GENO™ 1.0

The GAUSS User Manual

Ike's Research Ltd

Technical Document
RD-3-2005
Edition 3.0

Principal Author

ISAAC SIWALE

Copyright © 1997-2006: Ike's Research Ltd

6 December 2006

CONTACTS

GENO™ is a trademark of Ike's Research Ltd.
Copyright © 1997-2006
All Rights Reserved Worldwide

Contact:

The Managing Director
Ike's Research Ltd
265, Southcroft Road
London
SW16 6QT

email: Ike@Siwale.FSBusiness.co.uk

Phone: +44-(0)208-769-8336

GAUSS™ and GAUSS™ Engine are trademarks of Aptech Systems Inc.
Copyright © 1983-2006
All Rights Reserved Worldwide

Contact:

The President
Aptech Systems Inc.
Box 250
Black Diamond
WA 98010
USA

email: info@Aptech.com

Phone: (425) 432-7855

Fax: (425) 432-7832

URL: www.Aptech.com

PREFACE

NOTICE

This document contains proprietary material. No part of this material, which is protected by this copyright notice, may be reproduced or utilised in any form, or by any electronic or mechanical means, including photocopying, recording or any other informational storage and retrieval system, without written permission from the copyright owner.

Copyright © 1997-2006: Ike's Research Ltd.
All Rights Reserved.

This document describes the usage, and presents some performance results of a proprietary GAUSS program called GENO. GENO is an acronym for ***G***eneral ***E***volutionary ***N***umerical ***O***ptimiser: the word *general* is here used not in the sense of GENO being "able to solve all problems",[1] but rather in the sense that it is effective on a relatively wide range of problems as compared to most existing algorithms.[2]

GENO is a real-coded genetic algorithm that can be used to solve uni- or multi-objective optimisation problems. The problems presented may be static or dynamic in character; they may be unconstrained or constrained by equality or inequality constraints, coupled with upper and lower bounds on the variables. The variables themselves may assume real or discrete values in any combination. In fact, except for the relatively benign requirement that, if present, all equation constraints should preferably be affine in the current control, the algorithm does not require the problem presented to have any other special structure.

Although the generic design of the algorithm assumes a multi-objective dynamic optimisation problem, GENO may be "specialized" for other classes of problems such as the general static optimisation problem,[3] the "mixed-integer" problem, and the two-point boundary value problem, by mere choice of a few parameters. Thus, not only can GENO compute different types of solution to multi-objective problems, it may also be set to generate real or integer-valued solutions, or a mixture of the two as required, to uni-objective static and dynamic optimisation problems of varying types. These properties are easily pre-set at the problem set-up stage of the solution process. The design of GENO includes a quantization scheme that significantly enhances the rate of convergence, as well as the quality of the final solution. Furthermore, unlike other algorithms of the same type, GENO includes a stochastic mechanism for stopping the search when the solution is within a user-defined tolerance.[4]

A representative sample of numerical examples that covers a wide range of problem-types and solutions is provided to demonstrate GENO's capabilities. From the reported results, it is evident that GENO consistently out-performs many evolutionary optimisation methods in all cases in which a comparison was possible; and, in terms of solution quality, its performance is at least as good as some deterministic algorithms in most cases.

---

[1] The implications of the 'No Free Lunch Theorems' first enunciated by Wolpert and Macready (1997) render this improbable.

[2] The range of examples presented in this document attest to this assertion. Furthermore, Zhang (2001, p.34) presents a capability map of 20 existing NLP solvers against which GENO may be compared. It should be noted that, unlike GENO, none of the solvers considered by Zhang address the multi-objective optimisation problem (see Table 3A of Appendix III).

[3] It could be argued that the generalisation claimed here is "the wrong way round" since it has long been well known that a discrete-time optimal control problem is essentially a *static* mathematical program. This follows immediately from the variable stacking technique which considers each sampling instant as simply a label for an independent variable. However, GENO design is different: it does not rely on explicit use of variable stacking per se; rather the stance taken is that the dynamic case is actually more fundamental because the static problem can always be viewed as a dynamic problem evolving over one time period.

[4] This stopping mechanism is yet to be implemented in the GAUSS version of GENO.

