OPTICOSP

An Optimization Program Running on a Single Processor

Steve Bracker
s_bracker@hotmail.com

University of Mississippi

March 28, 2003

Mucool Note #270

OPTICOSP is a single-processor reduction of the multi-processor optimization program OPTICOOL[1]. Like its predecessor, OPTICOSP runs a user-supplied apparatus simulation program repeatedly, searching for an apparatus configuration that is optimal, varying operating parameters and evaluating measures of merit as specified by the user. OPTICOSP is written in Fortran-77. There are versions that run on PCs under Linux or Windows; porting to other platforms should be straightforward. This note describes OPTICOSP and gives an example of how it has been used with the ICOOL[2] simulator.

Introduction

In the early stages of apparatus design, it is frequently necessary to write a simulation program to assess the performance characteristics that might be achieved, and how the performance might vary as various design parameters are changed. Typically the simulator reads one or more ***simulator input files*** specifying the design parameters, and writes one or more ***simulator output files*** reporting various measures of apparatus performance.

As the design process advances, the designer will often need to vary small sets of design parameters and attempt to find those specific values of the design parameters that yield optimal performance. Doing so is often slow and tedious. The simulations can be very time-consuming, and the designer finds himself called upon to start one simulation, wait an hour or so for it to finish, retrieve the output, compare the performance of this apparatus configuration with others already examined, decide upon a new apparatus configuration to be studied next, prepare the simulator input files, and start another simulator run. Often this must be repeated dozens of times.

OPTICOSP's task is to automate the process just described. OPTICOSP is told how to perform a simulation run, including any pre-processors and post-processors that must be run. The user provides an ***optimization control file*** that tells the optimizer what design parameters may be changed, and within what bounds. He provides ***template files*** for any simulator input files that OPTICOSP is expected to change before starting a simulation. In the optimization control file, he also specifies how to compute a single figure of merit, based on simulator or post-processor outputs, that will be used to decide whether the optimizer has approached the optimum configuration closely enough to quit, and if not, to guide the optimizer in choosing the next apparatus configuration to study. OPTICOSP writes a report file which provides a simulation-by-simulation summary of the optimization process, a table characterizing the optimum found, and various measures describing how sensitive the apparatus performance is to changes in design parameters in the vicinity of the optimum.

The optimization process may be summarized:

1-Initialize the optimizer and read the optimization control file
2-Perform a few initial simulations
3-Based on previous simulations, choose the next candidate configuration
4-Prepare simulator input files for the new candidate configuration
5-Simulate the candidate and gather the simulator output files
6-Based on the simulator outputs, assess the performance of the candidate
7-If finding the optimal configuration still requires work, back to 3
8-If the optimal configuration has been found, report it
9-Perform additional simulations near the optimum and report the results

Terminology and Basics

The design parameters that may be changed by OPTICOSP are named ***optimization variables***. The number of optimization variables is typically 1-4, although more are possible.

Any simulator input file that OPTICOSP may alter in preparation for a new simulation has an associated ***template file***. A template file is simply a copy of the simulator input file in which data fields that may be changed -- typically numeric values -- are replaced by ***substitution markers*** such as **{phase}** or **{magCurrent}**. OPTICOSP prepares new simulator input files by copying the associated template file and replacing the substitution markers by numeric values, so **{phase}** might be replaced by **30.00** and {**MagCurrent}** might be replaced by **1437.1** for some specific simulator run.

At the end of each simulation run, either the simulator or a post-processor must generate a ***merit output file***. The merit output file lists the names and values for performance measures -- the ***merit variables*** -- that may be used by OPTICOSP to compute an overall figure of merit for the candidate configuration. By convention, the name of the merit output file is ***for010.dat***.

The user must prepare an ***optimization control file*** named ***OptCont.dat***. (The best way to do this is to copy and alter an existing file, because there are very useful comments in the existing files, and the format conventions for the file are demonstrated.) The optimization control file specifies:

1. The name and speed of the ***processor*** being used (irrelevant, a vestige of OPTICOOL).

2. The name of each ***optimization variable***, the interval within which it is allowed to vary, etc.

3. The name of each ***substitution marker***, and how it is associated with an optimization variable.

4. The name of each ***merit variable*** used to compute the overall figure of merit for an apparatus configuration, how the ***partial merit*** for that variable is derived from the value of the merit variable, and how the partial merit is folded into the overall ***configuration merit***.

5. The name of each ***template file*** and the associated ***simulator input file*** that is prepared prior to each simulation, based on the template.

6. A few additional ***optimization parameters*** -- the number of particles to be propagated during each simulation, the "close enough" criterion used to stop the optimization, etc.

The user must also supply a ***simulation script*** -- a DOS batch file or a Linux shell script -- which will be invoked each time a simulation is to be performed. At the minimum, the script must set the current working directory to the place on disk where all the data files reside, and then invoke the simulator program. It may also clean up files left from previous simulator runs, run preprocessors such as XICOOL, and run post-processors. The simulation script is named ***simulate.bat***.

OPTICOSP produces two output files -- a report file named ***report*** which records everything having to do with the optimization that is normally of interest to the user, and a diagnostic file ***diagnose*** which contains some additional information occasionally of use to track down errors.

All of the data files listed above -- the optimization control file, the simulation script, the template files, any additional simulator input files which do not need templates -- are gathered together in one ***job subdirectory***. The intermediate and output files from the simulator, and the optimization output files, are delivered into the same subdirectory. OPTICOSP is executed with a command-line argument which gives the full pathname to the job subdirectory. If the executable image of OPTICOSP is stored in ***/home/zot/opticosp*** and the job subdirectory is ***/home/zot/opticosp/applications/rfofo*** then you should issue (linux) commands:

```
cd  /home/zot/opticosp/applications/rfofo
/home/zot/opticosp/opticosp  /home/zot/opticosp/applications/rfofo
```

The first line isn't really necessary, but it guarantees that when the optimization job is done you will be in the subdirectory holding the report files you want to see. In the second line, you state the path and name of the program followed by the pathname of the job subdirectory where the data files live. Of course, depending on what environment variables, aliases, symbolic links, etc. you have set up, the command line you need may be a lot shorter. I usually compose a batch file (shell script) named ***go*** which sets the default directory, does any file cleanup needed, and invokes the optimizer.

