MANUAL FOR *PARETOEVOLVE*,
EVOLUTIONARY COMPUTATION SOFTWARE FOR MULTI-OBJECTIVE OPTIMIZATION

Joel H. Reynolds[1], Marianne C. Turley[2], Rie Komuro[3], Maureen C. Kennedy[4]

[1] U.S. Fish & Wildlife Service, Refuges, Anchorage, AK, Joel_Reynolds@fws.gov
[2]U. S. Forest Service, Portland, OR
[3]Bioengineering Institute, University of Auckland, Auckland, New Zealand
[4] Quantitative Ecology & Resource Management, University of Washington, Seattle, WA

Version 1.0, DRAFT

Suggested Reference:
Reynolds, J. H., M. C. Turley, R. Komuro, M. C. Kennedy. 2008. Manual for *ParetoEvolve*, evolutionary computation software for multiobjective optimization. Version 1.0, 6 Oct 2008. http://faculty.washington.edu/joel/Software.hmtl,

The latest version of this document can be found at:
http://faculty.washington.edu/joel/Software.hmtl

CONTENTS

INTRODUCTION

This document details the steps required to estimate the Pareto frontier of a process-based simulation model using the optimization program *ParetoEvolve*. The Pareto frontier summarizes the optimal tradeoffs in simultaneous performance across multiple criteria (Figure 1). It is the central information summary used in model assessment and the Pareto Optimal Model Assessment Cycle (see Reynolds and Ford 1999). The Pareto frontier is usually estimated more efficiently using optimization methods than Monte Carlo simulation.

The user must supply the simulation model as a stand alone executable program. *ParetoEvolve* requires that this user-supplied code read input files, run model simulations, and write output files as described below (see

PREPARING THE Model).

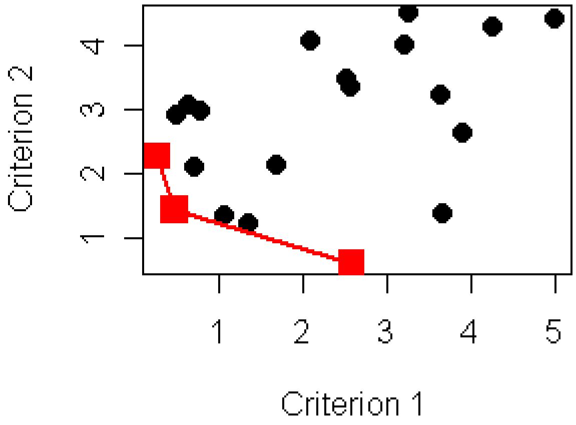

Figure 1. Feasible space (black circles and red squares) and Pareto frontier (red squares) for a two criteria minimization problem (e.g., minimize both criteria). The feasible space is that part of the objective space (Criterion 1, Criterion 2) that can result from a parameterization of the underlying model. The Pareto frontier is the tradeoff surface summarizing how 'improvement', minimization in this case, of one criterion is counterbalanced by worsening performance in the other.

Joel_Reynolds@fws.gov

WHAT IS *PARETOEVOLVE*?

ParetoEvolve is an evolutionary computation optimization program. It was specifically developed for estimating the Pareto frontier of process-based simulation models, but can be used for any multi-objective optimization problem. As far as *ParetoEvolve* is concerned, your simulation model is just another optimization problem. Even so, *ParetoEvolve* was written with certain algorithmic features that specifically support model assessment, such as its handling of binary error measures (see below) and memory structure based on *Pareto Groups* (Reynolds and Ford 1999, Komuro et al. 2007) (Figure 2).

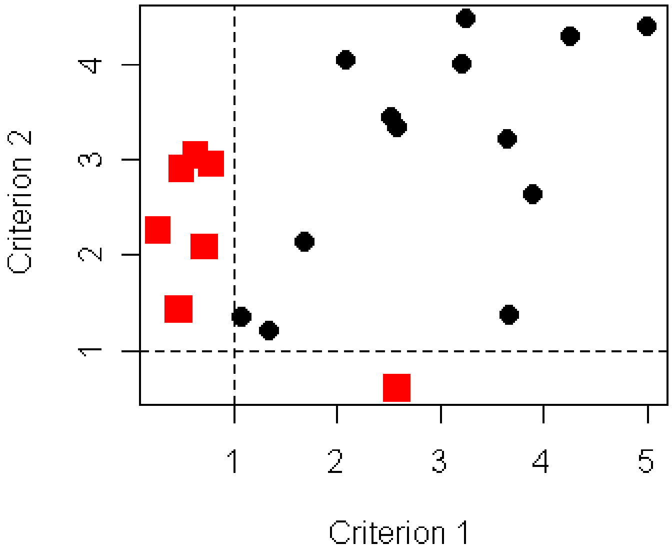

Figure 2. Often there is insufficient data to justify a precise target value for judging the model's ability to reproduce a specific characteristic. In this case one can define an interval of acceptable outcomes, e.g., "Criterion 1 predictions < 1.0 are acceptable", creating a binary error measure. Binary error measures partition the Pareto frontier into *Pareto Groups* – parameterizations that, while resulting in distinct outcomes, lead to the same assessment with regards to the binary error measures. If the error intervals for each criterion are '< 1 is acceptable' (dashed lines), then the red squares form the Pareto frontier and can be partitioned into two Pareto Groups – one with six parameterizations that adequately simulate Criterion 1 but fail on Criterion 2 and one with one parameterization that simulates Criterion 2 but fails on Criterion 1.