Copyright © 1997-2006: Ike's Research Ltd

CONTENTS

Contacts i

Preface ii

Contents iii

1. Introduction 1

2. Programming 2

2.1. Coding GENO Program Files 2

2.2. Coding GENO Data Files: Static Optimisation Problems 2

2.3. Coding GENO Data Files: Dynamic Optimisation Problems 3

2.4. Coding the Main Source Files 4

2.5. Program Output 8

3. Worked Examples 11

3.1. Preamble 11

3.2. Scope 11

3.3. Default Parameter Values 11

3.4. Problem Preparation 12

3.4.1. The Basic Approach 12

3.4.2. The NCP Approach 12

- Definitions 12
- Auxiliary M-program Formulation 13
- Implementation 14

3.4.3. The Elimination Approach I: Equality constraints 15

3.4.4. The Elimination Approach II: Inequality constraints 15

Copyright © 1997-2006: Ike's Research Ltd

3.5. Some Practical Examples 16

- Example 3.1: Static Unconstrained Optimisation: The Colville #4 Function 16
- Example 3.2: Static Unconstrained Optimisation: The Rastrigin Function 19
- Example 3.3: Static Constrained Optimisation: An Economic Dispatch Problem 22
- Example 3.4: Static Constrained Optimisation: A Pressure Vessel Design Problem 25
- Example 3.5: Static Constrained Optimisation: An Alkylation Process 29
- Example 3.6: Static Constrained Optimisation: Decentralised Economic Planning 32
- Example 3.7: Static Constrained Optimisation: Optimum Heat Exchanger Operation 37
- Example 3.8: Dynamic Optimisation: The Harvest Problem 42
- Example 3.9: Dynamic Optimisation: A Non-linear Resource Allocation Problem 45
- Example 3.10: Multi-objective Optimisation: Oligopolist Market Equilibrium Problem 48
- Example 3.11: Multi-objective Optimisation: The Euclidean Compromise Solution I 51
- Example 3.12: Multi-objective Optimisation: The Euclidean Compromise Solution II 55
- Example 3.13: Multi-objective Optimisation: Efficient Portfolio Selection 58
- Example 3.14: Multi-objective Optimisation: A Dynamic Non-cooperative Game 67
- Example 3.15: Mixed-Integer Non-linear Programming: Process Synthesis I 75
- Example 3.16: Mixed-Integer Non-linear Programming: Process Synthesis II 78
- Example 3.17: Mixed-Integer Non-linear Programming: Process Synthesis III 81
- Example 3.18: Generalised Disjunctive Programming: Process Synthesis IV 84
- Example 3.19: Generalised Disjunctive Programming: Process Synthesis V 87
- Example 3.20: Generalised Disjunctive Programming: Job Shop Scheduling 90

4. Summary 93

5. References 94

Appendices 98

I — An Introduction to Genetic algorithms

II — Example GENO Programs: A General Classification

II — Example GENO Programs: Reference to Further Information

III — A Classification of Constrained Nonlinear Programming Problems

III — A Comparative Capability Map of Some Existing NLP Solvers

Copyright © 1997-2006: Ike's Research Ltd

1. Introduction

The purpose of this document is to describe the usage of a program called GENO, and to present the results of some numerical experiments whose aim was to assess the efficacy of the said program. GENO is an acronym for ***G***eneral ***E***volutionary ***N***umerical ***O***ptimiser. It is a real-coded genetic algorithm that may be used on both static and dynamic optimisation problems, with or without equation and/or set constraints. It applies to single, as well as multi-agent problems and may be set to generate real or integer-valued solutions, or a mixture of the two as required. These attributes are easily pre-set at the problem set-up stage which in the GAUSS programming environment entails defining a few parameters and procedures in the '.gep' source files. The selected examples cover a wide range of problem-types; these are classified and indexed in Tables 2A and 2B of Appendix II.

GENO was conceived as a numerical search method for multi-objective or multi-agent[5] dynamic optimisation problems defined on a finite discrete domain. Using terminology borrowed from Optimal Control Theory, the general *p*-objective mathematical program (also referred to simply as: MP or M-program), defined on the time-sequence $\mathbf{T} = \{0,1,2,..,T\}$ may be stated thus:

MP1: $$\underset{\bar{\mathbf{u}}_T}{\text{Opt}}\ \mathbf{J}\left(\bar{\mathbf{x}}_{T+1}, \bar{\mathbf{u}}_T\right) = (J^1, J^2, \cdots, J^p)^T$$

Subject to: $$x^i_{k+1} = A^i_k(\bar{\mathbf{x}}_k, \bar{\mathbf{u}}_{k-1}) + B^i_k(\bar{\mathbf{x}}_k, \bar{\mathbf{u}}_{k-1}) \cdot u^i_k\ ; \qquad k \in \mathbf{T}\ ,\quad i \in \mathbf{N} = \{1,2,..,N\}\ .$$

$$G^j_k\left(\bar{\mathbf{x}}_k, \bar{\mathbf{u}}_k\right) \le 0\ ; \qquad k \in \mathbf{T}\ ,\quad j \in \mathbf{M} = \{1,2,..,M\}\ .$$

Notation: $\bar{\mathbf{x}}_k$, $\bar{\mathbf{u}}_k$ are sequences (up to time *k*) of the state and control vectors respectively;

$\bar{\mathbf{x}}_0$ is the *fixed* initial state vector (a known constant);

$\bar{\mathbf{x}}_{T+1}$ is the final state vector which may or may not be fixed.

$$\bar{\mathbf{u}}_k = [\mathbf{u}_0 \,|\, \mathbf{u}_1 \,|\, \mathbf{u}_2 \,|\cdots|\, \mathbf{u}_k] \in \bar{\mathbf{U}}_k \subset \mathbf{R}^{kN+N}$$

$$\bar{\mathbf{x}}_k = [\mathbf{x}_0 \,|\, \mathbf{x}_1 \,|\, \mathbf{x}_2 \,|\cdots|\, \mathbf{x}_k] \in \bar{\mathbf{X}}_k \subset \mathbf{R}^{kN+N}$$

$$\mathbf{u}_k = (u^1_k, u^2_k, .., u^N_k)^T \in \mathbf{U}^N_k \subset \mathbf{R}^N$$

$$\mathbf{x}_k = (x^1_k, x^2_k, .., x^N_k)^T \in \mathbf{X}^N_k \subset \mathbf{R}^N$$

The operator 'Opt' here means 'find the optimum' of the respective function: in uni-objective optimisation, this entails 'finding the maximum or minimum of a function'; in the multi-objective case this entails 'finding the Nash equilibrium point' or 'finding a point on the Pareto-efficient frontier'. The arguments of the generally non-linear functions $\mathrm{J}^p(\cdot), A^i_k(\cdot), B^i_k(\cdot), G^i_k(\cdot)$ are real-valued sequences that are in essence concatenations in time and *state* space.[6] Program MP1 is quite general; it includes, as special cases:

1. The uni-objective, non-linear, static optimisation problem (set: $P = 1$; $T = 1$);
2. The multi-objective, non-linear, static optimisation problem (set: $P \neq 1$; $T = 1$);
3. The uni-objective, discrete-time, non-linear, dynamic optimisation problem (set: $P = 1$; $T \neq 1$);
4. The multi-objective, discrete-time, non-linear, dynamic optimisation problem (set: $P \neq 1$; $T \neq 1$);
5. The two-point boundary value problem in which some component of the terminal state $\bar{\mathbf{x}}_{T+1}$ is fixed;
6. The "mixed-integer" problem in which some of the variables are restricted to discrete values.