Setting Up an OPTICOSP Optimization?

1. Set up the simulator you will be using and understand its performance.

First, set up the simulator you use (I will assume it's ICOOL) for the task at hand. That will mean creating a simulation control file (for001.dat), perhaps a beam particle file (for003.dat) , perhaps one or many field-defining files (for0nn.dat), etc.

Explore the behavior of the simulation by running simulation jobs by hand. Choose the input operating parameters to be varied, and decide upon reasonable ranges for each; these are the ***optimization variables***. Decide which output quantities will enter into a final figure of merit; these are the ***merit variables***. Do not try to optimize too many optimization variables at once, or the optimization process will take forever.

Determine how many particles you must run through the simulator to achieve the precision in the merit variables you feel you require. If you tell the optimizer to determine the location of the optimum very precisely but run too few particles to characterize the apparatus performance to corresponding precision, the optimizer keep searching "forever".

Compose a script (command file) to perform one simulation. The script might delete old output files, run any preprocessors you use to help prepare simulator input files, run the simulator, and then run any postprocessors you use to examine simulator output and further characterize the apparatus performance.

Time spent at this stage is almost always time well spent. Do not try to set up an optimization until you have a good quantitative grasp of what it is reasonable to expect the optimizer to accomplish.

2. Prepare a program to output the merit variables after each simulation

Among the options: customize ICOOL, customize an existing postprocessor, or create your own postprocessor. The output must be a file named ***for010.dat***, and it must be a free-format text file having the following format:

```
meritVariableName1  meritVariableValue1
meritVariableName2  meritVariableValue2
               etc.
```

For example, suppose that you use two merit variables: muon survival fraction and transverse emittance. Assuming that you already have a program that generates these quantities, then you need something like:

```
      open (unit, name='for010.dat', status='unknown')
      write (unit,800) muonSurvivalFraction
 800  format ( 'muonSurvivalFraction', 2x, f10.6)
      write (unit,801) transverseEmittance
 801  format ( 'transverseEmittance', 2x, f12.8)
      close (unit)
```

Make sure that when you run your script, the for010.dat file is produced and the numbers are reasonable.

3. Prepare templates for the simulator input files that OPTICOSP needs to change

Often, the only file that needs to be changed is for001.dat, the simulation control file. For the moment, let's suppose that is the case. To make a template, simply copy the input file and give the copy a name like for001.tpl. For each optimization variable -- the variables that OPTICOSP will be authorized to change to achieve an optimal apparatus -- substitute for the actual numeric value a substitution marker that looks like {optVariableName}.

For example, suppose that you want to vary the second parameter of a reference particle definition:

```
refp
nnnn  0.1983  nnn  nnnnn  nnnn
```

In place of the numeric value, you might place the name of an optimization variable, for example PZ0REF:

```
refp
nnnn {pz0ref}  nnn  nnnnn  nnnn
```

You can call it by its ICOOL fortran variable name or whatever you like. You must include the curly brackets. {pz0ref} is termed a *substitution marker*; it indicates a place in the text of the input file that OPTICOSP may edit as it prepares the current version of for001.dat for the simulator to use.

If you want OPTICOSP to have control over the number of particles to be simulated, you can place the substitution marker {NPART} wherever the particle count is defined. If you want OPTICOSP to be able to control the starting random number seed, you can place the substitution marker {RNSEED} wherever the random number seed is defined.

Save your template file by a name similar to but different from for001.dat. Each time it needs a new simulation to be performed, OPTICOSP will read the template, fill in numeric values for the substitution variables, save the resulting file in for001.dat, and then run the simulator. ***Note carefully that any existing for001.dat is overwritten.***

You might want to make the following test at this point. Copy your template file to for001.dat. Edit for001.dat, substituting reasonable numbers for each substitution marker. Run your simulation script and verify that everything runs ok.

4. Set up a subdirectory for your job and move the files into it.

Now, at last, you are nearly ready to define your optimization task. To begin, set up a subdirectory to hold your optimization and simulation files. On linuxfarm1, this should be a subdirectory of /home/oc/opticosp/applications. Suppose that you define an application ***ringwraith*** for assessing the performance of some cooling ring design. All of your data files -- the ICOOL files, the OPTICOSP files and any pre/postprocessor data files -- should be in /home/oc/opticosp/applications/ringwraith. In addition to data files, you may put the source and executable for any pre/postprocessor that is customized for that specific application.

Among the files that will certainly be found in the ringwraith subdirectory at the beginning of an optimization run:

OptCont.dat -- the optimization control file
simulate.bat -- a script that is run every time OPTICOSP needs to simulate a new apparatus configuration
forxxx.tpl -- the template files for each simulator input file that may be changed by OPTICOSP.
forxxx.dat -- the simulator input files that will not be changed by OPTICOSP
go -- a script which starts the optimizer; it usually cleans up outputs left from previous runs and then invokes OPTICOSP.

As optimization progresses, additional simulator input files will be generated:

forxxx.dat -- the simulator input files produced by OPTICOSP based on the forxxx.tpl templates.

