitism is inherently a diurnal process. It appears that neither *Aphytis* nor *Encarsia* are active at night. There is evidence that in the summer season Red Scale may be able to pass though its various developmental stages during a particularly warm summer evening. In such warm circumstances, parasitism may not occur. Figure 6 shows cumulative degree-days incorporating the daily and high/low temperature curves together.

<<Figure 6 here>>

5.3 Dodecahedron of Triangular Faces to Approximate Spherical Orange

First let us acknowledge that we don't really know how these insects move around the surface of an orange. We believe that they may move to someplace "close" or someplace "far away", but we may have little data to develop this. For example, Red Scale moves longer distances if it is on a twig substrate, but settles rather rapidly on the fruit substrate, and *Aphytis*, is known to be a reasonably poor flier, hence one might expect its search techniques to be compact, walking quickly from one Red Scale to another. It appears unlikely for *Aphytis* to routinely fly between distance branches. Since there is a risk of building artificial relationships

by pushing too hard on a fine mechanistic model of movement, we are keeping this simple for now. However, we have implemented a finer scale triangular coordinate system that can be used for more precise movements (Section 6).

We have postponed development of the fine scale triangular grid described below for now. Consider instead a 20-sided polygonoid, an icosohedron with triangular faces. Red Scale, for instance, might reside on one of these 20 faces. In fact, there may be a modest population of red scale on a single face. We keep track of which face the insect resides on, but ignore fine detail of exact position. We could incorporate some density-dependent feedback, say on the chance that a crawler successfully settles on a given face. Migration would be between faces on an orange, and on or off the orange via the green twig.

6 Triangular Spatial Coordinate System

For any ecological system in which one desires to simulate the dynamics of individuals, one must be able to calculate the distance between individuals and features of their environment, including other individuals. Since the distances and directions between objects must be computed a large number of times, an efficient computing algorithm is a necessity. We have simplified that on a coarse scale above using triangular faces on an orange. When it is necessary to locate organisms more finely, we propose a triangular coordinate system that is accurate to about 6% while providing fast computation.

Given the Cartesian coordinates $p=(x_p,y_p)$ of objects in two-dimensional space, the standard method for distance between two points p and q is simply to take the square root of the sum of the squares $d=[(x_p-x_q)^2+(y_py_q)^2]^{1/2}$. The angular position, relative to the origin is calculated as $\theta=\tan^{-1}[(y_p-y_q)/(x_p-x_q)]$. The inverse calculation relative to the origin is $(x,y) = (d\cos(\theta), d\sin(\theta))$. Computationally, the arithmetic operations used are extensive. A triangular coordinate system with a hexagon overlay has been developed to reduce the cost with a modest loss of precision. The properties of the triangular coordinate system are as follows (Figure 8):

i. 3 axes, 120 degrees to one another.
ii. The address (triplet) of any point on the grid sums to zero. Therefore only two of the three coordinates is stored as the third can be calculated by subtraction.
iii. Any point can be considered the origin of the coordinate system -- this implies the geometry of distances and direction relative to any given point.
iv. The origin of the triangular coordinate system can be translated to any arbitrary point by simple subtraction.

To calculate the distance between any two points p and q with coordinates (a_p, b_p, c_p) and (a_q, b_q, c_q), one simply computes $d = \max\{|a_p-a_q|, |b_p-b_q|, |c_p-c_q|\}$. Using this technique one need only perform three subtractions, three absolute values and a maximization to obtain the distance. This calculation is accurate to within 7.5% (see Figure 8). This appears to be an acceptable error when compared to other uncertainties in both the input data and to the assumptions made in the simulation.

To calculate the direction from p to q one constructs a hexagon centered on p in the triangular coordinate system. Then by a series of comparisons of the relative values of a_p-a_q, b_p-b_q, c_p-c_a and their absolute values one can determine direction down to one of the 12 half-sectors. This is accurate to within 15 degrees, or 4.2% (Figure 7).

Figure 7: Cumulative Degree Days and Simulating random dispersion from some origin is straightforward. Begin by selecting a distance d from the origin according to some distribution. Then select a sign (- or +) and major axis (a, b or c) and set that axis, say c, to ±d. Then select a second axis, say b, and sample uniformly between 0 and -c. The third axis, a, is computed as the difference a = -(b+c). Thus one needs to only compute one distance d and pick one of the 12 half-sectors (±a, ±b, or ±c for primary axis, plus the secondary axis). With Cartesian coordinates, one would choose a distance d and a direction θ, and then convert back to (x,y) using multiplication and trigonometric functions sin(·) and cos(·).