Evolutionary computation optimization uses the concepts of natural selection and genetic diversity to evolve solutions to optimization problems (Michalewicz 1996, Deb 2001). Such algorithms use a population-based search in contrast to traditional (analytical) mathematical optimization methods that calculate the objective value associated with a single parameterization then seek a next trial parameterization value with better performance (Figures 3, 4). Each stage ('generation') of a population-based search method evaluates a set of parameterizations, producing a set of objective values. A fitness is assigned to each parameterization, and the fitness values are used to probabilistically selected ‘parent’ parameterizations to ‘breed’ and create ‘offspring’ forming the next generation of parameterizations (Figure 4). This process continues, ultimately evolving optimal solutions.

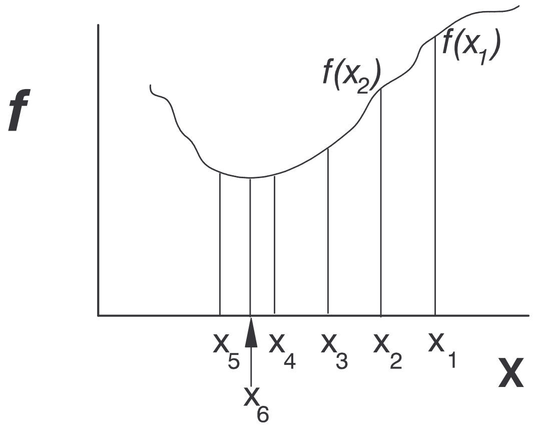

Figure 3. Traditional mathematical optimization methods, e.g. Newton-Raphson iteration, evaluate a single parameterization (x_i) at a time, each step seeking a new parameterization that further improves (here, minimizes) the objective function, f(x). A plausible sequence of trial values is illustrated: $x_i, x_2, \ldots, x_6$. In contrast, population-based optimization evaluates a set of parameterization each step, then grades their performance for deciding on how to best select the next set of parameterizations.

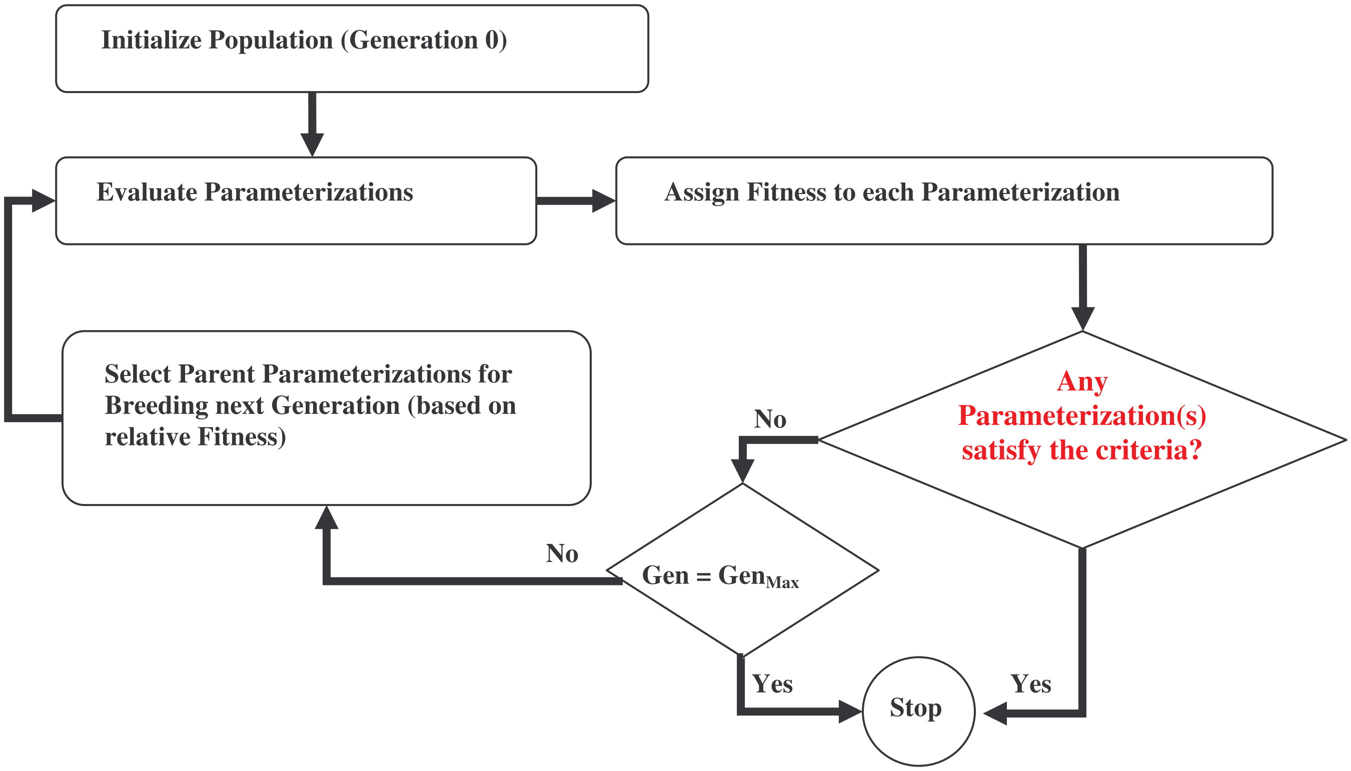
Initialize Population (Generation 0)
Evaluate Parameterizations
Assign Fitness to each Parameterization
Select Parent Parameterizations for Breeding next Generation (based on relative Fitness)
Any Parameterization(s) satisfy the criteria?
No
No
Gen = GenMax
Yes
Stop
Yes

Figure 4. Simplified flowchart of a population-based search algorithm. The key components are the definition of *fitness* for evaluating each parameterization, the definition of the *selection* process for picking parents, and the definition of the *breeding* process for generating a new population of parameterizations from the parents.