Among the files that will be produced in the ringwraith subdirectory in the course of an optimization run:

for002.dat -- the main report file generated by ICOOL (each simulation)
for009.dat -- the particle history file often produced by ICOOL and often read by postprocessors, merit file generators, etc. (each simulation)
for010.dat -- the merit file that must be produced by each simulation run and is read by OPTICOSP to help determine the next configuration to simulate. (each simulation)
report -- the optimization report file generated by OPTICOSP (one per optimization)
diagnose -- the optimization diagnostic file generated by OPTICOSP (one per optimization)

It is important to distinguish clearly between files that are generated once per simulation (ICOOL outputs, most postprocessor outputs, merit files) and those that are output only one per optimization. Upon completion of an optimization run, the once-per-simulation files are **<u>not</u>** (in general) from a simulation run at or near the optimum, but the **report** file generated by OPTICOSP will have some simulation-by-simulation summary data in it, as well as a rather complete characterization of the optimization process and results.

The simulate.bat file

Assuming that you are already familiar with ICOOL, you will already understand what most of the input files do. Of those that will be new, simulate.bat is the easiest to understand.

Suppose you were optimizing "by hand", running the simulator and its pre/postprocessors from the keyboard. You would probably get tired of typing commands to

```
start the preprocessor
start the simulator
run the first postprocessor
run the second postprocessor
```

each time you wanted to test a new apparatus configuration. You would compose a small script. For each new simulation you would edit the input files as needed and then type the name of the script, for example ***simulate.bat***.

OPTICOSP invoked the simulator again and again, and it too wants a script. That script must be called ***simulate.bat*** and must be located in the ***current working directory***. The current working directory should be your application directory; in this example, it's in /home/oc/opticosp/applications/ringwraith.

A typical simulate.bat, for a simulation that involves calling ICOOL and one custom postprocessor MYPOST, would consist of only three lines:

```
cd  /home/oc/opticosp/applications/ringwraith
/home/oc/icool252/icool
/home/oc/opticosp/applications/ringwraith/mypost
```

The first line is very important. The simulation is run in a different instance of the shell than the one in which the optimizer itself is running. It does not inherit process context like the current working directory from the optimizer's shell.

The second line invokes the simulator; you must be able to assume, at this point, that all of the simulator's input files have been prepared and are here in the current working directory defined in the line above. The version of ICOOL you are using is off in its own directory.

The third line invokes the postprocessor. This happens to be a custom postprocessor located in your own application area, but it could just as well be a standard postprocessor in the ICOOL area or elsewhere. The postprocessor will generally output the merit file for010.dat.

Hopefully it is clear that you could invoke preprocessors (e.g. XICOOL[3]) and additional postprocessors in this script as well. For example:

```
XICOOL to help prepare the input files
ICOOL to do the simulation
ECALC9F (standard postprocessor) to calculate emittances, etc.
MYPOST to read the ECALC9F output file and generate the merit file
```

The Optimization Control File OptCont.dat

Here is an example optimization control file:

```
! + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + +

WorkerNames

! For OPTICOSP, just one worker, and its name and speed don't matter.

  dellmonster    2000

! + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + +

OptimizationVariables

! Define the optimization variables. Each has:
!  --a name (often the same as the fortran source code name)
!  --a lower bound
!  --an upper bound
!  --a number specifying how steeply the merit curve falls beyond the bounds
!  --the number of summary configurations to be computed
!  --an optional comment at the end of the line

  !Aperture radius (meters)
  apertureRadius           0.015   0.050   3.  11  fairly loose bounds

  !X coordinate displacement (meters)
  coordinateDisplacement  -0.010  +0.010  15.  11  very strict bounds

! + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + +

InputSubstitutions

! Define the input file substitutions. Each has:
!  --a substitution marker e.g. {abc}
!  --an optimization variable name
!  --an output format specifier
!  --a multiplier M
!  --an addend A

! For every occurrence of the marker in a ghost input file, the value
! M * optValue + A is substituted, formatted as specified.

  {aper}   apertureRadius          f8.4   1.0  0.0
  {disp}   coordinateDisplacement  f8.4   1.0  0.0

! + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + +
MeritVariables

! Define the merit variables. Each has:
!  --a name (often the same as the fortran source code name)
!  --a merit combination code, M, A1, A2 ... A9
!  --merit variable value at normalized merit = 0.1
!  --merit variable value at normalized merit = 0.9
!  --minimum value of the merit function (usually 0.0)
!  --maximum value of the merit function (usually 1.0)

  !Transverse emittance
  ! Anything greater than 0.0002 is pretty bad
  ! Anything less than 0.0001 is just fine
  transverseEmittance  M  0.0002  0.0001  0.0  1.0

  !Fraction of particles that survive
  ! Anything less than 60% survival is pretty bad
  ! Anything greater than 90% survival is just fine
  survivalFraction     M  0.6000  0.9000  0.0  1.0


! + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + +

FileTransfers

! Define the files to be edited using substitution markers.
!
! The input template (first filename) is edited by substituting numeric
! values in place of substitution markers, and the results placed in the
! data file (second filename) to be read by the simulator (or a simulator
! preprocessor).

  ! Primary simulation control file for ICOOL
  for001.tpl for001.dat


! + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + +

Parameters

! Define various operating parameters controlling the optimization
! process. Each line consists of a name and a value. All parameters
! must be explicitly specified; there are no defaults.
!
! nPartSim -- number of particles to propagate each optimization
!  simulation
! nPartSum -- number of particles to propagate each summary
!  simulation
! stopCriterion -- how small a volume you wander within before you
!  stop simulating, in normalized coordinates
! maxOptSims -- maximum number of optimization simulations
! pauseEachSim -- 0 for free-running, 1 to pause after each simulation

  nPartSim  1000
  nPartSum  1000
  stopCriterion 0.001
  maxOptSims 50
  pauseEachSim 0
```

Commentary:

1. Blank lines are ignored.

2. Lines whose leftmost non-space character is ! are comment lines, and are ignored.

3. All other lines are interpreted as meaningful input data.

4. The file is divided into sections. Each section begins with a one-line section header, which must start in column 1 and contain the section name. Valid section names are:

WorkerNames
OptimizationVariables
InputSubstitutions
MeritVariables
FileTransfers
Parameters

5. Data lines which are not section headers must have at least one leading space.

If the comments and blank lines were stripped away in the example above, this would be a very tiny file. There are only 18 input data lines, including the 6 section headers. The comments may be very important to ***people*** reading the file, and ought not be omitted.

