Make Things Better

Evolution is cool. Mediocre things evolve into really fit things. Wouldn't it be nice if your programs could do that?

Well, of course they can. There are plenty of Genetic Algorithm (GA) implementations that provide a multitude of exciting interfaces for specifying your problem, the fitness metric, and various parameters of the search. Catch is, none of those interfaces is particularly convenient for generically improving arbitrary programs. That's why we built **gamake**.

How gamake Works

Despite the name, **gamake** actually has two separate search modes: exhaustive and GA. All it does is to try various combinations of parameter values by invoking standard **make** to build whatever and create a metric output, which it then parses to determine fitness.

Yeah, that could take a while, so **gamake** is actually an MPI program that runs the makes in parallel across however many nodes you told MPI to run on.

What gamake Needs To Know

The parameter names and value ranges to be searched are given in a file named **gain**. For each parameter, the following is specified:

Name:	The parameter name as it will be passed to **make**
Type:	Currently **int**, **float**, or **double**
Min:	The minimum value for this parameter
Max:	The maximum value for this parameter
Step:	The granularity of steps in the value

The choice between exhaustive and GA search, and some other runtime choices, are made on the **gamake** command line.

When **gamake** runs each parameter set, it does so by executing make in a directory on one of the nodes it was **mpirun** to. It is the responsibility of the **Makefile** to create a file called **eval** in response to being invoked with "**make eval**". This file is then parsed looking for a decimal ASCII floating-point value which is taken to be the metric value. Greater values are considered better, but using negative values can essentially invert that order.

An Example

Here's a trivial program for which **gamake** will find the optimal values of **A**, **B**, and **C**. The parameters used are:

```
A int 1 5 1
B int 40 50 1
C int 1 5 1
```

The **Makefile** is:

```
test:   test.c
     cc test.c -o test \
          -DA=$(A) -DB=$(B) -DC=$(C)
eval:   test
        cc test.c -o test \
          -DA=$(A) -DB=$(B) -DC=$(C)
        ./test >>eval
        rm -f test
```

Here's the C source code that computes **eval**. The reference values are used to compare with for the evaluation – **gamake** should find the reference values automatically:

```
#define AREF 5
#define BREF 42
#define CREF 3
double f(double x) {
  return(A + (B * x) +
         (C * (x * x)));
}
double ref(double x) {
  return(AREF + (BREF * x) +
         (CREF * (x * x)));
}
main() {
  double e = 0, d, x;
  for (x=0; x<=100; ++x) {
    d = ref(x) - f(x);
    d *= d;
    e += d;
  }
  printf("%lf\n", -e);
}
```

How Do I Get gamake?

The current version of **gamake** is fully functional, but we intend to delay release somewhat until we are certain that the interface will not change. At that time, it will be made freely available online as full source code at http://aggregate.org/GAMAKE/

This document should be cited as:

```
@techreport{sc13gamake,
author={Matt Wiggins and Henry Dietz},
title={Make Things Better},
institution={University of Kentucky},
address={http://aggregate.org/WHITE/sc13gamake.pdf},
month={Nov}, year={2013}}
```