This list is by no means complete: Appendix III lists other M-program categories that may be accommodated.

---

[5] The terms 'multi-objective' and 'multi-agent'—the latter derived from Game Theory—are essentially synonymous. They, and any other associated terminology, shall therefore be used interchangeably throughout this Manual. Refer to page 54, footnote 71 for an explanation.

[6] Note that $\bar{\mathbf{x}}_k$ is in essence matrix with dimensions, (N x k+1): it is also a time-indexed sequence, i.e. a *concatenation in time* of the state vector; on the other hand, at any time *k*, the state vector $\mathbf{x}_k$, is in essence a *concatenation in state space*.

Copyright © 1997-2006: Ike's Research Ltd

2. Programming

2.1 Coding GENO Program Files

The GAUSS program files for GENO have an '.e' filename extension. They are fairly standard in design and the user need not alter these except perhaps to specify different file paths. The '.e' files are located in the folder called 'Examples'. A typical GAUSS program file for GENO is depicted below.

Program 2.1: A Typical GENO Program

```
new;
#linesoff
   use geno;
   #include ../source/static_opt/sc_sp1.gep;
   load input_data[] = ../data/sc_sp1.dat;
   create sol_fp = ../output/sc_sp1.out with sc, horizon, 8;
   output file = ../output/sc_sp1.txt reset;
   outwidth 250;
   call geno_main;
   output off;
   fp = close(sol_fp);
end;
```

2.2 Coding GENO Data Files: Static Optimisation Problems

GENO input data files have a '.dat' filename extension and are located in folders labelled 'Data'; they provide specific numbers required to solve a given problem. The creation of a GENO data file is perhaps best explained by example. Consider the problem:[7]

Example 2.1: $\min_{\mathbf{x}} J(\mathbf{x}) = 5.3578547x_3^2 + 0.8356891x_1x_5 + 37.293239x_1 - 40792.141$

Subject to:

$$0 \le 85.334407 + 0.0056858x_2x_5 + 0.0006262x_1x_4 - 0.0022053x_3x_5 \le 92$$

$$90 \le 80.51249 + 0.0071317x_2x_5 + 0.0029955x_1x_2 + 0.0021813x_3^2 \le 110$$

$$20 \le 9.300961 + 0.0047026x_3x_5 + 0.0012547x_1x_3 - 0.0019085x_3x_4 \le 25$$

$$x_1 \in [78, 102];\ x_2 \in [33, 45];\ x_3 \in [27, 45];\ x_4 \in [27, 45];\ x_5 \in [27, 45]$$

This example has five variables. In order to use GENO, one first assumes that the variables are components of a five-dimensional state vector driven by a process that evolves over a *single* time period, starting from the origin $\mathbf{x} = \mathbf{0}$, according to the discrete-time dynamic equation:

$$x_{k+1}^i = A_k^i(\bar{\mathbf{x}}_k, \bar{\mathbf{u}}_{k-1}) + B_k^i(\bar{\mathbf{x}}_k, \bar{\mathbf{u}}_{k-1}) \cdot u_k^i$$

In this particular case (and in most cases of static optimisation) the functions A and B are both identity matrices. And because of this, the bounds for the state and control variables are the same. The control vector is selected at $t = 1$, but the actual value of interest is the state vector that obtains at $t = 2$.[8] Suppose further that the variable X2 is restricted to integer multiples of 0.34, then the data set supplied to GENO would in this case be as follows:

[7] **Source**: Hock and Schittkowski (1981, p.102).

[8] However, the optimal solution will occasionally be stated without the time subscript in this Manual.

Copyright © 1997-2006: Ike's Research Ltd

Table 2.1: Input Data for **Example** 1

VARIABLE NAME		X1	X2	X3	X4	X5
UCB		102	45	45	45	45
LCB		78	33	27	27	27
USB		102	45	45	45	45
LSB		78	33	27	27	27
Initial State Vector		0	0	0	0	0
Final State Vector		0	0	0	0	0
Discrete Values		0	0.34	0	0	0
Inter-connexion Matrix	X1	1	0	0	0	0
	X2	0	1	0	0	0
	X3	0	0	1	0	0
	X4	0	0	0	1	0
	X5	0	0	0	0	1

Legend:

UCB — ***U***pper ***C***ontrol ***B***ound

LCB — ***L***ower ***C***ontrol ***B***ound

USB — ***U***pper ***S***tate ***B***ound

LSB — ***L***ower ***S***tate ***B***ound

2.3 Coding GENO Data Files: Dynamic Optimisation Problems

If a given problem is in fact actually dynamic in character, the input data file would have the same structure as that shown above except for:

a) possibly some non-zero off-diagonal entries in the Inter-connexion Matrix wherever one variable appears in the dynamic equation of another

b) possibly some non-zero entries in the Final State Vector row for those components of the state vector that are required to remain fixed at some value at $t = T+1$, the optimisation horizon

c) possibly some non-zero entries in the Initial State Vector row

The table below is the actual data for a dynamic optimisation problem that is featured in this document.[9]

[9] See Example 3.13 below.