Section ***WorkerNames***

Since OPTICOSP is by definition single-processor, there can be only one worker name. It doesn't matter what that name is or what speed is associated with it. I usually put down the computer's network name and its speed in MHz. (Worker names are a vestige left over from OPTICOOL.)

Section ***OptimizationVariables***

Optimization variables are the variables that OPTICOSP is authorized to change in order to search for the nearly-optimal apparatus configuration Each must have a ***name***, a ***lower bound***, an ***upper bound***, and a ***measure of how strictly the bounds should be enforced***. If the optimizer tries to wander outside the specified bounds, it is not absolutely prohibited from doing so, but the merit of the resulting configuration is reduced. The fourth field (3 and 15 in the examples above) specifies how steeply the merit should fall for configurations where optimization variables are a bit outside the bounds. With low numbers like 1 or 2, the bounds enforcement is quite lax. With numbers as high as 15, the bounds are very strictly enforced. (See plots in the Appendix.)

The last field specifies the ***number of summary configurations*** to be computed after the optimum has been located. Summary configurations are defined in the following manner:

All the optimization variables are set to their value at the optimum. One optimization variable is chosen. Its value is progressively set to {lower-bound, lower-bound + delta, lower-bound + 2*delta . . . . upper-bound}, where delta is chosen to produce the specified number of summary configurations for that variable. For each point so defined, a simulation is performed and the results are output in the report file. This is done for every optimization variable in turn.

Computing summary configurations and carefully examining the results are crucial to ensuring that the optimum you have found is meaningful. Among the things you might discover: (1) the optimizer was fooled by a false optimum, (2) the optimum is located outside the optimization variable bounds you specified, (3) the optimum is extremely insensitive to one or more optimization variables, or (4) the merit-height of the optimum is not high compared to the simulation-to-simulation noise (e.g. not enough particles are being propagated).

Section ***InputSubstitutions***

Once OPTICISOP has decided where it wants the optimization variables set for the next simulation, how does it convey that information to the simulator? By editing the template files, which have substitution markers instead of numeric values for the optimization variables. This section establishes a correspondence between the substitution markers in the template files and the optimization variables. You can specify the format in which the optimization variable's value is to be written, and two constants, M and A. Each line in this section says "Every time you see {whatever} in an input template, take the value V of the associated optimization variable, generate M * V + A, and output that quantity, in the specified fortran format, in place of the substitution marker."

For simple cases, the substitution marker has the same name as the optimization variable (enclosed in curly brackets), M=1, A=0, and there is only one substitution marker associated with each optimization variable. The substitution marker is found at only one place in the input templates.

However, there are cases in which matters are more complicated. (1) A given substitution marker is found at several places in one or more template files. Then M * V + A is substituted at each place the marker is found. (2) There are two or more substitution markers based on a single optimization variable. For example, there are two meters of space into which two pieces of apparatus (A and B) must fit. We know that it is good to make A and B each as long as possible, but don't know how to apportion the 2 meters between the two.

Then we may define two substitution markers based on a single optimization variable, the length of A:

```
{lengthA}  lengthA  f8.4  1.  0.

! lengthB = 2 - lengthA
{lengthB}  lengthA  f8.4  -1. 2.
```

Those substitution markers are placed in the template where the lengths of A and B are defined. This is a way to impose simple coupled constraints on the apparatus design.

Frequently unasked questions:

Is it ok to have an optimization variable with no associated substitution markers? Yes, although the variable will play no part in the optimization other than slowing it down.

Is it ok to define a substitution marker that is not used in any template? Yes, although the variable associated with the substitution marker will play no part in the optimization unless it is also associated with other substitution markers which do appear in a template.

Is it ok to leave in place a substitution marker associated with an optimization variable that has been temporarily removed from the optimization? No, that will produce an immediate error halt. Just put ! in front of the line if you want to keep it around for later use.

Section ***MeritVariables***

Each quantity that may enter into the calculation of each configuration's merit is defined as a merit variable. Common examples: emittance (low is good), muon survival fraction (high is good).

For each merit variable, you specify a name, a merit combination code, the value of the merit variable to produce a normalized merit of 0.1, the value of the merit variable to produce a normalized merit of 0.9, the minimum merit, and the maximum merit.

Simple cases: The merit combination code is M for multiply, the minimum merit is 0 and the maximum 1. For normalized merit of .1, the value of the merit variable of which you would say "this is bad enough that I wouldn't really want to see it get much worse." For normalized merit of 0.9, the value of the merit variable for which you would say "this is so good that it really doesn't matter if it gets much better."

The merit of all the merit variables is computed, and then the product of all the individual merits becomes the final figure of merit.

For special cases, there are more complex ways of computing the final merit. For example, suppose there are two merit variables of which you might say "I care about both of these, but I care twice as much about B as I do about A". Then you might define:

```
  meritVarA  A1   --.-  --.-  0.0  1.0
  meritVarB  A1   --.-  --.-  0.0  2.0
```

In this case the two merits will be added together with a weighting factor of 1 (A) or two (B) to form partial merit A1. (There can be up to nine partial merits A1...A9). Imagine that there are also multiplicative (M) partial merits (M1, M2, M3) defined. The final merit M = M1 * M2 * M3 * A1 . . . .

Suppose that you have a merit variable M3 that you don't want to contribute very strongly to the overall merit. If you set the minimum partial merit to 0.8 and the maximum partial merit to 1.2, for example, that will make the total merit relatively insensitive to the partial merit in question.

```
 meritVarA  M    --.-  --.-    0.8  1.2
```

Frequently Unasked Questions:

Where do the values of the merit variables come from? They come from the simulator or a postprocessor that analyzes the results from the simulation. They are transmitted in merit file for010.dat, having a format:

```
  meritVarNameA   0.64754   (merit variable A's name and value)
  meritVarNameB   1.347     (merit variable B's name and value)
      etc.
```

Must every merit variable declared in OptCont.dat have a corresponding line in the merit file for010.dat? Yes.

