A Problem Generator

Jörg Fliege
Fachbereich Mathematik
Universität Dortmund
44221 Dortmund
Germany
email fliege@math.uni-dortmund.de

September 1, 1999

Abstract

Scientists in Operations Research and Optimization face a problem which increases in it's difficulty almost daily. More and more different objective functions have been considered for minimization, and it is therefore more and more difficult to find a new, previously unconsidered one. We propose a problem generator that can automatically generate new optimization problems. These problems can then be analyzed by scientists with, e. g., standard techniques, and standard algorithms can be applied. After such a generating process, all that remains is to invent an application for the new problem.

1 Introduction

Scientists in Operations Research and Optimization face a problem which increases in it's difficulty almost daily. More and more different objective functions have been considered for minimization, and it is therefore more and more difficult to find a new, previously unconsidered one. We propose a simple remedy for this situation.

While it might be possible to come up with new research results even for objective functions which have been considered before, this is seldom seen as a solution to the problem described above. In order to come up with results that can be classified as new, it is often the case that one has to know at least a part of the literature. As a consequence, the generation of new problems can be considered as easier. This the approach taken in this paper.

We propose a simple problem generator that can automatically generate new optimization problems. These problems can then be analyzed by scientists with, e. g., standard techniques, and standard algorithms can be proposed to solve them.

2 The Problem Generator

We propose a simple problem generator that can generate new objective functions. The problem generator starts with a string of length 1 containing the symbol "@", takes as input a number of grammatical rules of the form

$$@ \longrightarrow \text{string},$$

where the string may contain the symbol "@" again, and applies these rules recursively at random to the resulting string. It stops after a prespecified number of recursions. Any unresolved occurences of the recursion symbol @ are then replaced by a variable name x_i, $(i = 1, \ldots, n)$, where i is chosen randomly and the number of unknowns n has to be prespecified by the user.

3 Numerical Results

Actual numerical results, are of course, unimportant. Instead, a simple implementation of the principle described in the last section will be given. It was found that the idea described above is so simple that it can even be implemented in Pascal. See the appendix for the corresponding source code. Of course, the implementation is totally inefficient, as it has to be the case for a prototype code. The input parameters as well as the rules have to be given in a file with name `rules`. The output is standard LaTeX code. This is, at the moment, the only important output format, since actual calculations with the generated functions within a computer code are not an issue when one tries to find efficient algorithms. In our example, the input file looks like this:

```
5
2
18
@ + @
@ - \left( @ \right)
@ @
\sqrt{\left| @ \right|}
\frac{1}{d+\left(@\right)^2}
```

```
\cos\left(@\right)
\sin\left(@\right)
@ + c
\alpha @
 x^{\top} a
 \Vert x \Vert_2
 \Vert x \Vert_1
 \Vert x \Vert_{\infty}
 \Vert x \Vert_p
\sum_{i=1}^m w_i \left( @ - a_i \right)
\max_{i=1}^m w_i \left( @ - a_i \right)
\left| @ \right|^p
\sqrt[p]{\left| @ \right|}
```

As it can be seen, we employ the recursive rules exactly five times, we consider problems with two unknowns, and we have specified eighteen rules. The first rule given takes the form

$$\texttt{@} \longrightarrow \texttt{ @ + @ },$$

while, e. g., the last one has the form

$$\texttt{@} \longrightarrow \texttt{\textbackslash sqrt[p]\{\textbackslash left| @ \textbackslash right|\}}$$

i. e. the place holder @ is replaced by $\sqrt[p]{|@|}$. Of course, more or completely different rules can be specified at will. Note also that several rules are able to stop the recursive process prematurely. Moreover, as it can be seen in the input file displayed above, the rules used here are rather locationally oriented, i. e. they mimic problems typically encountered in location science. One of the first applications of the program produced the following results:

```
f(x_1, x_2) :=\alpha \sin\left( \Vert x \Vert_{\infty} + x_{1} + c\right)

f(x_1, x_2) :=\sqrt[p]{\left| \sqrt{\left| \max_{i=1}^m w_i
\left( x_{2} + c^{\top} a - a_i \right) \right|} \right|}

f(x_1, x_2) :=\sum_{i=1}^m w_i \left( \sqrt{\left| \left|
\sum_{i=1}^m w_i \left( x_{1} - a_i \right) \right|^p \right|} -
a_i \right)\Vert x \Vert_2

f(x_1, x_2) :=\sum_{i=1}^m w_i \left( \sqrt{\left| x_{2} \sum_{i=1}^m
w_i \left( x_{1} + x_{2} - a_i \right) \right|} - a_i \right)
```

These outputs translate readily to

$$f(x_1, x_2) := \alpha \sin\left(\|x\|_\infty + x_1 + c\right),$$

$$f(x_1, x_2) := \sqrt[p]{\left|\sqrt{\left|\max_{i=1}^{m} w_i\left(x_2 + c^\top a - a_i\right)\right|}\right|},$$

$$f(x_1, x_2) := \sum_{i=1}^{m} w_i \left(\sqrt{\left|\left|\sum_{i=1}^{m} w_i\left(x_1 - a_i\right)\right|^p\right|} - a_i\right) \|x\|_2,$$

and

$$f(x_1, x_2) := \sum_{i=1}^{m} w_i \left(\sqrt{\left|x_2 \sum_{i=1}^{m} w_i\left(x_1 + x_2 - a_i\right)\right|} - a_i\right).$$

The author firmly believes that none of these functions has ever been considered in location science. It it perhaps surprising that a comparatively simple strategy as the one proposed here can make such a significant contribution to research. After such a generation process, all that remains is to apply a standard optimization strategy and to invent an application with a corresponding model for the objective function. Then, a new paper has been finished.