The triangular coordinate method, though not as precise as the standard method, is significantly more efficient. The conversion from the Cartesian coordinate system to the triangular coordinate system is also straightforward. To convert from the Cartesian coordinate system to the triangular coordinate system one need only to solve the following equations:

$$a = \left(\frac{2+\sqrt{3}}{4}\right)x - \left(\frac{2+\sqrt{3}}{12}\right)y, \quad b = -\left(\frac{2+\sqrt{3}}{4}\right)x - \left(\frac{2+\sqrt{3}}{12}\right)y$$

and c=-(a+b), where (a,b,c) are the coordinate triplets in the triangular coordinate system and (x,y) are the coordinate pairs in the Cartesian system. To convert from the triangular coordinate system one need only to solve the following equations:

$$x = \left(\frac{2}{2+\sqrt{3}}\right)(a-b) \qquad y = -\left(\frac{6}{2+\sqrt{3}}\right)(a+b)$$

Once a hexagon structure of some suitable size is selected, it becomes possible to overly the whole study area with a honeycomb of hexagons that are related to the underlying triangular coordinate system. The location of any adjacent hexagon can be found by simple arithmetic (Figure 1) providing direct access to all trees in that hexagon through a linked data structure.

The appropriate computer algorithms for the triangular coordinate system with hexagon overlay have been written and tested using R.

<<Figure 7 here>>

6.4 Sierpinski Search Algorithm

One of the more fascinating attributes of this modeling scheme is the ability to simulate the activities of individuals localized in both time and space. Of those activities none is more intriguing than the search techniques of an individual parasite as it tries to locate a viable host while attempting to avoid predators that are likewise looking for it. *Aphytis* is not a particularly adept flier and hunts only during daylight hours. At night, *Aphytis* tends to stay near the interior of the citrus tree in an apparent attempt to avoid predators such as spiders. During the day it will search individual twigs, leaves and fruit for available Red Scale, concentrating mainly on fruit. Other parasitoids such as *Encarsia* seem to display a preference for hosts residing on twigs and leaves (Yu and Luck 1988). One of the key questions concerning both predation and parasitism is how does a particular parasite optimize a search strategy? Clearly random searching may not be the optimal way for a given individual to find an available host. In the case of the Red Scale/*Aphytis* complex, it appears that the substrate may emit a chemical that attracts parasites to a particular area of active Red Scale.

In the current simulation, we have been using a simple random movement and search algorithms with mixed success. It may be advantageous to use a modified Sierpinski gasket

algorithm to determine the migration patterns of individual Red Scale and search patterns of *Aphytis*.

We are developing a search algorithm based on the Sierpinski gasket (see Wirth 1968), a recursive curve that exhibit a fractal quality. In the simulation we define a local area to be searched and select the order of the Sierpinski search (Figure 9) based on the number of hosts in the area. Essentially, the order determines the amount of time that a given individual spends in a given local area, which translates into the intensity that a given area is searched. As the order of search algorithm is increases the resolution of the basic search pattern increases. By using an explicit order the organism's search is guaranteed to terminate, at which point the organism must select another behavior, such as leaving the area.

<<Figure 8 here>>

7 Future Considerations

At present, we have developed a protype simulation which relies on implementations implicit in R of the items listed below. This works in practice for hundreds or thousands of individuals over thousands of future events. However, in order to simulate systems with millions of individuals and events, it will be necessary to carefully consider how to optimize speed and use of dynamic storage. Some of the concerns that must be address in a more comprehensive simulation are:

(1) pseudo-random number generation
(2) methods to organize future event scheduling (priority queues and leftist trees)
(3) dynamic storage allocation to handle the changing population
(4) garbage collection
(5) Hilbert and Sierpinski Curves

There are, however, reasons to postpone (1) and (3). The newest versions of S (version 4) and R (version 1.2) have very efficient memory management, and may obviate the need for implementation of our own DSA scheme as outlined below. In the next few section we shall discuss a bit of the "philosophy" inherent in the design of a simulation based on the dynamics of individuals with the population.