Is it ok to have quantities defined in the merit file for010.dat that aren't defined as merit variables in the OptCont.dat file? Yes. You may output all the variables that you think might possibly be used as merit variables someday, and only use a small subset of them.

How does OPTICOSP distinguish between merit variables that are better the bigger they are (e.g. muon survival rate) and those like emittance that are better the smaller they are? By the way that the 10% merit and 90% merit points are defined. If the variable's value at the 90% merit point is the larger, then bigger (variable value) is better (higher merit). If the variable's value at the 90% merit point is the smaller, then smaller is better.

What values does the merit function take between the 10% point and the 90% point? The actual function used is a scaled and offset arctangent function. See the charts in the appendix.

Can I just calculate my own final merit in my postprocessor rather than having OPTICOSP do it? Yes. You might compute the configuration merit and output it in ***for010.dat*** as totalMerit. The merit should be positive everywhere, higher merit means better apparatus performance. In OptCont.dat, declare a single merit variable:

```
totalMerit  M  0. 1. 0. 1.
```

Section ***FileTransfers***

For every template file that is to be altered by substitution of variable values for substitution markers, this section must state the name of the template and the name of file to be read by the simulator (or a simulator preprocessor).

In the example above, for001.tpl is the template for for001.dat, the control file input for ICOOL.

Section ***Parameters***

Here a few optimization parameters must be defined. See the example above for details.

Normalized Optimization Variables

From the user's point of view, optimization takes place within an N-dimensional box defined by the bounds of the N optimization variables. The bounds are defined in user coordinates. For example, the user may declare that a magnetic field lie between 1.00 and 1.40 tesla, that an absorber be between 0.4 and 0.6 meters long.

The optimizer itself works in normalized coordinates, where the lower bound of the optimization variable is zero and the upper bound is one. Neglecting for the moment the fact that optimization bounds are not absolute, the optimum will generally be found within (or very close to) a unit N-dimensional hypercube.

When the optimization has been completed, the values of the optimization variables are transformed back to user coordinates: magnetic field = 1.23 tesla, absorber length = 0.45 meters....

In most cases, the user need not concern himself with normalized coordinates, but there are a few cases in which they are useful. One is determining the point at which the optimization is good enough. The parameter StopCriterion defines the size of a small volume in normalized optimization space. If, for the past few optimization steps, the optimum has always resided within a single such volume, the optimization is declared "good enough", and the optimum is calculated as the "center of gravity" of the last few points.

Normalized coordinates also provide a quick way to determine whether the optimum lies near or beyond one or more optimization variable bounds. In-bounds optima always lie between normalized values 0 and 1, for each optimization variable.

Merit variables are also computed in normalized form; the normalized merit is always between 0 and 1. The 10% and 90% points in the merit transfer function refer to normalized merits of 0.1 and 0.9. As explained above, the user may request that these merits be offset by an additive constant or scaled by a multiplicative constant before being used in the final merit calculation.

A Final Word or Two

OPTICOSP is not a polished program; it has seen only very restricted use on real optimization problems. There are doubtless things which are either unclear or simply don't work as expected. There is a tendency among physicists who encounter a stumbling block in a software package to beat on the problem far longer than appropriate -- until it's metaphysically certain that the problem lies in the software or the documentation -- before calling for help.

Don't do that. I'll be happy to help you get your optimization going, and within reason to tailor OPTICOSP to the needs of particular problems. The potential user base for this software is probably not so large that this approach will be unduly burdensome; if it becomes so, I will let folks know.

When I'm not at Ole Miss, I am still reachable by email, and still happy to help you with your optimization setups.

If there is one thing I would caution about, it is "Don't deploy OPTICOSP too soon". Before it can wielded effectively, you must have developed a good feel for the problem at hand. This usually only comes with running some simulations by hand, and then doing a preliminary optimization using a fairly wide search space and lots of summary configurations. Only then are you ready to do the final, often time-consuming precision optimization. Think of OPTICOSP as a scalpel in the hands of a skilled surgeon who has studied the x-rays and carefully planned his procedure, not a snake oil cure-all that can just be poured over a problem to resolve it.

Example: A Closed Orbit Optimization for the RFOFO Cooling Ring

Here I describe a simple and somewhat unusual optimization originally set up by Juan Gallardo. Given apparatus consisting of one cell of a muon cooling ring design, search for a closed orbit for particles having a specified longitudinal momentum.

In accelerator (e.g. ICOOL) coordinates (S, X, Y, Ps, Px, Py) a muon enters the cell at S=0, Xstart, Ystart, Ps=Pstart, Px=0, Py=0). Physics processes -- acceleration and absorption -- are turned off. At the end of the cavity, the muon exits the apparatus at position Sfinal, Xfinal, Yfinal. The goal: to find, for a specified Pstart, a transverse position (Xstart,Ystart) such that Dist (Xstart, Ystart, Xfinal, Yfinal) is zero where the distance is the ordinary:

$$\mathtt{DistXY} \equiv \mathtt{Dist\ (Xstart,\ Ystart,\ Xfinal,\ Yfinal)} = \mathtt{Sqrt((Xfinal-Xstart)}^{2} + \mathtt{(Yfinal-Ystart)}^{2}\mathtt{)}$$

The optimization was performed for about a dozen different values of Pstart.

For this optimization, the apparatus configuration, as specified in ***for001.dat***, does not change. Instead, a single particle with starting coordinates specified in beam file ***for003.dat*** is propagated through the cell. ICOOL delivers intermediate and final coordinates in ***for009.dat***. A post-processor extracts (Xstart,Ystart) from ***for003.dat*** and (Xfinal,Yfinal) from ***for009.dat***, computes the distance DistXY, and outputs the merit file ***for010.dat***. Three merit variables -- DistXY, Xfinal-Xstart, and Yfinal-Ystart -- were output. Only DistXY was actually used in the optimization.