Numerous algorithms have been proposed in this large, active field of research (see Deb 2001 for an introduction). *ParetoEvolve* is based on a modification of the Non-dominated Sorted Genetic Algorithm (NSGA) (Srinivas and Deb 1994). The algorithm details are described in the code's internal documentation and are not presented here.

This manual does:
- outline the modifications required for both the user-supplied model code and *ParetoEvolve* to prepare each for use,
- briefly summarize the available run-time arguments for calling *ParetoEvolve*.

This manual does not:
- fully detail the optimization algorithm; please peruse the *ParetoEvolve* source code and its internal documentation or read Komuro et al. (2007)[1].
- fully explain evolutionary optimization algorithms. Please see, for example, Deb (2001) for a review of this burgeoning field.
- explain the Pareto Optimal Model Assessment Cycle; please see Reynolds and Ford (1999), Komuro et al. (2006), and Komuro et al. (2007).

PARETOEVOLVE SOURCE CODE

ParetoEvolve is written in the C programming language and consists of four source files and three header files. These need to be edited as described below and compiled to build the executable program. The original code was created in MS Visual C++ but does not specifically contain any function calls unique to that environment (as far as we know).

The *model executable* referred to below is the user-supplied executable code, which has to do much more than just simulate the process. It must be written so that when it is called it (i) reads in the input file of parameterizations created by *ParetoEvolve*, (ii) for each parameterization, runs the simulation model and calculates the resulting summary characteristics from the model output (the '*criteria vector*'), and (iii) writes the parameterizations and their criteria vectors to an output file. These requirements are detailed below.

---

[1] The algorithm presented in Komuro et al (2007) includes an elitist component that is not used in the version of *ParetoEvolve* described in this manual.

Joel_Reynolds@fws.gov

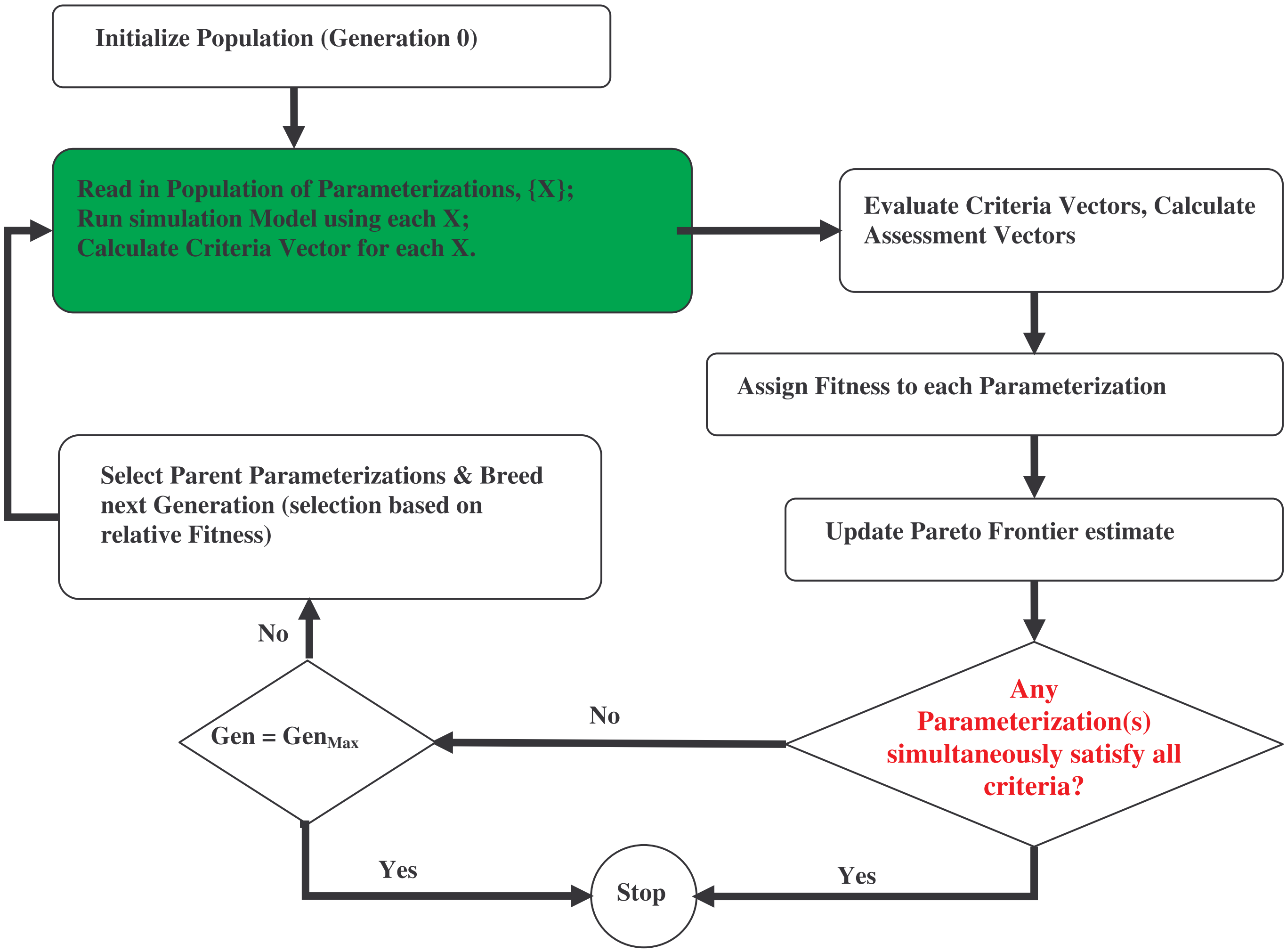
Initialize Population (Generation 0)
Read in Population of Parameterizations, {X}; Run simulation Model using each X; Calculate Criteria Vector for each X.
Evaluate Criteria Vectors, Calculate Assessment Vectors
Assign Fitness to each Parameterization
Update Pareto Frontier estimate
Any Parameterization(s) simultaneously satisfy all criteria?
No
Gen = Gen_Max
No
Select Parent Parameterizations & Breed next Generation (selection based on relative Fitness)
Yes
Stop
Yes