Copyright © 1997-2006: Ike's Research Ltd

Table 2.2: Input Data for a Dynamic Optimisation Problem

VARIABLE NAME		X1	X2	X3	X4	X5	X6	X7	X8	X9
UCB		10	10	1	1	1e20	100	5	1	1e20
LCB		-10	-10	-1	-1	0	-100	0	-1	0
USB		1e20	1e20	2	2	1e20	1e20	8	3	1e20
LSB		-1e20	0	-2	-2	0	-1e20	0	0	-1e20
Initial State Vector		0	0	1	2	1	100	8	2	100
Final State Vector		0	0	0	0	0	0	0	0	100
Discrete Values		0	0	1	1	0	0	0	0	0
Inter-connexion Matrix	X1	0	1	0	0	0	0	0	0	0
	X2	1	1	0	0	0	0	0	0	0
	X3	0	0	1	1	0	0	0	0	0
	X4	0	0	1	1	0	0	0	0	0
	X5	0	0	0	0	1	0	0	0	0
	X6	0	0	0	0	0	1	0	0	0
	X7	0	0	0	0	0	0	1	0	0
	X8	0	0	0	0	0	0	0	1	0
	X9	0	0	0	0	0	0	0	0	1

Notes:

1. The state vector has nine dimensions; unlike in the static case, it's initial value is not necessarily zero
2. This is a two-point boundary value problem since the variable X9 is required to be 100 at the final time
3. The is a mixed-integer problem since the variables X3 and X4 are restricted to integer values
4. The dynamic equation driving X1 does not contain the variable X1 itself; it is partially governed by X2
5. The dynamic equation driving X2 it is partially governed by X1 and X2
6. The dynamic equation driving X3 it is partially governed by X3 and X4
7. The dynamic equation driving X4 it is partially governed by X4 and X3
8. The rest of the state vector dynamics are independent
9. The inter-connexion matrix encapsulates the mutual influences of the dynamic constraints

2.4 Coding the Main Source File

The main source file in a given problem has a '.gep' filename extension (for ***G***AUSS ***E***xample ***P***roblem). The gep files are in folders labelled 'dynamic_opt' and 'static_opt' in the case of GENO; and these folders are themselves contained within a folder called 'source'. The gep files are what the user edits and supplies to GENO. They are fairly standard in design for any given problem-type. A typical a gep file for GENO based on Example 2.1 above is shown below and explanatory notes of the various symbols are appended.

Note that most of the symbols labelled as 'default' need not appear explicitly in the .gep file as depicted below. To avoid clutter and thus improve readability of the source code, most of these parameters are pre-set and "hidden away" in the accompanying `static_gep_defaults.src` or `dynamic_gep_defaults.src` file that is #include'd near the top of the .gep file. The user only needs to override the default values that specifically pertain to the problem at hand.

Copyright © 1997-2006: Ike's Research Ltd

```
// A constrained uni-objective static optimization problem
// Source: Hock and Schittkowski (1981, p.102).

#definecs p_maxgens  500                                  // Problem-specific
#definecs p_popsize  30                                   // Problem-specific
#definecs p_agents   1                                    // Problem-specific
#definecs p_order    5                                    // Problem-specific
#definecs p_plan     1                                    // Problem-specific

#include static_gep_defaults.src                          // Problem-specific

let vars[p_agents, p_order]   = 1   1   1   1   1;        // Problem-specific

adj_mode             = "s";                               // Problem-specific
solution_type        = "e";                               // Problem-specific
maximise             = false;                             // Problem-specific
pos_orth             = false;                             // Problem-specific

view_vars            = false;                             // Optional
constraints_check    = false;                             // Optional
vdu_output           = false;                             // Optional
timer                = false;                             // Optional

p_s_xover            = 0.55;                              // Default
p_a_xover            = 0.55;                              // Default
p_b_xover            = 0.00;                              // Default
p_h_xover            = 0.55;                              // Default
p_d_xover            = 0.55;                              // Default
p_shuffle            = 0.00;                              // Default

d_factor             = 0.80;                              // Problem-specific
quantum_0            = 0.1;                               // Problem-specific
rand_seed            = 240657;                            // Problem-specific


proc (1) = m_rate(i,d); retp(0.05); endp;                 // Default


proc (1) = bm_rate(d); retp(0.005); endp;                 // Default


proc (2) = f(i,d,v_array);                                // Problem-specific
local c,fv,u,x,z;
   u = matinit(order,plan,0);
   x = matinit(order,horizon,0);
  {u,x} = assign_sequences(i,d,u,x);[10]

   //evaluate constraints
   c = constraints(0,x,horizon);
   v_array = evaluate_constraints(c, v_array);

   //evaluate objective
   fv = objective(0,x,horizon);
   retp (fv,v_array);
endp;


proc (1) = objective(z,x,k);                              // Problem-specific
local fv;
   fv = 5.3578547*x[3,k]*x[3,k] + 0.8356891*x[1,k]*x[5,k] + 37.293239*x[1,k] - 40792.141;
   if(maximise); fv = fv; else; fv = -fv; endif;
   retp(fv);
endp;
```

[10] The procedure **proc** `assign_sequences(i,d,u,x)` is internal to GENO. It assigns to the local matrices x and u respectively whole "time" sequences of those components of the state and control vectors that belong to the *i*-th chromosome of the genetic population, and are controlled by the *d*-th optimising agent (i.e. they are associated with the *d*-th component of a multi-objective problem).