7.1 Deepening the Simulation

Bland Ewing continues to have ideas about how to deepen the realism of such simulations. He offered another level of environmental "indirection" which might be useful later. Consider that each of the 20 faces on an orange might be classified in terms of its micro-environment, say shady (1), partial shade (2) or sunny (3). These might affect how insects behave on the face, etc.

Since we are considering a fruit as a 20-element structure, and similarly a leaf as a 2-element structure (top and bottom, ignoring actual spatial location), we have in some senses a compartment model, with 20 fruit faces, one twig and 2 leaf sides connected in a graph. It is not much of a effort to add more leaves down the twig, making a more realistic branch. However, we do not really need to use the physical geometry of the branch explicitly. Instead, we have

probabilities of moving among objects (fruit face, leaf top, etc.). Extending this, there is a small probability that a crawler could eventually move, perhaps over several generations, to another fruit. It is much more probable that *Aphytis* move between various substrates.

The possibilities are endless. There is the very real possibility that a model of an entire tree could be performed on today's multiprocessor computer system. Such a simulation might be extended to a simulation of an entire grove. However, we should step back and consider the span and resolution. It is highly unlikely that there will be much movement between one tree and another over a month for a Red Scale crawler.

On another front, the triangular system can be extended to arbitrary three-dimensional surfaces. The icosohedron being used to simulate an orange can be tiled with triangular systems on each surface that perfectly match with adjacent triangles. Further, additional cylinders each of 10 triangles can be inserted in the middle of the "orange" to turn it into a "banana". Modern image processing uses triangular meshes to capture arbitrarily complex objects, and there is no reason that our system could not be similarly extended.

7.2 Competing Risk Structure Implementation

In most biological systems, one is generally impressed with the rich structure even the simplest of biological organism displays, and when one studies interrelationships in small groups of complex individuals one perceives a complex mosaic of their possible interactions. It has also become apparent that the advancement in computer technology has been very beneficial in aiding the tasks faced by most biologists. What may not be as apparent is that computers, particularly computer languages and operating systems also possess an intricate structure that can, in some cases be extremely efficient in solving certain types of problems, and totally inappropriate in other applications. In constructing a computer simulation of a problem in population biology, there are significant advantages to be gained if a given computer system, including its implemented languages, can efficiently handle the dynamics of a structured biological system.

In this section, we explore the general structure of a biological system in the context of a structure common to nearly all computers. However, no attempt will be made to describe any one computer or system of computers. The discussion of the modeling technique in terms of a particular computer language, in this case, R was described above.

For any biological system, there are myriad different biological structures that can be analyzed in many different ways. There is a temporal structure of interacting individuals and there is a also a spatial distribution of that individuals may change as a function of time. In addition there is the inherent structure of the organism itself, and there is a hierarchical structure in which that organism is imbedded.

In the computer implementation envisioned here, each individual organism is represented by a block of computer storage called a cell. This cell can be further subdivided into three partitions: unique attributes such as sex, age, location, and health; a membership method to relate the organism to other members of the population; an event structure with the events that can occur to that organism (Figure 9).

<<Figure 9 here>>

Given this basic structure, we can now consider a structure that is comprised of a number of individuals as shown in Figure 10. We consider the group called species "A' to be a list of individuals that comprise that species. There may be empty cells in the list. Further, each cell has pointers that multiply connected to other cells within the list, and possibly to other species. The property of being multiply connected defines a ring structure.

<<**Figure 10 here**>>

It is now possible to connect various species or list by defining a pointer structure F, that locates the first individual of that species, and a second pointer structure that locates the first empty location in each species table or list, where the empty locations are crosshatched (Figure 11).

<<Figure 11 here>>

This ring structure can further be decomposed into multiple substructures. There is the possibility of connecting individuals with common characteristics. In the case of the Red Scale/*Aphytis* simulation one might wish to group Red Scale on a twig as one subgroup and Red Scale on the fruit as a separate subgroup. For larger simulations, one might wish to group branches that are comprised of stems, twigs, fruit and leaves as a subgroup. We shall define a set as the links that join individuals in terms of common, and we shall refer to a group as a specialized set that defines any sub-population. It is clear that we can combine the ring structure into extremely complicated family of structures such that it is possible to mimic the structure of a population that has been resolved to each individual in the system (Figure 12).