What files must be prepared to perform such an optimization?

1. OptCont.dat

The optimization control file defines optimization variables ***Xstart*** and ***Ystart*** and appropriate substitution markers for each. It specifies a single merit variable ***DistXY***, such that smaller is better. It specifies one template file ***for003.tpl*** producing one ICOOL input file ***for003.dat***. The number of particles to be propagated is one. The convergence requirement is extremely rigorous we can afford this because each simulation is very quick and there is no statistical fluctuation to consider, as long as one stays well away from numerical loss of significance.

2. for003.tpl

The template file for ICOOL's beam input file. It consists of header lines and a single particle definition line. Two substitution markers, {Xstart} and {Ystart}, appear in this file. OPTICOSP will edit the template to produce a new ***for003.dat*** for each ICOOL simulation.

3. simulate.bat

This is the file that specifies what it means to do a simulation. Here we set the working directory to the place where all the data files are located (important!), execute ICOOL, and then execute the postprocessor. Assuming that all goes well, the postprocessor will write the merit file ***for010.dat*** after each simulation, and OPTICOSP will use that information to decide what (Xstart,Ystart) position to try next.

4. postproc
This is the executable image of the custom postprocessor used to generate the merit file. Since it is a program specific to this optimization task, it was placed in this application's subdirectory, but you can put it anywhere as long as it is properly referenced in simulate.bat. The fortran source postproc.for is in the application's subdirectory too.

Of course there are other ICOOL input files that needed to be prepared and placed in the application's subdirectory, including ***for001.dat***, and an input file specifying the magnetic field.

It took about 40 passes of the simulator (probably about 60-70 simulations) to complete the task. The optimizer was not allowed to stop until DistXY had become very small.

The entire optimization took less than a minute on a not-so-fast PC. This demonstrates that the optimizer itself, including the overheads involved in starting the simulator and its postprocessor dozens of time, runs very quickly. In the more typical case, the simulations themselves take almost all the execution time (hours, days).

In this case, I was quite sure that the merit function was smoothly varying and the optimizer had landed somewhere very close to the one true optimum. To confirm that, I ran (by hand) a final ICOOL simulation at the supposed optimum and let the postprocessor compute XYdist. It was a few tens of nanometers!

Does this allow me to claim that the closed orbit points for this apparatus are known to tens of nanometers? No! All I know is that given the simulation as defined, the magnetic field as defined, the optimizer found a very well-defined optimum. The field is calculated from a multi-pole parameterization of a highly non-uniform field generated by numerical integration, starting with a small ensemble of idealized current sheets. The particle is stepped through that field using a Runge-Kutta stepper. None of this is likely to yield trajectories good to tens of nanometers. To assess the simulation errors, it would be reasonable to run ICOOL again, at the nominal optimum, with a better (or at least different) field map and with a smaller (or larger) step size and see how far (Xfinal,Yfinal) wanders around.

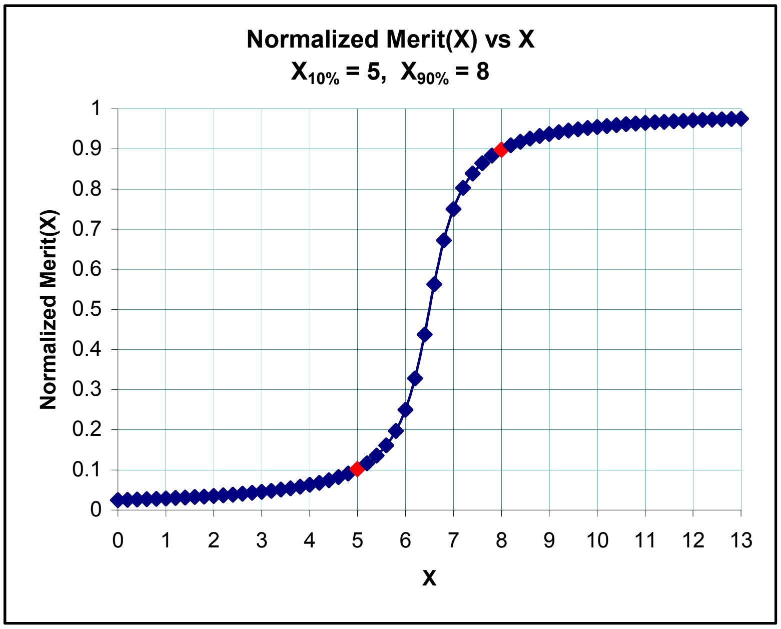
In the case of typical simulations, in which probabilistic physics processes are turned on, one must also vary the number of particles propagated and the random number seed to assess the uncertainty in the results. It is very important to run and carefully examine summary simulations to ensure that the merit function is sufficiently well behaved, so that one can say with assurance that the optimizer has homed in close to the one true optimum.