Figure 5. Simplified flowchart for the optimization program *ParetoEvolve*. *ParetoEvolve* generates each population of parameterizations then passes them as a set to the user-provided simulation model program (green box). The model code has to read in and run each parameterization, calculating its performance with regard to the selected characteristics (the *criteria vectors,* assuming there are multiple characteristics being assessed), and pass the information back to *ParetoEvolve*. The optimization program then evaluates the results and implements all the other steps. Information is passed between *ParetoEvolve* and the model code via writing and reading specially formatted text files.

BRIEF OVERVIEW OF *PARETOEVOLVE*'S SEARCH PROCESS

The multi-criteria optimization software *ParetoEvolve* estimates a model's Pareto Frontier (and associated Pareto Optimal Set) by an iterative process (Figure 5):

1. *ParetoEvolve* generates a population of feasible parameterizations, writes them to a text file (model 'input file'), the calls the model executable.
2. The model executable then reads in each parameterization from the text file, simulates the process with the parameterization, calculates the simulation's *summary*

characteristics and resulting *assessment criteria* values, then writes each parameterization and its associated assessment criteria to a second text file (model 'ouput file') (Reynolds and Ford 1999).

3. *ParetoEvolve* reads in the output file, evaluates each parameterization's criteria vector to form an *assessment vector*, calculates the Pareto Frontier and Pareto Optimal Set of the population of parameterizations and assessment vectors (Reynolds and Ford 1999, Komuro et al. 2007), and uses that information to breed a new population of parameterizations (the next generation of the evolutionary optimization search).
4. The process then repeats Step 1 and the cycle continues for another generation until either (i) a parameterization is found that simultaneously *satisfies* all the criteria or (ii) the user-defined maximum number of generations has passed. The Pareto frontier is estimated from all the evaluations and returned by the algorithm (the *Historic* Pareto frontier rather than just the last generation's frontier).

Thus before using *ParetoEvolve* you must first:

1. Prepare your model code to interact with *ParetoEvolve* and
2. Prepare *ParetoEvolve* to interact with your model code.

HOW COMPUTATIONALLY COMPLEX CAN THE MODEL BE?

Running *ParetoEvolve* generally entails 10,000s to 1,000,000s of model simulations, depending on the choices of population size and maximum generation number. The necessary minimum values for these two controls are model-dependent and hence unknown. From our experience we suggest population sizes of 100-200, depending on the number of unknown parameters, and a minimum of a couple thousand generations. These demands should be considered in the context of available computing resources to judge applicability of *ParetoEvolve* to a particular model. Note that the model simulation step is the bottleneck and that simulating a population of parameterizations is inherently parallelizable if you write the model executable to take advantage of multiple networked machines.

EXAMPLE MODEL

The steps required in preparing the model executable code and the *ParetoEvolve* code are illustrated using the example of a model of the hourly growth of the leading shoot of a Sitka spruce (*Picea sitchensis* (Bong.) Carr.) tree. The underlying motivation for the model and the results of its assessment and revision are detailed in Komuro et al. (2006, 2007a, and 2007b).

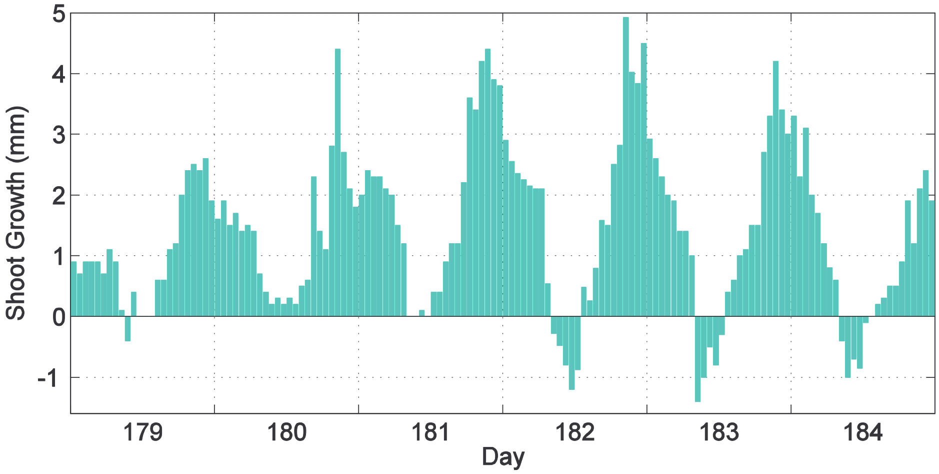

Figure 6. Observed hourly shoot growth (expansion and contraction) by Julian date (see Komuro et al. 2006).