Copyright © 1997-2006: Ike's Research Ltd

```
proc (1) = constraints(z,x,k);                                  // Problem-specific
local c; c = zeros(6,1);
   c[1] =  (85.334407 + 0.0056858*x[2,k]*x[5,k] + 0.0006262*x[1,k]*x[4,k] - \
            0.0022053*x[3,k]*x[5,k] - 92);
   c[2] = -(85.334407 + 0.0056858*x[2,k]*x[5,k] + 0.0006262*x[1,k]*x[4,k] - \
            0.0022053*x[3,k]*x[5,k]);
   c[3] =  (80.512490 + 0.0071317*x[2,k]*x[5,k] + 0.0029955*x[1,k]*x[2,k] + \
            0.0021813*x[3,k]*x[3,k] - 110);
   c[4] = -(80.512490 + 0.0071317*x[2,k]*x[5,k] + 0.0029955*x[1,k]*x[2,k] + \
            0.0021813*x[3,k]*x[3,k] - 90);
   c[5] =  (9.3009610 + 0.0047026*x[3,k]*x[5,k] + 0.0012547*x[1,k]*x[3,k] + \
            0.0019085*x[3,k]*x[4,k] - 25);
   c[6] = -(9.3009610 + 0.0047026*x[3,k]*x[5,k] + 0.0012547*x[1,k]*x[3,k] + \
            0.0019085*x[3,k]*x[4,k] - 20);
   retp(c);
endp;
```

Points to Note: The gep Source File

1. `p_maxgens` — A matrix variable: it defines the maximum number of evolutionary generations. This constant is obviously problem-dependant since some problems need more evolutionary operations than others to produce the same quality of solution

2. `p_mingens` — A matrix variable: it defines the maximum number of "mini" evolutionary generations. Experience indicates that '2' is an efficient value for this parameter on most problems. To minimise clutter, this is located in the `static_gep_defaults.src` source file (see below).

3. `p_popsize` — A matrix variable: it defines the size of the evolutionary population. For the same reason as in Note 1 above, this parameter is problem-dependant: numerical experiments on a variety of problems seem to suggest that the optimal population size lies in the 20 – 30 range for most problems

4. `p_agents` — A matrix variable: it defines the number agents, or the number objectives in MP1

5. `p_order` — A matrix variable: it defines the size of the state vector, i.e. the number of variables in the problem

6. `p_plan` — A matrix variable: it defines the period over which the *control* vector is defined. It follows that the state vector is defined on `horizon = p_plan + 1`. On static optimisation problems, `p_plan = 1`.

7. `p_eqms` — A constant equal to 1.[11]

8. `static_gep_defaults.src` — A source file: it contains default declarations of global constants and procedures that are common to all static optimisation problems. The symbols are placed there for the sake of code clarity and may need to be re-defined within the gep file depending on the problem at hand.[12]

9. `vars` — A binary matrix variable: it is an 'incidence matrix' that shows what variables are directly controlled by each agent; equivalently, what variables are in each sub-problem of the multi-agent M-program.

10. `adj_mode` — A character variable: it specifies the adjustment mode of the GENO search procedure. The available options are:

    `"s"` — for singly-rational adjustment mode

    `"g"` — for group-rational adjustment mode

    This parameter should be set to `"s"` if the type of solution sought is the Nash equilibrium point, and to `"g"` for all other types of solutions.

    Note: The Nash equilibrium solution corresponds to the usual global maximum or minimum on uni-objective (i.e. single-agent) optimisation problems; hence one sets `adj_mode = "s"` on these problems.

---

[11] This parameter is a legacy of the original GENO design. It is located in the source file `static_gep_defaults.src` (or its dynamic equivalent: see next footnote) in order to minimise clutter in the gep file.

[12] Another similarly designed source file is `dynamic_geno_constants.src`

Copyright © 1997-2006: Ike's Research Ltd

11. `solution_type` — A character variable: it defines the type of solution sought. The available options are:

    e — for a Nash equilibrium solution

    p — for a Pareto-efficient solution

    i — for a Euclidian compromise solution

    m — for a Nash-Pareto mixed solution

12. `sequential_solve` — The parameter `sequential_solve` (not shown above) is a Boolean-valued variable normally associated with `adj_mode` and `solution_type` above. It is a legacy of GENO's original design, and it is located in the source file `static_gep_defaults.src` with a default value false. It should be re-set to true when the type of solution sought requires the adjustment mode, `adj_mode`, to be `"g"` and the solution type, `solution_type`, to be `"m"`.

13. `maximise` — A Boolean variable: it specifies whether the object is to maximise (maximise = true) or minimise (maximise = false) a function

14. `pos_orth` — A Boolean variable: it specifies whether the constraints restrict the solution to the positive orthant (pos_orth = true) or the negative orthant (pos_orth = false)

15. `view_vars` — A Boolean variable: it determines whether the algorithm's evolutionary progress is reported in variables values (view_vars = true) or objective function values (view_vars = false)

16. `constraints_check` — A Boolean variable: it allows for a check on the values of the constraints at the solution

17. `vdu_output` — A Boolean variable: it allows one to switch off output to the vdu; the default is vdu_output = true

18. `timer` — A Boolean variable: it allows one to switch on the loop timer; the default is timer = false

19. `p_u_xover` — A matrix variable: it sets the probability of applying the uniform crossover operator

20. `p_s_xover` — A matrix variable: it sets the probability of applying the simple crossover operator

21. `p_a_xover` — A matrix variable: it sets the probability of applying the arithmetic crossover operator

22. `p_b_xover` — A matrix variable: it sets the probability of applying the boundary crossover operator

23. `p_h_xover` — A matrix variable: it sets the probability of applying the heuristic crossover operator

24. `p_d_xover` — A matrix variable: it sets the probability of applying the differential crossover operator

25. `p_shuffle` — A matrix variable: it sets the probability of shuffling the population of chromosomes

26. `d_factor` — A matrix variable: it defines a direction factor for the differential crossover operator (see Appendix I).