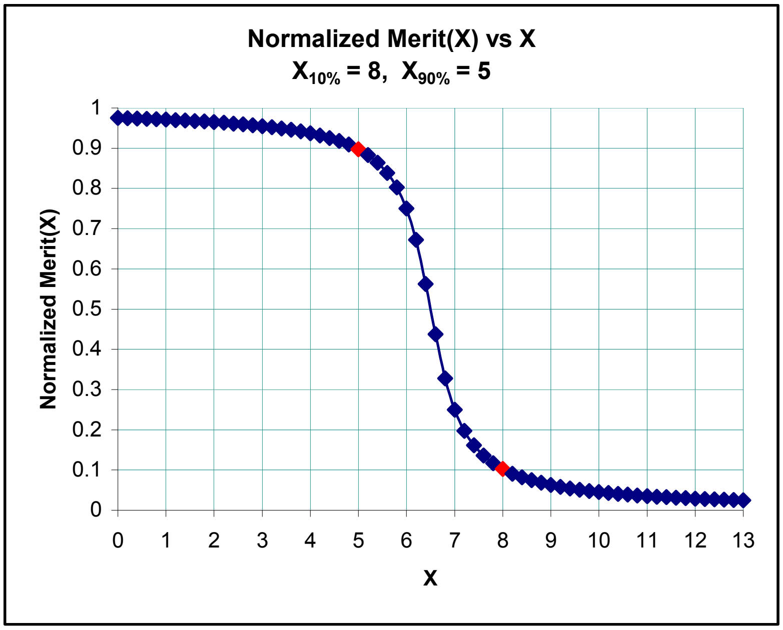
Appendix A: Example merit functions

X is the merit variable's value. The merit is an arctangent, scaled and offset.

Upper plot: X M 5.0 8.0 0.0 1.0 ($X_{10\%}$ = 5, $X_{90\%}$ = 8; higher = better)

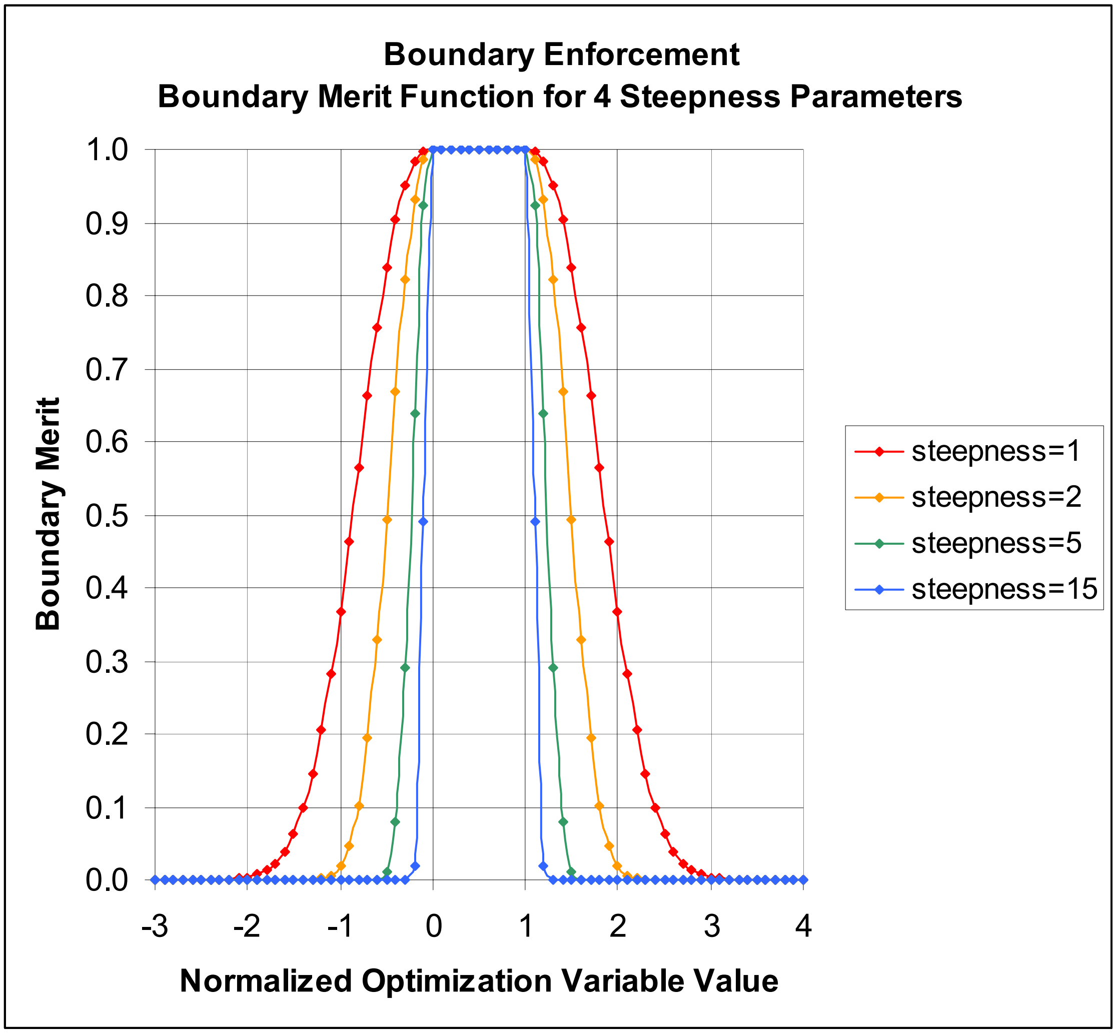
Lower plot: X M 8.0 5.0 0.0 1.0 ($X_{10\%}$ = 8, $X_{90\%}$ = 5; lower = better)





Bounds Enforcement -- 4 Different Levels of Enforcement

Low numbers (1-2) yield very lax enforcement of optimization variable boundaries. As the strictness parameter increases in value, the bounds enforcement becomes more and more strict.