Extension and contraction of a shoot were automatically measured every hour (Figure 6) using an electro-mechanical sensor (Milne et al. 1977). A model (Figure 7; see Komuro et al. 2006 for details) was proposed for the functional dependence of the observed expansion and contraction on current and recent solar radiation, temperature, and transpiration calculated using the Penman-Monteith equation (Monteith et al. 1965).

$$S_t = x_1 \cdot \underbrace{\left(\sum_{k=1}^{24} T_{t-k}\right)\Big/24}_{\text{Average temperature for the past 24h}} - x_2 \cdot \underbrace{\left(\sum_{k=25}^{48} T_{t-k}\right)\Big/24}_{\text{Average temperature for the past 48h–24h}} + x_3 \cdot \underbrace{\left(\sum_{k=25}^{48} R_{t-k}\right)}_{\text{Total radiation for the past 48h–24h}} + x_4 \cdot \underbrace{\left(\sum_{k=49}^{72} R_{t-k}\right)}_{\text{Total radiation for the past 72h–48h}} - x_5 \cdot \Delta_t D \cdot \underbrace{\left(\sum_{k=1}^{24} S_k^*\right)}_{\text{Total growth for the previous day}}$$

with $\boxed{\Delta_t D \equiv D_t - D_{t-1} = W_t - U_t = W_t - x_6 \cdot D_{t-1}}$

Figure 7. Hourly shoot growth, S_t, modeled as a function of current and recent hourly mean temperature, T_t, solar radiation, R_t, and change in water deficit, $\Delta_i D$ (Komuro et al. 2006). The coefficients x_i are the unknown model parameters. Water deficit is estimated as the difference between estimated uptake from the soil, U_i, and loss through evapo-transpiration, W_i.

The model was assessed with regard to twenty four characteristics capturing mean rates of growth during four time periods each day (Figure 8): early morning (hours 2--6), late morning

(8--12), afternoon (12--16), and late evening (20--24). These represent, respectively, the pre-dawn expansion period when water deficit is lowest; the late morning period of maximum contraction; the afternoon when recovery from contraction starts; and late evening when maximum expansion occurs. Comparisons were quantified as (Predicted mean growth - Observed mean growth) during each four hour period, thus assessing both timing and magnitude.

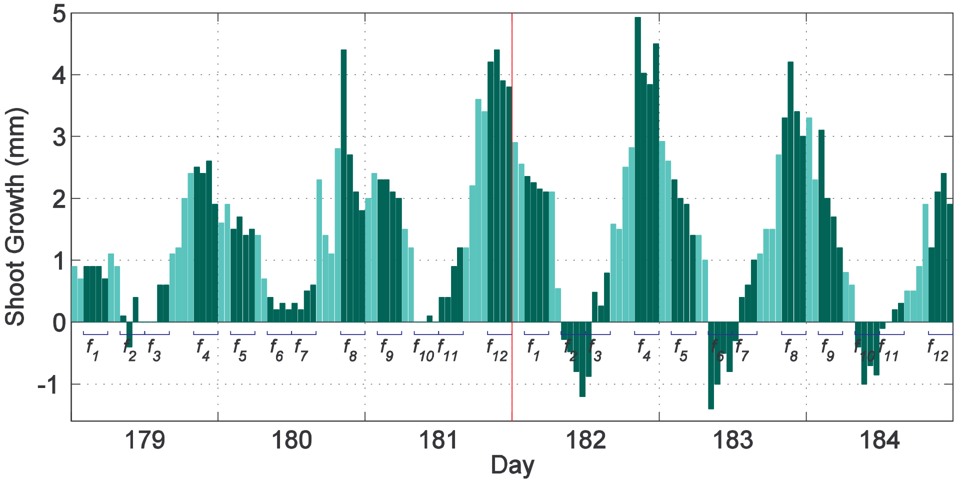

Figure 8.
Observed hourly growth data with assessment periods highlighted (dark green). The model was assessed with regard to mean growth rates in each of four time periods each day, see Komuro et al. (2006, 2007a, 2007b).

PREPARING THE MODEL EXECUTABLE

The model executable must be able to

(i) Read in a text file of parameterizations, one per row. The pathway and name of the text file are defined by the user in setting up the *ParetoEvolve* code (`MODEL_INPUT_FILE variable,` set in `UserSettings.h`), so must the number of parameterizations in each file (the ‘population size’; `POP_SIZE`, set in `UserSettings.h`) and the number (`NUMPARAMS`, set in `UserSettings.h`) and name (`pnames`, set in `UserArrays.h`) of each parameter. The model code must read in either a fixed number of rows (the population size) or, better, an unspecified number of rows - i.e., read in and simulate each row (parameterization) until the end of the file.

The parameterization text file will start with a row of parameter names then list the parameterizations as fields separated by a white space. The following example is from assessing the shoot growth model. The model has six parameters (`x1, x2, x3, x4, x5, x6`):

```
x1  x2  x3  x4  x5  x6
```

```
0.000000  0.001000  0.043000  0.047000  0.174000  0.371000
0.000000  0.019000  0.070000  0.036000  0.159000  0.539000
0.005000  0.004000  0.034000  0.051000  0.253000  0.787000
...
```

(ii) Run the simulation model with each parameterization (this may involve multiple simulations per parameterization if the model is stochastic).

(iii) Calculate the summary characteristics (objectives) that were chosen for assessing model performance and evaluate them with regard to the chosen criteria, creating the criteria vector (Reynolds and Ford 1999; Komuro et al. 2006).

For example, the executable for the shoot growth model runs the simulation with a given parameterization, uses the predicted hourly shoot growth to calculate the mean growth rate during each of the four periods for each day of the simulation (i.e., the summary characteristics), then combines this with the observed growth rates to calculate each periods' criterion value: Predicted mean growth - Observed mean growth.

(iv) Create an output file with one line per parameterization, listing the parameterization then the criteria vector. The pathway and name of the file are defined by the user in setting up the *ParetoEvolve* code (`CRIT_OUT_CALL`; `UserSettings.h`), as are the number (`NUMCRITERIA`; `UserSettings.h`) of objectives and their names (`cnames`; `UserArrays.h`).