27. `quantum_0` — A matrix variable: it sets the initial size of quanta for GENO's quantisation scheme In selecting this parameter, the object should be to ensure that the initial population is sufficiently varied along each dimension of the problem. A choice of the smaller between `quantum_0` = 0.1 and 10% of the smallest variable range is normally efficient. But, if one seeks an *integer* solution, then one sets: `quantum_0` = 1.

28. `rand_seed` — A matrix variable: it defines a seed value for the random number generator

29. `m_rate` — A procedure: it returns the probability of mutation

30. `bm_rate` — A procedure: it returns the probability of boundary mutation

31. `f(i,d,v_array)` — A procedure: it specifies the method of solution

32. `objective` — A procedure: it specifies the objective function under study

33. `constraints` — A procedure: it specifies the functional constraints

Copyright © 1997-2006: Ike's Research Ltd

2.5 Program Output

The results of a successful GENO execution are available in four ways: (1) as printed output to the VDU; (2) as printed output to a text file; (3) as printed output to a GAUSS/GENO data file; (4) as a global variable called `sol_mtx` (for solution matrix). The full set of outputs (i.e. when all options are turned on) is shown below; a set of explanatory notes follow the listing.

Output List 2.1: GENO output

```
A. GENO Parameters
   ===============
   optimisation period:                      1
   adjustment mechanism:                     s
   solution type:                            e

   population size:                          20
   random seed:                              240657
   initial quanta:                           0.100000

   mutation rate:                            0.050000
   boundary mutation rate:                   0.005000

   probability of simple crossover:          0.550000
   probability of arithmetic crossover:      0.550000
   probability of boundary crossover:        0.000000
   probability of heuristic crossover:       0.550000
   probability of differential crossover:    0.550000
   probability of shuffling population:      0.000000

   differential operator factor:             0.800000
   ---------------------------------------------------


B. GENO Evolution
   ==============

   Generation
   Number                 Objective1           Objective2

     0                2571734.743855            48.019520
    20                    685.227954             0.000000
    40                    682.883463             0.000000
    50                    682.807954             0.000000
    60                    680.814853             0.000000
    70                    680.787579             0.000000
    80                    680.702849             0.000000
    90                    680.692505             0.000000
   100                    680.654102             0.000000
   110                    680.651367             0.000000
   120                    680.645037             0.000000
   130                    680.639011             0.000000
   140                    680.638547             0.000000
   150                    680.637505             0.000000
   160                    680.631026             0.000000
   170                    680.630309             0.000000
   180                    680.630139             0.000000
   190                    680.630115             0.000000
   200                    680.630094             0.000000
   210                    680.630065             0.000000
   220                    680.630058             0.000000
   230                    680.630057             0.000000
   240                    680.630057             0.000000
   250                    680.630057             0.000000
   260                    680.630057             0.000000
   270                    680.630057             0.000000
   280                    680.630057             0.000000
   290                    680.630057             0.000000
   300                    680.630057             0.000000

   --------------------------------
C. Loop Time:  95.708000 seconds
   --------------------------------
```


Copyright © 1997-2006: Ike's Research Ltd

D. GENO Optimal Solution
==========

Best Vectors for Agent 1:

Control1:	2.330499	
Control2:	1.951372	
Control3:	4.365727	
Control4:	-0.624487	
Control5:	1.594227	
State1:	0.000000	2.330499
State2:	0.000000	1.951372
State3:	0.000000	4.365727
State4:	0.000000	-0.624487
State5:	0.000000	1.594227

Best Function Value: 680.630057

Best Vectors for Agent 2:

Control6:	0.000000	
Control7:	0.000000	
State6:	0.000000	0.000000
State7:	0.000000	0.000000

Best Function Value: 0.000000

E. Equations Vector at Solution
==========

-0.477541
1.038131

F. Inequalities Vector at Solution
==========

252.561724
144.878178

G. Contents of Solution Matrix
==========

2.330499	0.000000
1.951372	0.000000
4.365727	0.000000
-0.624487	0.000000
1.594227	0.000000
0.000000	2.330499
0.000000	1.951372
0.000000	4.365727
0.000000	-0.624487
0.000000	1.594227
680.630057	680.630057
0.000000	0.000000
0.000000	0.000000
0.000000	0.000000
0.000000	0.000000
0.000000	0.000000

Copyright © 1997-2006: Ike's Research Ltd

Points to Note: GENO Output

1. `Viewing the Output` — The VDU output is controlled by the parameter `vdu_output` whose default value is `true`; this output may be turned off by including the assignment `vdu_output = false` in the relevant gep file

2. `Timing Program Run` — The evolution loop timer is controlled by the parameter `timer` whose default value is `false`; the timer may be turned on by including the assignment `timer = true` in the relevant gep file

3. `Checking Feasibility` — GENO allows the user to check the feasibility of the computed solution via the parameter called `constraint_check` whose default value is `false`; the values of the various constraints at the solution may be viewed by including the assignment statement `constraint_check = true` in the relevant gep file

4. `Program Output File` — GENO's output is directed to a text file with same name as the program but with a '.txt' filename extension. This file is located in the folder called 'output' that is under the geno directory. The text file output cannot be turned off.

5. GAUSS / GENO `Data File` — Note that the main program (see **Program** 2.1, p.2) includes the following lines of code:

```
create sol_fp = ../output/prog_name.out with sc, horizon, 8;
fp = close(sol_fp);
```

   These instructions create and open a GAUSS data file called `prog_name.out` to which GENO's output written, and close the said data file at the end of the run. The file pointer is `sol_fp` and this file itself is located in the folder called 'output' under the geno directory. The GAUSS data file output cannot be turned off.

