## Introduction

Optimisation is a technique developed within Operations Research which aims to minimise of maximise the value of a function. For example, we may wish to minimise the function F = f(x), where **x** is called the decision variables, subject to a set of constraints, **g**(**x**) = 0; **h**(**x**) > 0; i(x) >= 0. In other words, we are answering what values in **x** gives rise to the minimum value of F.

This can be extended to consider multiple objectives, whereby we may want to minimise **F** = f(**x**). In general, there may not be a single value of **x** that minimises each objective function F, contained in **F**. Therefore, the aim of multiobjective optimisation is to find the Pareto optimal set of decision variable values (non dominated solutions wrt **F**)

This is a very mathematical construct, but one that has immense practical use. For example, a seemingly simple task such as grocery shopping is inherently an optimisation problem. We have constraints (how much money we have available and that we need to buy enough food to maintain life for those we are shopping for), and objectives (such as minimising cost, maximising enjoyment, maximising nutrition). Note that many of these objectives are often competing. This is typical of most real-world situations.

Evolutionary algorithms have been developed to help solve real-world multiobjective optimisation problems. One such evolutionary algorithm is called the Non-Dominated Sorting Genetic Algorithm (version 2). It was developed by Deb, and published in 2001?. It is a widely used evolutionary algorithm that has been shown to be suitable for solving optimisation problems across the breadth of water resources management and planning.

This document describes an implementation of an NSGAII. It was developed for Michael Di Matteo using WenYan’s code as a basis, but may be used for other purposes as well.

In particular, the code has been developed for linking the NSGAII algorithm to an external model that calculates the objective function, which is executed through a system call within the NSGA code. The means of linking is through the use of ascii text files for communication between the NSGAII executable and the objective function (model) executable. For the decision variable **x**, that the algorithm wants to evaluate f(x) for, the model writes x within a text file, and calls a pre-specified command line executable. This command line executable calculates the objective functions and constraints, and writes these to a separate ascii text file. Once the executable has exited (finished running), the NSGAII then reads in the objective function values from this text file and injects these values into the NSGAII heuristic.

## Decision variable text file format

The decision variable , is represented in the decision variable ascii file as:

5

0

0

5

0

0

0

0

1

0

5

1

## Objectives and Constraints text file format

This file includes comments, which are to be ignored. Comments begin with the first ‘/’ character on a line (not necessarily preceded with a blank), and continue to the end of the line. The comment does not need to necessarily start at the beginning of the line.

The objective vector , and constraint **G** = **g**(**x**) = 0, for which , is represented in the objectives and constraints ascii file as:

//Objective function values and constraints violations

2 /number of objectives (Jeff, feel free to skip this line if no\_obj is input elsewhere)

12345678.12 /objective 1 value

12345678.12 /objective 2 value

7 /number of constraints (Jeff, feel free to skip this line if no\_cons is input elsewhere)

1.0 /cons - value returned for constrained MUSIC outputs

1.0

1.0

1.0

0.0

0.0

0.0

0.0

0.0 / overallcons - threshold for constrained MUSIC outputs - constraint met if = 0.0, constraint violated if > 0.0

0.0

0.0

0.0

0.0

0.0

0.0

## My modified code for evaluation objectives in Wenyan’s NSGAII

void evaluatepop(population \*pop\_ptr, int gen)

{

/\*===========================start population evaluation==================================\*/

pop\_ptr->ind\_ptr=&(pop\_ptr->ind[0]);

pop\_ptr->maxrank=0;

for (i=0;i<popsize;i++)

{ //this loop asigns fit, cons & overalcons to each ind

pop\_ptr->ind\_ptr=&(pop\_ptr->ind[i]);

if (pop\_ptr->ind\_ptr->tag==1) ///////// JEFF: What is this. Perhaps a tag to

///////// specify whether the obj functs and

///////// constraints have been evaluated yet?