The first row of the file should contain the parameter names and objective names. NAMES must not include blank spaces between characters. The row must report the parameter settings THEN the criteria values.

For example, the following output is from assessing the shoot growth model on just the days 179-181 (Komuro 2006). There are six parameters (`x1, x2, x3, x4, x5, x6`) and twelve criteria assessing the goodness of fit of the predicted growth time series (`d1, d2, d3, d4, d5, d6, d7, d8, d9, d10, d11, d12`). The simulation summaries from the first three parameterizations are shown.

```
x1  x2  x3  x4  x5  x6  d1  d2  d3  d4  d5  d6  d7  d8  d9  d10  d11  d12
```

```
0.000000 0.001000 0.043000 0.047000 0.174000 0.371000 -0.070371 0.017149 -
0.252480 0.139678 -0.075626 0.281741 0.097565 0.302846 -0.119141 0.269034 -
1.033098 0.233234
0.000000 0.019000 0.070000 0.036000 0.159000 0.539000 -0.004016 0.326310
0.249490 -0.197787 -0.289778 0.318527 0.355739 0.386018 -0.291224 0.654089 -
0.201839 -0.212053
0.005000 0.004000 0.034000 0.051000 0.253000 0.787000 -0.148511 0.002888 -
0.010412 0.053456 -0.331739 0.283286 0.356075 0.256863 -0.674833 0.152905 -
0.400107 -0.162460
…
```

After making sure the model executable satisfies these requirements, recompile it and record the drive location of the executable file.

PREPARING THE *PARETOEVOLVE* CODE

The user must modify the *ParetoEvolve* code to define

(i) the number of generations and population size to use in the search (in `UserSettings.h`);

```
      /****************** UserSettings.h ********/
…
      /*******************************************************************/
      /****** OPTIMIZATION CONTROL SETTINGS ******/
      /*******************************************************************/
          /***** SEARCH SIZE AND DURATION *****/
      #define POP_SIZE 100   /* Number of parameterizations per generation */
      #define GEN_NUM 1000   /* Number of Generations to evolve solution */ …
```

The user needs to experiment to decide how many generations are sufficient for convergence as it will be problem dependent. From our experience, we suggest at least 1000 or more generations and population sizes of at least 100.

Table 1. Values to be modified by the user in order to adapt *ParetoEvolve* for the user-defined program.

File	Name	Description
UserSettings.h	NUMPARAMS	number of parameters (decision variables)
	NUMCRITERIA	number of objectives (assessment criteria)
	GEN_NUM	number of generations to let the solution search evolve
	POP_SIZE	number of individual parameterizations in the population for each generation
	SYST_CALL	pathway and name of the user-supplied executable that calculates objective values for each parameterization (i.e., the model)
	MODEL_INPUT_CALL	pathway and name to use when creating the parameter input file
	CRIT_OUT_CALL	pathway and name of the model output file (created by the user-supplied model)
	CRIT_COPY_CALL	operating system command to save a copy of the current generation's criteria output file.
	FRONTIER_CALL	pathway and name of file for archiving the non-dominated Pareto frontier estimates. **WARNING**: the path must exist (i.e., folders and subfolders already created) in order for *ParetoEvolve* to write these output files.
	BINARY	flag for whether the error measures are binary (1) or continuous (0)
	GEN_TO_PRINT	frequency for printing current results to files
UserArrays.	pnames[NUMPARAMS][11]	array of parameter names
	cnames[NUMCRITERIA][11]	array of objective names
	psearch[NUMPARAMS][3]	search ranges for each parameter (min, max, step size)
	assessinfo[NUMCRITERIA][2]	target values for each objective—for continuous errors the first is the target value, the second is not utilized; for binary errors the first is the lower bound of the target range, the second is the upper bound of the target range

(ii) the pathway and filenames for where to locate the model executable (`SYST_CALL`), where to place the model input file (`MODEL_INPUT_CALL`), where to locate the model output file (`CRIT_OUT_CALL`), where to intermittently copy the model output file created during the search (`CRIT_COPY_CALL`), and the pathway to the directory used to store intermittent copies of the current Pareto frontier estimate during the search (**WARNING**: create the directory before running *ParetoEvolve*). The intermittent copies of the current population results and the current Pareto frontier estimate are written every X generation, where X is the value defined for the constant `GEN_TO_PRINT in the file`

`UserSettings.h`.

```
/*********  UserSettings.h *********************/
…
/******************************************************************/
/****** PATH AND FILENAMES for model executable and files ******/
/******************************************************************/
    /* model executable.*/
#define SYST_CALL "c:\\Tree\\criteria\\Debug\\criteria"
/* model input file (created by ParetoEvolve) EDIT PATH, NOT FILENAME!*/
#define MODEL_INPUT_CALL "c:\\Tree\\input.txt"
/* model output file (read by pareto_evolve) EDIT PATH, NOT FILENAME!*/
#define CRIT_OUT_CALL "c:\\Tree\\crit.out"

/******************************************************************/
/****** FILENAMES for recording intermediate status of search.******/
/******************************************************************/
/* System call to create a permanent copy of the current model output
file.  The root filename in the 'destination' path will be appended with
the current generation number.  E.g., 'crit.out10*/
#define CRIT_COPY_CALL "copy c:\\Tree\\crit.out  c:\\Tree\\crit"

/* Path and root filename for storing intermediate Pareto Frontier
Archives.  The root filename will be appended with the current
generation number. ('Historical Pareto Frontier' = cumulative elitist
archive of search.)
#######
WARNING:
The path must exist, ParetoEvolve cannot create it.
I.e., create the necessary folders and subfolders where you want
ParetoEvolve to put the output files before running the code.*/
#define FRONTIER_CALL "C:\\TREE\\ParetoSets\\Paretoset"
```

The model input and output locations must match those in the model code. Under the Windows operating system, the copy command is ‘copy’; under Unix/Linux it is ‘cp’ (e.g., in `CRIT_COPY_CALL`).