6. `The Solution Matrix` — GENO's output is also temporarily stored in a global variable called `sol_mtx` until the next execution of any GENO program. The contents of `sol_mtx` may be viewed by including the assignment `sol_mtx_check = true` in the relevant gep file; or by simply typing 'sol_mtx' at the GAUSS prompt after the end of the program run.

7. `Output Matrix Format` — The global variable `sol_mtx` is a compact matrix whose first dimension (# rows) depends on the number of agents and the variables they each control; the second dimension (# columns) is always the optimisation horizon, which is given by the parameter `horizon = p_plan + 1`. The variable `sol_mtx` contains the figures listed under Section B of the output list in the same order; the data layout in the GAUSS/GENO Data File is the same. Table 2.3 below shows the layout of the data; it includes an extra header row and two header columns for explanatory purposes only.

Table 2.3: Format of `sol_mtx` and the GAUSS / GENO Data File

Control Application Time:		$t = 1$	$t = 2$
Agent 1:	Control1:	2.330499	0.000000
	Control2:	1.951372	0.000000
	Control3:	4.365727	0.000000
	Control4:	-0.624487	0.000000
	Control5:	1.594227	0.000000
	State1:	0.000000	2.330499
	State2:	0.000000	1.951372
	State3:	0.000000	4.365727
	State4:	0.000000	-0.624487
	State5:	0.000000	1.594227
	Best Function Value:	680.630057	680.630057
Agent 2:	Control6:	0.000000	0.000000
	Control7:	0.000000	0.000000
	State6:	0.000000	0.000000
	State7:	0.000000	0.000000
	Best Function Value:	0.000000	0.000000

Copyright © 1997-2006: Ike's Research Ltd

3. Worked Examples

3.1 Preamble

The purpose of this section is to present a sample of fully worked examples that further illustrate the preparation of gep and dat files, and that sufficiently expose the capabilities of GENO. As regards the latter, the emphasis is on the quality of the final solution found. But although assessing GENO's efficiency was not explicitly part of the exercise, the algorithm's performance in this respect is mentioned where necessary. Section 3.4 presents outlines of some problem pre-processing that may be necessary. This is followed by descriptions of the examples themselves using a set pattern: first an "Executive Summary" is presented comprising (a) a formal mathematical statement of the problem (b) a listing of GENO's output and (c) some pertinent remarks. This is followed by a listing of the actual gep file and a table of the dat file for each particular example. Each presentation ends with a list of pertinent points explaining some aspects of the gep and dat files.

3.2 Scope

The examples presented are as follows:

- Examples 3.1 and 3.2 are uni-objective, unconstrained, non-linear, static optimisation problems. They demonstrate the easy of programming GENO, and its effectiveness for this class of problem.
- Examples 3.3 through 3.7 are uni-objective, constrained, non-linear, static optimisation problems. The first example is principally meant to illustrate how equality constraints can be dealt with; the second example shows how to program GENO for an mixed-integer solution;[13] the third example illustrates how the parameter `q_mod` may be used to improve the algorithm's performance by accounting for differences in variable ranges; and the fourth and fifth examples describe an alternative method of treating inequality constraints that is efficient if the constraints are actually active at the solution.
- Examples 3.8 and 3.9 are uni-objective, constrained, non-linear, dynamic optimisation problems. The former is a two-point boundary value problem; the latter further illustrates GENO's ability on mixed-integer optimisation problems.
- Example 3.10 is a popular unconstrained, multi-objective, non-linear, static optimisation problem that has been included so as to afford a comparison of GENO's Nash equilibrium solution to a benchmark, as well as to other solutions computed by different techniques.
- Examples 3.11 through 3.14 serve to further illustrate GENO's ability on multi-objective optimisation problems. The first two examples are artificial bi-objective static optimisation problems solved for the Euclidean compromise solution (or ECS); the third example is a practical application of GENO's ECS method to the Markowitz's portfolio selection problem; and the last example is a comprehensive multi-objective dynamic optimisation problem solved for the Nash equilibrium solution.
- Examples 3.15 through 3.20 serve to illustrate how GENO may be used to solve nonlinear mixed-integer and generalized disjunctive programming problems.

3.3 Default Parameter Values

GENO's parameters were first pre-set to the values shown below. These values proved to be efficient in most cases. Varying the Direction Factor usually results in a significant change in performance.

Population Size:	10 - 30
Probability of Mutation:	0.05
Probability of Simple Crossover:	0.55
Probability of Arithmetic Crossover:	0.55
Probability of Boundary Crossover:	0.00
Probability of Heuristic Crossover:	0.55
Probability of Differential Crossover:	0.55
Probability of Shuffle:	0.00 – 0.55
Direction Factor [for Differential Crossover]:	0.15 - 0.80

[13] This example is a "mixed integer" problem in that the variables X1 and X2 are required to be integer multiples of 0.0625

Copyright © 1997-2006: Ike's Research Ltd

3.4 Problem Preparation

GENO requires the problem to be presented in one of several formats as described below.