<<Figure 12 here>>

7.3 Competing Risk Structure and Priority Queues

In the absence of mathematical or statistical indicators suggesting a preferred method for defining the competing risk structure, any method is acceptable. However, the choice of a particular method can determine the simulations ability to run efficiently. The competing risk structure is complicated hierarchical structure containing of all of the events in the simulation. Those events can be placed into three broad categories: (1) system events, (2) species-specific events (birth, death, etc), and (3) interaction events (parasite/host or predator/prey). At the system level there are essentially four types of events: (a) start/stop events, (b) edit events, (c) graphics and statistics gathering events, and (d) a priority queue.

Knuth (1968) defines a simple queue as having a "first in-first out" behavior in the sense that every deletion from the lists removes the oldest item. The competing risk structure described above has a "smallest in – first out" behavior in which every deletion removes the item with the smallest *key* from the list. (The key, in our case time, governs the sort of the queue.) Such a list is defined to be a *priority queue*. In the context of our simulation, the priority queue contains all of the events for each individual in the list.

Crane (1971) represents priority queues as linked binary trees. His method requires two linked fields and a small count field in every record, and has the following advantage over other methods:

i. When the priority queue is treated as a stack (Knuth 1968) the insertion and deletion operations are very efficient, taking a fixed amount of time independent of queue size.
ii. The records never move; only the pointers change.
iii. Two disjoint priority queues having a total of n elements can be merged into a single priority queue in only $O(\log n)$ steps.

Crane's original example is a special kind of binary tree called a leftist tree structure (see Knuth (1971) for properties and appropriate search and sort algorithms). The algorithm to merge two or more ordered leftist trees can be used to insert or delete a single element from a leftist tree. For the simulation discussed here, the leftist tree structure is dynamic, with each element containing the time of the next future event and required pointers. The occurrence of an event for an individual leads to scheduling its next future event, changing its position on the priority queue through a combination of deletion and re-insertion. This event may cause changes for other individuals, resulting in other deletions or insertions as well. Clearly this dynamic priority queue changes in both size and content as a function of the sequence of events.

As a prototype, computer codes were written to sort both leftist trees and doubly linked rings to determine their ability to perform a set of desired tasks and their efficiency. Figure 13 shows the time required to search both a doubly linked ring and a leftist tree that contains n items. When n is small, a ring search is more efficient. However, with more than 70 elements in the priority queue, a leftist tree search becomes much more efficient. Figure 5 suggests that the efficiency of the simulation is inversely proportional to the number of events in the priority queue. Because of this relationship, it appears that some sort of efficient method of allocating and de-allocating storage as events occur is extremely important. In the next section we discuss some of the properties of the dynamic storage allocation (DSA) routine.

<<Figure 13 here>>

7.4 Dynamic Storage Allocation

Dynamic storage allocation is defined as the allocation and de-allocation of blocks of storage in which the block of storage may vary in size. Various DSA techniques are describe in Knuth (1968). We focus on the requirements that influence the design of DSA for simulations using a priority queue such as that discussed above. Ewing, et al, (2001) discussed a typical problem in which two hundred individuals with ten attributes were followed for five generations. The dimensionality foe such a simulation is on the order of 10^4. If each individual is resolve into ten states then the state space for the simulation is on the order of $10^{10,000}$. Such a problem is intractable unless there is some way to reduce the size of the state space. It is critical that the only those events in the active event list or priority queue actually occupy storage. For this class of simulations to be effective the DSA must be able to efficiently allocate storage for those tasks entering the event queue and de-allocate storage for those event deleted from the event queue.

Unfortunately most traditional DSA techniques are designed to allocate or de-allocate large blocks rather infrequently. That is, they are not designed to operate in the tightest loops of

the simulation. Such DSA routines cannot search large numbers of small elements of storage to obtain the required blocks of storage to process. Our simulation requires precisely those features that traditional DSA techniques are not designed to handle. Specifically our simulations requires a DSA procedure that can:

i. allocate and de-allocate storage of a highly dynamic system;
ii. coalesce unused storage as soon as possible;
iii. operate at the innermost simulation loops and allocate very small amounts of storage (on the order of five words); and
iv. minimize search time for available storage.