The present implementation uses randomly chosen rules and recursion symbols. As a consequence, the results are somewhat arbitrary. Of course, it is relatively simple to replace this random element by a deterministic procedure. In this way, a complete list of all problems composed with the specified rules and with a fixed recursion depth can be generated. When the rules are chosen to reflect all functions appearing in a specific application field, one could therefore generate all objective functions which might occur in this field. After the generation process, these function can then be classified according to some criterion.

4 Outlook

The prospects are rather bright. Preliminary analysis of the computational results on an old 486 Intel machine showed that it is possible to generate several thousands objective functions within a few hours. Of course, not all of them are new. Nevertheless, it should be possible to write several hundred technical reports a year based on this scheme.

These reports can then be submitted to journals.

Even if only a fraction of these papers get accepted, a conservative estimate shows that it should be possible to produce about 10–20 publications a year. This is a great advantage for young scientists, whose prospects of getting a position are based only on the number of publications they have. Moreover, with a correspondingly enlarged bibliography, it should also be easier to get funding.

Further work should now be directed to automatic report-writing programs, which can improve the quantity of the output even more. However, it might be argued that the scheme outlined above puts a great pressure on the peer-review system, employed by most editors in the scientific publishing business, on the publishers, who have to print all these results, and on the readers. But the publishing houses can simply increase the prices for the journals, and nobody reads scientific publications anyway. What remains is the problematic refereeing process. To facilitate this process, an automatic rejection generator, following the lines of the program above, is in development. Further research is under away.

Disclaimer. The author takes no responsibility for any kind of damage to a brain or to a career that this work produces. The views and opinions expressed in this work are those of the author. No person except the author has proofread this paper, and the author did not listen to any suggestions at all. Moreover, while producing the scientific results outlined in this work, the author was neither partially nor fully supported by a grant from a research agency.

Appendix. The Source Code

```
PROGRAM Generator;

CONST place = '@';
CONST max_places = 255;
CONST max_no_op = 127;
CONST max_depth = 255;
CONST max_unknowns = 255;

VAR no_op : 1..max_no_op;
VAR no_place : 0..max_places;
VAR depth : 1..max_depth;
VAR unknowns : 1..max_unknowns;
VAR i, position, newindex : INTEGER;
VAR formula, localformula, newpart, firstpart : STRING;
VAR operator : ARRAY[1..max_no_op] OF STRING;
VAR index_place : ARRAY[1..max_places] OF INTEGER;
```

```
VAR inputfile : TEXT;

BEGIN;

RANDOMIZE;

ASSIGN(inputfile, 'rules');
RESET(inputfile);
READLN(inputfile, depth);
READLN(inputfile, unknowns);
READLN(inputfile, no_op);
FOR i := 1 TO no_op DO
    READLN(inputfile, operator[i]);
CLOSE(inputfile);

formula := place;

FOR i := 1 TO depth DO
    BEGIN;
       localformula := formula;
       no_place := 0;
       REPEAT
           position := POS(place, localformula);
           IF (position<>0) THEN
              BEGIN;
                 no_place := no_place + 1;
                 IF (no_place=1) THEN
                    index_place[1] := position
                   ELSE index_place[no_place]
                        := position + index_place[no_place-1];
                 DELETE(localformula, 1, position);
              END
             ELSE localformula := '';
       UNTIL(localformula='');
       position := index_place[1+RANDOM(no_place)];
       newindex := 1 + RANDOM(no_op);
       newpart := operator[newindex];
       DELETE(formula, position, 1);
       INSERT(newpart, formula, position);
   END;
```

```
WRITE('f(');
IF (unknowns=1) THEN WRITE('x_1');
IF (unknowns=2) THEN WRITE('x_1, x_2');
IF (unknowns=3) THEN WRITE('x_1, x_2, x_3');
IF (unknowns>3) THEN WRITE('x_1, \ldots, x_{',unknowns,'}');
WRITE(') :=');
REPEAT
  i := POS(place, formula);
  IF (i<>0) THEN
        BEGIN;
           firstpart := COPY(formula, 1, i-1);
           DELETE(formula, 1, i);
           WRITE(firstpart);
           WRITE('x_{',1+RANDOM(unknowns),'}');
        END
     ELSE
        BEGIN;
           WRITELN(formula);
           formula := '';
        END;
UNTIL(formula='');

END.
```