{

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* \*

\* Jeff's interpretation of variables in this function: \*

\* \*

\* 1. It looks like Wenyan's NSGAII can consider decision variable vectors \*

\* consisting of both float and integer types. \*

\* 2. To make matters more confusing, it looks like the integer decision \*

\* variables are stored in pop\_ptr->ind\_ptr->chrom1. Then, in a funct \*

\* decode, which is called before evaluatepop, these are variables are \*

\* given a physical meaning through mapping the int. values to things \*

\* such as pipe size etc, which are stored in the arrays \*

\* xchrom1[maxchrom1], \*

\* ychrom1[maxchrom1], \*

\* zchrom1[maxchrom1], \*

\* uchrom1[maxchrom1], \*

\* vchrom1[maxchrom1]; \*

\* These were then copied into local array variables defined in this \*

\* function, in Wenyans original code. \*

\* NOTE: Copying takes time and memory. There is no reason not to use \*

\* the variable values as stored in the pop struct which is passed \*

\* to this function as a pointer. [As far as I can tell \*

\* \*

\* MY CHANGES \*

\* a. Integers dvs copied into the vector x, \*

\* copied from pop\_ptr->ind\_ptr->chrom1 \*

\* \*

\* b. Floats dvs copied into the vector r, \*

\* copied from pop\_ptr->ind\_ptr->chrom2 \*

\* \*

\* \*

\* \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

std::vector<int> x(pop\_ptr->ind\_ptr->chrom1, pop\_ptr->ind\_ptr->chrom1 + no\_intevar);

std::vector<double> r(pop\_ptr->ind\_ptr->chrom2, pop\_ptr->ind\_ptr->chrom2 + no\_realvar);

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* \*

\* Jeff's interpretation of what Michael needs \*

\* 1. It looks as though Michael is optimising a problem with only \*

\* integer decision variables, and we print these to an ascii file \*

\* 2. The name of this ascii file has not been specified. We will call it \*

\* "dv.txt" \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

DecisionVariable2Ascii<int> myWriter("dv.txt");

myWriter(x);

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* \*

\* Evaluate the objectives \*

\* 1. Call the system excutable \*

\* 2. Read in the obj. funct. values and constraints \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

//Call the system executable

/\*!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

\* \*

\* REPLACE "A.OUT" WITH THE COMMAND LINE CALL, TO EXECUTE YOUR CODE \*

\* \*

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!\*/

system("a.out");

//Read in the obj funct. values and constraints

bool is\_obj = no\_obj > 0 ? true : false;

bool is\_cnstr = no\_cons > 0 ? true : false;

Ascii2ObjectiveValues<double, double> myReader("testObj.txt", is\_obj, is\_cnstr);

std::shared\_ptr<ObjectivesAndConstraints<double, double> > obj\_and\_cnstr = myReader();

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* \*

\* Jeff's interpretation of outputs required from this function: \*

\* \*

\* 1. The objective functions get put into the fit[] array of each ind in \*

\* as accessed through the pop\_ptr pointer. MAKE SURE TO SPECIFY \*

\* NUMBER OF OBJECTIVE FUNCTIONS, CONSTRAINTS ETC. \*

\* a. It looks to me that inputs for the GA are specified in moga.c or \*

\* are asked of the user from the command line when run \*

\* \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

// Sanity checks

if (obj\_and\_cnstr->objectives.size() != no\_obj)

{

std::cerr << "ERROR: Wrong number of objectives\n";

}

if (obj\_and\_cnstr->constraints.size() != no\_cons)

{

std::cerr << "ERROR: Wrong number of constraints\n";

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* \*

\* NOTE ON OVERALLCONS \*

\* \*

\* Wenyans code uses a variable overallcons (associated with an individual \*

\* to specify the overall constraint. Mike - your input file defined this \*

\* as a vector of threshold values for constrained MUSIC outputs - \*

\* constraint met if = 0.0, \*

\* constraint violated if > 0.0 \*

\* \*

\* WHAT I'VE DONE. \*

\* If the "cons" value is > then the "overallcons" value, then I work out \*

\* how much it is over (cons - overallcons). \*

\* from Wenyan's code, I think overallcons should be positive. With \*

\* greater values indicating greater constraint violation \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

for (k=0;k<no\_obj;k++)

pop\_ptr->ind\_ptr->fit[k]=obj\_and\_cnstr->objectives[k]; /\*assign fitness to fitness array fit[no\_obj] of ith indi \*/

if (no\_cons>0)

{

error = 0.0;

for (k=0;k<no\_cons;k++)

{

double constraint\_violation = obj\_and\_cnstr->constraints[k]

- obj\_and\_cnstr->constraint\_thresholds[k];

pop\_ptr->ind\_ptr->cons[k] = (constraint\_violation > 0) ?

constraint\_violation : 0;

error += pop\_ptr->ind\_ptr->cons[k];

}

pop\_ptr->ind\_ptr->overallcons=error; /\*assign the overall constarints to the individual\*/

}

noeval+=1;

pop\_ptr->ind\_ptr->eval=noeval;

pop\_ptr->ind\_ptr->gen=gen+1;

} //if the tag==1

} //check each ind in the population

}