The process of joining adjacent blocks of storage can result in a high percentage of fragmentation. There will be a large number of small blocks of storage, as opposed to a few large blocks, and an efficient method for minimizing search time is highly desired. With these requirements in mind, a dynamic storage allocation procedure was developed in which a rather complex pointer routine was devised to minimize search times. The procedures that were developed were:

1. CREATE(ADDRESS,SIZE): Given a request for a block of storage of length SIZE, return the location of available storage in ADDRESS, if available. This requires the following procedures:
    a. LEAVE(ADDRESS,SIZE): Remove a block of storage of length SIZE from the head of the junk ring, returning the location of the storage in ADDRESS.
    b. APPORTION(ADDFREE,ADDLOC,REMAINS): Split a free block designated by ADDFREE into a free block of length REMAINS and a allocated block. The location of the allocated block is stored in ADDLOC, and the size of the allocated block was determined by the external variable SIZE.
    c. JOIN(ADDRESS,SIZE): Add a small block of length SIZE located at ADDRESS to the tail of the junk ring
2. DESTROY(ADDRESS): De-allocate the block of storage located at ADDRESS. The size of the block is determined by its last use. This requires additional procedures:
    a. ENTER(ADDRESS): Insert a large block of freed storage at ADDRESS into an appropriate place in the junk ring.
    b. DELETE(ADDRESS): Remove a free block located at ADDRESS from its junk ring.

In addition to the above procedures, we need two computer specific procedures generically called ACQUIRE and RELEASE. ACQUIRE(SIZE) can acquire additional storage from the operating system-controlled DSA system. This routine is only used when there is no storage available in the user-controlled stack. RELEASE(ADDRESS) returns free storage to the system DSA controlled stack. Everything “above” ADDRESS is released and is no longer available to the simulation without a call to ACQUIRE.

In the current simulation using R, we have not made an effort to optimize the dynamic storage allocation of priority queue using the procedures describe above. In order to effectively implement these procedures, we may need to rewrite the R procedure is amore flexible language

such as C++. However, it may be that improvements in R with version 1.2 will significantly improve memory management such that DSA considerations can be postponed.

7.5 Multiprocessor Implementation on the HPCF

With the advent of sophisticated multiprocessor systems, such as the 32 Silicon Graphics (SGI) ONYX, the possibility of simulating complex interconnected process become possible. For example it should be feasible to simulate thirty-two branches of a citrus tree using each processor to simulate a complex branch containing fruit, twigs, and leaves at some level of resolution. Each processor simulates a branch under possibly different environmental conditions. Since Red Scale are relatively sedentary, each branch can be treated as being quasi-independent. *Aphytis* and *Encarsia* appear to move randomly from branch to branch searching for hosts. However, those events are rare, as both parasitoids have a tendency to remain in the local area.

In the context of a multiprocessor system such as the Onyx System a branch would be handled by a single processor called a node, and information concerning the relationship between various nodes is handled using the Inter-process Communication Protocol (IPC). The IPC allows processes to coordinate the use of both shared data objects and processors. In our example, a single processor is capable of updating the node stored in memory without interfering with its neighboring processors, unless directed to do so. Alternatively, a process is capable of making data available to its neighbor process. The network is capable of either operating synchronously or asynchronously and processors may be dynamically added to or deleted from the simulation. In our application any one processor has direct communication with only it's nearest neighbors. Communication between any two connected processors is accomplished by creating a segment of storage that is mapped into the address space for the neighboring processors. Such connections establish *signals* which allow notification of a software and/or hardware event asynchronously, and *semaphores* that coordinate access to various resources.

<<Figure 14 here>>

The IPC functions require the use of a shared arena, which is the segment of memory that is mapped into the address space of multiple processes. The shared arena is identified with a file that acts as backup storage for arena memory and communicating processes gain access to the arena by specifying its filename. This implementation makes it relatively easy for "unrelated" processes to communicate. The data stored in any one of the node processors is specific to the branch being simulated. All other information resides in the fusion processor. On the SGI ONYX, we shall use thirty-one node processors to update thirty-one branches in shared memory. Since each multi-processor can operate asynchronously, we shall assign one additional node the task of monitoring progress of all the other nodes. This processor performs bookkeeping and graphics function in addition to scheduling the inter-processor events. In addition to the inter-processor events, this processor will also monitor any global "events" and calculate the change in global parameters such as diurnal and degree-day events. Those events may be calculated using various "continuous time simulation and then sent to a particular processor or groups of processors as modified piecewise continuous functions. Essentially this processor acts a "fusion processor" to monitor the entire simulation, and monitor the system level events.