Appendix B: Selected Files for the Optimization Example

OptCont.dat -- Optimization Control File

```
! + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + +

WorkerNames

! For OPTICOSP, just one worker, and its name and speed don't matter.
  dellmonster    2000

! + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + +

OptimizationVariables

! Define the optimization variables. Each has:
!  --a name (often the same as the fortran source code name)
!  --a lower bound
!  --an upper bound
!  --a number specifying how steep the merit curve is at the boundaries
!  --the number of summary configurations to be computed
!  --an optional comment at the end of the line
!  The X and Y starting positions of the beam particle may be adjusted.

  !First optimization variable
  xStart   0.00800    0.01000  10.  11

  !Second optimization variable
  yStart  -0.00100    0.00100  10.  11

! + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + +

InputSubstitutions

! Define the input file substitutions. Each has:
!  --a substitution marker e.g. {abc}
!  --an optimization variable name
!  --an output format specifier
!  --a multiplier M
!  --an addend A
!  The substitution markers for X and Y starting positions.

! For every occurrence of the marker in a ghost input file, the value
! M * optValue + A is substituted, formatted as specified.

  {xStart}  xStart     f15.10   1.0  0.0
  {yStart}  yStart     f15.10   1.0  0.0

! + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + +

MeritVariables

! Define the merit variables. Each has:
!  --a name (often the same as the fortran source code name)
!  --a merit combination code, M, A1, A2 ... A9
!  --merit variable value at normalized merit = 0.1
!  --merit variable value at normalized merit = 0.9
!  --minimum value of the merit function (usually 0.0)
!  --maximum value of the merit function (usually 1.0)
!  The figure of merit is completely defined by the difference between
!   (Xstart,Ystart) and (Xfinal,Yfinal)

  !X-Y Distance
  ! Anything greater than 0.0003 is pretty bad
  ! Anything less than 0.00001 is just fine
  xyDist   M  0.0003  0.00001   0.0  1.0


! + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + +

FileTransfers

! Define the files to be sent from boss to workers. All files are
! sent from the bd (boss-data) area of the boss to the wd
! (worker-data) area of each worker.
!
! If a single filename is specified, then the file is transferred
! unmodified and with the same name to the designated worker.
!
! If two filenames are specified, then the input file (first name)
! is subjected to variable substitution, producing an output
! file (second name) in the boss data area. The output file is then
! copied to the designated worker and deleted from the boss data area.

  ! Beam particle definition file for ICOOL
  for003.tpl for003.dat


! + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + +

Parameters

! Define various operating parameters controlling the optimization
! process. Each line consists of a name and a value. All parameters
! must be explicitly specified; there are no defaults.
!
! nPartSim -- number of particles to propagate each optimization
!  simulation (1 particle for this study)
! nPartSum -- number of particles to propagate each summary
!  simulation (1 particle for this study)
! stopCriterion -- how small a volume you wander within before you
!  stop simulating, in normalized coordinates. In this study, a
!  very rigorous criterion.
! maxOptSims -- maximum number of optimization simulations
! pauseEachSim -- 0 for free-running, 1 to pause after each simulation

  nPartSim  1
  nPartSum  1
  stopCriterion 0.000001
  maxOptSims 100
  pauseEachSim 0
```

Simulate.bat -- The simulation script file

Set the current working directory to the job subdirectory (data file location)
Run ICOOL
Run the postprocessor that generates the merit file.

```
cd /home/oc/opticosp/applications/gallardo1
/home/oc/icool252/icool
/home/oc/opticosp/applications/gallardo1/postproc
```

for003.dat -- The beam particle file

The single beam particle is defined. The starting X and Y position may be changed by OPTICOSP, so the numeric values are replaced by substitution markers.

```
2nd phase rot unipol corr grad wind (DEC00 2.15 S7a) x,y offset for clos orb
    0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000
    1 0 2 0 0.0 1.0    {xStart}  {yStart} 0.0   0.0 0.0  0.1500   0.0 0.0 0.0
```

Excerpt from the optimization *report* file

The (Xstart,Ystart) -- the first column of numbers -- is well within the search space specified.

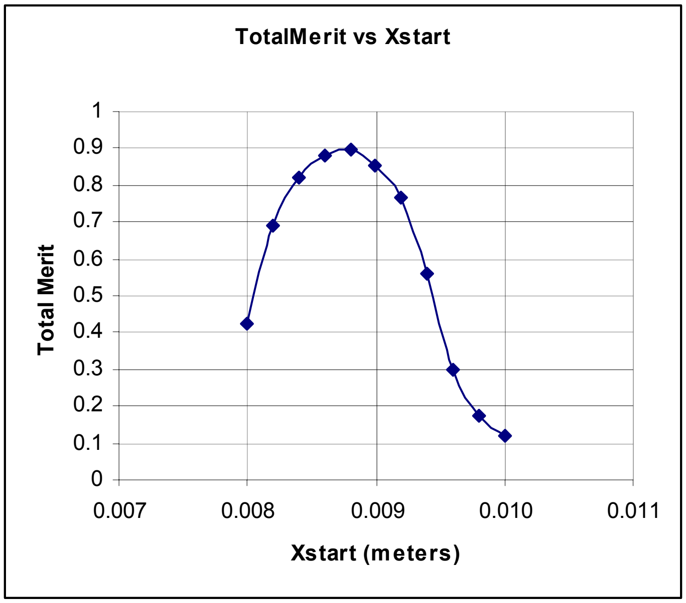
```
 LOCATION OF THE OPTIMUM - - -
   Optimization variable name
   Optimal value
   Normalized optimal value
   Lower bound
   Upper bound

 xStart      0.00874707      0.37353325      0.00800000      0.01000000

 yStart     -0.00001709      0.49145666     -0.00100000      0.00100000
```

Plot from data in the *report* file

From the simulation summary report, we can check to see whether the merit function is well-behaved. In this case, it is. Remember that in this example there are no random physics processes turned on, so that having an adequate particle count is not an issue.



References

[1] Steve Bracker, "OPTICOOL - A Multiprocessor Optimizer for Muon Cooling Simulation Codes", March 2001,
http://www-mucool.fnal.gov/mcnotes/public/pdf/muc0197/muc0197.pdf

[2] R. C. Fernow, "ICOOL: A Simulation Code for Ionization Cooling of Muon Beams", April 1999,
http://accelconf.web.cern.ch/AccelConf/p99/Papers/THP31.pdf

[3] Steve Bracker, "The XICOOL Preprocessor for ICOOL Input Files", March 2001,
http://www-mucool.fnal.gov/mcnotes/public/pdf/muc0196/muc0196.pdf