Joel_Reynolds@fws.gov

(iii) the parameter names (`pnames`), their feasible search ranges (`psearch`) and minimum search increment, and the criteria names (`cnames`) and target values or acceptable ranges (`assessinfo`) (in UserArrays.h):

```
/************** UserArrays.h ********************/
…
/* PARAMETER NAMES to be used in printing out search results */
   /* NAMES are LIMITED to 10 characters!! ADD ELEMENTS AS NEEDED.*/
char pnames[NUMPARAMS][11] = {"x1","x2","x3","x4","x5","x6"};

   /* CRITERIA NAMES to be used in printing out search results */
   /* NAMES are LIMITED to 10 characters!! ADD ELEMENTS AS NEEDED.*/
char cnames[NUMCRITERIA][11] = {"d1", "d2", "d3", "d4", "d5", "d6",
"d7", "d8", "d9", "d10","d11", "d12"};
```

```
/***** SEARCH RANGES AND SPECIFICS ******/
      /* This array should have a 3 component row-vector for each
parameter. The first element is the MINIMUM value of that parameter's
search range, the second component is the MAXIMUM value of the search
range, and the third component is the MINIMUM STEPSIZE for the parameter.
E.g., '{0.0, 0.4, 0.001}' implies feasible search values of 0.0, 0.001,
0.002, 0.003, ..., 0.399, 0.400; (400 possible search values for this
parameter).
The first row is associated with the first parameter in pnames[],
the second row with the second parameter, etc.
ADD ELEMENTS AS NEEDED.*/
double psearch[NUMPARAMS][3] = {
      /* {Min, Max, Min Step}, */
      {0.0, 0.4, 0.001},   /* search for param x1 */
      {0.0, 0.1, 0.001},   /* search for param x2 */
...};


double assessinfo[NUMCRITERIA][2] = {
   /* BINARY? {Min, Max}. CONTINUOUS? {Target, trash} */
      {-0.41,0.41}, {-0.41,0.41}, {-0.41,0.41}, {-0.41,0.41}, {-0.41,0.41},
      {-0.41,0.41}, {-0.41,0.41}, {-0.41,0.41}, {-0.41,0.41}, {-0.41,0.41},
      {-0.41,0.41}, {-0.41,0.41}
   };
...
```

The minimum and maximum value and minimum step size are used to define a grid of potential values for each parameterization. **The user must be sure that each increment is an integer step**. That is, `(max - min)/increment` must be an integer. <u>In this example, the parameter search range for x1 is (0, 0.4) with a minimum step size of 0.001, coded as `{0.0, 0.4, 0.001}`</u>.

If the search uses a continuous error measure, the user must specify the objective targets where the first number specifies that target and the second number is ignored. If the search is to use a binary error measure, the user must specify the objective target ranges where the first number specifies the minimum of a range and the second number specifies the maximum. The objectives are all evaluated using binary error measures (`BINARY = 1` below). The acceptable interval for the first criteria, `d1`, is [-0.41, 0.41]. This is an unusual example in that the criteria all use the same target range.

(iv) the number of parameters (`NUMPARMS`), the number of objectives (`NUMCRITERIA`), the type of discrepancy measure used to assess each criterion (binary or continuous) (`BINARY`), and how frequently to record the current search results (`GEN_TO_PRINT`) (in UserSettings.h).

```
…
#define NUMPARAMS 6   /* number of parameters in model being assessed */
#define NUMCRITERIA 12    /* number of criteria used in assessment */
…
#define BINARY 1   /* =1 if all discrepancy measures are binary,
                        =0 if all are continuous */
#define GEN_TO_PRINT 500 /* how often to print out intermediate Pareto
                  Frontier Archives ('Historic Pareto Frontiers'). */
#define GEN_TO_PRINT 25 /* how often to record current search results */
...
```

If the search uses *binary* error measures, a criterion value is assessed as to whether or not it falls within that user-specific acceptable range (step iii above). If so, it is assigned a value of 1, if not then it is assigned a value of 0. If the error measures are *continuous*, an individual's assessment is quantified by the coordinate distance between its criterion result and a criterion's target value (specified in step iii above).

(v) If necessary (e.g., long pathnames) revise the array dimensions for `critname` and `paretofilename` in ParetoEvolve.h and pareto_main.c

After making these changes to the *ParetoEvolve* code, the user must recompile all of the files (ParetoEvolve.h, UserSettings.h, UserArrays.h, pareto_main.c, pareto_update.c, pareto_breed.c, pareto_misc.c) to create the optimization search executable.

RUNNING THE *PARETOEVOLVE* CODE

Open up a DOS window (Windows OS) or terminal window (Unix, Linux), move to the directory containing the *ParetoEvolve.exe* file, and enter either:

- *ParetoEvolve.exe*
  to start the search at generation 0 and run through the maximum number of generations (`GEN_NUM`);
- *ParetoEvolve.exe* 5
  for example to start the search at generation 5 and run through the maximum number of generations (`GEN_NUM`). *ParetoEvolve.exe* will look for the `CRIT_OUT_CALL` file and read it in, assuming it is the output from the 5th generation's search.
- *ParetoEvolve.exe* 5 120
  for example to start the search at generation 5 and run through generation 120, stopping after generation 120 (which is assumed less than `GEN_NUM`). *ParetoEvolve.exe* will look for the `CRIT_OUT_CALL` file and read it in, assuming it is the output from the 5th generation's search. It will stop after having run the model simulations through the 120th generation of parameterizations.