The modeling technique described here is inherently nonlinear, and the simulation attempts to model processes that are far from equilibrium. One very important study involves

placing identical versions of the simulation on each of the processors and then running each processor independently of its neighbors. However, each processor would use a different starting seed for the random number generator. Essentially, the HPCF would be used to perform a large number of independent runs in order to explore the properties of this highly non-linear simulation. Such Monte Carlo studies require a significant amount of computer time if performed sequentially, however such problems should be extremely efficient on the HPCF. In order to use the HPCF effectively, all graphical routines would be performed after the simulations were completed.

We have learned that the R system has been successfully installed on an SGI ONYX machine. Thus, while efficiencies can be gained by converting R code to a multiprocessor compatible version of C++, this may not be crucial at the present time. We are currently developing an effective and flexible method for parallelizing the program for use on the HPCF.

8 Acknowledgements

The authors would like to thank Dr. David L. Wood for his continual and unwavering encouragement over the long haul. We would like to thank David Baasch and Dr. William Waters for the contributions and encouragement they have provided in this effort. The authors would like to thank Drs. C. Huffaker, R. Smith, and J. Knox for their support in the development of this modeling technique..

This work was performed in part under the auspices of the U.S. Atomic Energy Commission and supported in part by the National Science Foundation through a grant (NSF GB-34718) to the University of California; and by the Environmental Protection Agency through an interagency agreement (EPA-IAG-COV-4) with the Lawrence Livermore Laboratory. One of the authors (B.E.) was supported by the U. S. Forest Service in the initial phase of this effect. The findings, opinions and recommendations expressed herein are those of the author(s) and not necessarily those of the University of California, the National Science Foundation, the Environmental Protection Agency, the U. S. Forest Service, the Atomic Energy Commission or the Lawrence Livermore Laboratory. This paper contains part of the intended Doctoral Dissertation of one of the authors (B.E.).

References

Chiang, CL (1968) *Introduction to Stochastic Processes in Biostatistics*, Wiley: New York.

Chiang, CL (1972) "On constructing current life tables," *J. Amer. Statist. Assoc. 67*, 538-541.

Clifford, P (1977) "Nonidentifiability in stochastic models of illness and death," *Proc. Natl. Acad. Sci. USA 74*, 1338-1340.

Crane, C.A. (1971)

David, H.A. (1974) "Parametric approaches to the theory of competing risks," *Reliability and Biometry*, SIAM: Philadelphia, 275-290.

Ewing, B., P. Rauch, and J. Barbieri (1974) "Simulating the dynamics and structure of populations," Lawrence Livermore Laboratory Report, UCRL-76046 (Rev. 1).

Ewing, B., B.S. Yandell and J.F. Barbieri and R.F Luck (2001) “Quantitative Population Ethology,” TR 1032, Department of Statistics, University of Wisconsin-Madison.

Fix, E, and J Neyman (1951) "A simple stochastic model of recover, relapse, death and loss of patients," *Hum Biol 23*, 205-241.

Forester, L.D., R Luck, and EE Grafton-Cardwell (1998) " Life stages of California red scale and Its Parasitoids", Univ. of Calif. Div. of Agriculture and Natural Resources, Pub 21529.

Keyfitz, N (1966) "A life table that agrees with the data," *J. Amer. Statist. Assoc. 61*, 305-312.

Keyfitz, N (1968) "A life table that agrees with the data: II," *J. Amer. Statist. Assoc. 63*, 1253-1268.

Knuth, D. (1968) Searching and Sorting Algorithms

Knuth , D. (19xx) Fundamental Algorithms

Tsiatis, A (1975) "A nonidentifiability aspect of the problem of competing risks," *Proc Natl Acad Sci USA 72*, 20-22.

Wirth, N (19xx) Data Structures, Algorithms and Programs, Addison Wesley Chapter 3, pg 130-137

Venables, W.N., and Ripley, R.D., 1999. *Modern Applied Statistics with S-PLUS*, 3rd ed., Springer-Verlag, New York, NY.

Venables, W.N., and Ripley, R.D., 2000. *S Programming*, Springer-Verlag, New York, NY.

Yandell, BS (1982) "Nonidentifiability of lethality in the survival experiment with serial sacrifice," *Math Biosci 62*, 1-6.

Yu, D.S., and Luck, R.F., 1988. Temperature dependent size and development of California red scale (Homoptera: Diaspididae) and its effect on host availability for the ectoparasitoid, *Aphytis* melinus DeBach (Hymenoptera: Aphelinidae). Envir. Entom. 17: 154-161.