The model's approximate Pareto Frontier and Pareto Optimal Set will be given in the final `FRONTIER_CALL file.`

UNDERSTANDING THE OUTPUT FROM *PARETOEVOLVE*

The output files saved by the `CRIT_COPY_CALL` command should be investigated for obvious signs of problems in the parameterization search, such as stagnation in the population members.

The refinement of the search's Pareto Frontier estimates can be investigated by exploring the Pareto Frontier files created in the folder set in the `PARETO_PATH` argument defined in step (ii) above. The files are named `ParetosetX`, where X is a whole multiple of the `GEN_TO_PRINT` value defined in step (iv) above.

Each ParetosetX file consists of variable length sections, one for each *Pareto Group* or set of parameterizations which produce the same assessment vector (Reynolds and Ford 1999). When continuous error measures are used, each parameterization is (generally) its own group; when binary error measures are used, groups may contain many parameterizations. This is discussed further in Reynolds and Ford (1999) and Reynolds and Golinelli (2004).

Each Group starts with the same marker row,

`-999999.999 Assessment Vector 999999.999,`

followed by a row listing the criteria names then a row listing the group's common *assessment vector* (evaluated criteria error measures) (Reynolds and Ford 1999). This is followed by another marker row (see below), a row of the parameter names and criteria names, then a list of each parameterization in the group, one per row, along with their original criteria vectors. For example, the following Pareto Frontier file contained a group that satisfied all but the criteria focused on each day's last time period (d4, d8, d12) using binary error measures as the assessment vector was
(1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 0) =
(True, True, True, False, True, True, True, False, True, True, True, False) =
(Satisfied, Satisfied, Satisfied, Unsatisfied,…).
The beginning of one of the Pareto Optimal parameterizations from that group is shown.

```
Pareto Frontier:
-999999.999 Assessment Vector 999999.999
   d1   d2   d3   d4   d5   d6
   d7   d8   d9   d10   d11   d12

   1.00   1.00   1.00   0.00   1.00   1.00
   1.00   0.00   1.00   1.00   1.00   0.00
=============== Simulations ===============
  x1  x2  x3  x4  x5  x6  d1  d2  d3  d4  d5   d6
   d7  d8  d9  d10  d11  d12
 0.35 0.090 …
```

The separators (‘`-999999.999 Assessment Vector 999999.999`‘) allow for automatic file parsing using grep or awk-type commands.

MISCELLANEOUS NOTES

ParetoEvolve assumes that all parameters are 'independent' and that the feasible search space is simply the Cartesian product of the individual parameter's search ranges. If this is not true, i.e., there are constraints or explicit relations among the feasible parameter values, then the model code must either:

- have it's input section written in terms of 'independent' parameters from which the other parameters are explicitly calculated (with the constraints being implicitly incorporated into the transfer between parameter spaces, e.g. *ParetoEvolve's* ⇔ the model's parameter space), or
- check the feasibility of each parameterization generated by *ParetoEvolve* and, when an infeasible parameterization is found, either convert it into a feasible parameterization (somehow) or set it's criteria vector to some absolute worse value so that, implicitly, the infeasible parameter space is abandoned by the evolutionary optimization.

The latter strategy should be avoided when possible so to minimize computer effort wasted on defining the feasible search space.

ParetoEvolve assumes all parameters are floats (or doubles). When this is not true, the model code should read in floats then recast individual parameters to other types as necessary.

It will be available via the MCmodeler interface (Steele-Feldman and Reynolds 2007) at http://faculty.washington.edu/joel/Software.html in late 2008. That version eliminates the need for user modification and recompiling of *ParetoEvolve,* but restricts uses to models with 20 or fewer parameters and 20 or fewer model assessment criteria or outputs, nor does it support the use of run-time arguments discussed above.

More information

Further documentation, updates and examples can be found at:
http://faculty.washington.edu/joel/index.html under Model Assessment or Software.

REFERENCES

Deb, K. 2001. *Multi-Objective Optimization using Evolutionary Algorithms.* John Wiley & Sons, Inc., New York, New York.

Komuro, R., Ford, E. D., and Reynolds, J. H. 2006. The use of multi-criteria assessment in developing a process model. *Ecological Modelling* **197** (3-4): 320-330.

Komuro, R., Reynolds, J. H., and Ford, E. D. 2007. Using multiobjective evolutionary algorithms to assess biological simulation models. In S. Obayashi et al. (eds): *EMO 2007, Lecture Notes in Computer Science vol 4403.* Springer-Verlag, Berlin, pp 560-574.

Michalewicz, Z. 1996. *Genetic Algorithms + Data Structures = Evolution Programs* (3rd ed.). Springer, New York, New York.

Reynolds, J. H. and Ford, E. D. 1999. Multi-criteria assessment of ecological process models. *Ecology* **80** (2): 538-553.

Reynolds, J. H. and Golinelli, D. 2004. Multi-criteria inference for process models: structural and parametric inference for a stochastic model of feline hematopoeisis. In *2004 Joint Statistical Meeting Proceedings*, American Statistical Association, pp 2978-2986.

Srinivas, N., and K. Deb. 1994. Multi-objective function optimization using non-dominated sorting genetic algorithms. *Evolutionary Computation Journal* **2** (3): 221-248.

Steele-Feldman, A., and J. H. Reynolds. 2007. MCmodeler: a model assessment interface. Available at http://faculty.washington.edu/joel/Software.html in late 2008.

Joel_Reynolds@fws.gov