3.4.1 The Basic Approach

Consider the following mathematical program that is defined on the set K_1:

MP: $\min_{\mathbf{x}} f(\mathbf{x})$

$$\text{Subject to:} \quad c(\mathbf{x}) \geq 0$$
$$s(\mathbf{x}) = 0$$
$$\mathbf{x} \in K_1 \subseteq \mathbf{R}^n$$

In the absence of the equality constraint $s(\mathbf{x}) = 0$, MP would be coded directly by simply writing out the procedures `constraints` and `objective` that implement the functions c and f respectively. But, if the constraint s is present then, in addition to `constraints` and `objective`, one to code s into appropriate GAUSS procedures. If s is dynamic, it should be manipulated so as to identify the coefficient expressions A and B of MP1 and these may then be coded in a straight forward manner. On the other hand, if the constraint s is static but not of the form stipulated by G of MP1, then one should eliminate it, and the easiest way to do this is to introduce an equivalent pair of inequalities as follows:

MPa: $\min_{\mathbf{x}} f(\mathbf{x})$

$$\text{Subject to:} \quad c(\mathbf{x}) \geq 0$$
$$s(\mathbf{x}) \geq 0$$
$$-s(\mathbf{x}) \geq 0$$
$$\mathbf{x} \in K_1 \subseteq \mathbf{R}^n$$

M-program MPa is now of the required form[14] and it may be coded for solution as a uni-objective optimisation problem, or as a bi-objective problem that separates the constraints satisfaction part of the solution procedure at the expense of speed. Of the examples listed in the index of GENO programs provided in Appendix II, the names of those coded using the first approach are post-fixed with the letter 'a'; those using the bi-objective method are post-fixed with the letters 'aa'.

3.4.2 The NCP Approach

The method outlined above is easy to implement, but much depends on the structure of the feasible region circumscribed by the constraints. For problems whose solutions lie on several boundaries of the feasible region, tuning the GENO for a good solution may be difficult. However, numerical experience appears to suggest that the quality of the solution can generally be improved upon (albeit at the expense of speed) by converting an actively-constrained mathematical program into a bi-objective optimisation problem as explained below.

Definitions

Given an n-dimensional set $K \subseteq \mathbf{R}^n$ and a mapping $\mathbf{F} : K \to \mathbf{R}^n$, the *Variational Inequality Problem*, denoted by VI(K, $\mathbf{F}$), is defined as follows:

- Find the n-vector $\mathbf{x}^*$ in K such that:[15] $\langle \mathbf{F}(\mathbf{x}^*), (\mathbf{y} - \mathbf{x}^*) \rangle \geq 0, \quad \forall \mathbf{y} \in K$

In the special case when K is restricted to the non-negative orthant, i.e. when $K \subseteq \mathbf{R}^n_+$, then VI($K$, $\mathbf{F}$) is equivalent to the *Nonlinear Complementarity Problem* denoted by NCP($\mathbf{F}$):

- Find the n-vector $\mathbf{x}^*$ in K such that: $\mathbf{F}(\mathbf{x}^*) \geq 0, \quad \mathbf{x}^* \geq 0, \quad \langle \mathbf{F}(\mathbf{x}^*), \mathbf{x}^* \rangle = 0$

[14] Needless to say, the original problem can be one of maximisation, and the inequalities may also be of the 'less than' type.

[15] The notation ⟨a, b⟩ denotes the inner product $a^T b$ of the vectors 'a' and 'b'.

Copyright © 1997-2006: Ike's Research Ltd

Associated with NCP(**F**) are three index sets of interest:

1. $\alpha \equiv \{i : x_i > 0\}$
2. $\beta \equiv \{i : F_i(\mathbf{x}) > 0\}$
3. $\gamma \equiv \{i : x_i = 0 = F_i(\mathbf{x})\}$

The solution is said to be *non-degenerate* if the index set is empty, i.e. $\gamma = \varnothing$.

There is a certain type of bivariate function normally associated with NCP(**F**) which is usually referred to by the term 'NCP-function'. An *NCP- function* is a real-valued mapping defined as follows:

- $\phi : \mathbf{R}^2 \to \mathbf{R}$ and ϕ is such that: $\phi(a, b) = 0 \quad \Leftrightarrow \quad a \geq 0, \quad b \geq 0, \quad ab = 0$.

NCP-functions are the subject of much current research but our interest here is more utilitarian than theoretical. To that end, the following properties are pertinent.

Given any NCP-function ϕ, define the *n*-vector Φ as follows:

- $\Phi : \mathbf{R}^n \to \mathbf{R}$ such that: $\Phi(\mathbf{x}) = \text{vec}\{\phi(x_i, F_i(\mathbf{x})\}$.[16]

Clearly $\mathbf{x}^* \in \mathbf{R}^n$ is a solution to NCP(**F**) if and only if it solves the system of equations $\Phi(\mathbf{x}) = \mathbf{0}$.

Alternatively, one may consider a suitably defined function Ψ which provides a measure of the degree of coincidence between the solution set of NCP(**F**) and that of the M-program $\min_{\mathbf{x}} \Psi(\mathbf{x})$.

For obvious reasons, the function Ψ is normally called a *merit function*; and for any given NCP-function, there is a natural merit function given by

- $\Psi : \mathbf{R}^n \to \mathbf{R}$ and $\Psi(\mathbf{x}) = \Phi(\mathbf{x})^T \Phi(\mathbf{x})$.

GENO's NCP method employs a natural merit function (denoted by *h*) based on an NCP-function attributed to Fischer and Burmeister (see Sun and Qi: 2000).

Auxiliary M-program Formulation

There are several numerical techniques for solving VI(*K*, **F**) or its variant NCP(**F**), each with its own merits and drawbacks. An up-to-date (but brief) discussion of algorithms that could be applied may be found in Aghassi, et al. (2005). Of the available methods, the ones that are of interest here are those that re-formulate the original problem — expressed as VI(*K*, **F**) or NCP(**F**) — into an appropriately defined mathematical program. The main motivation for adopting these techniques is the demonstrable efficiency of GENO on multi-objective optimisation problems (see Examples 3.10 – 3.14); and the challenge is to formulate an auxiliary optimisation problem that is somehow associated with the constraints of the original problem and, in some manner, aids the search for the optimal solution, **x***. Such an auxiliary M-program may be derived as follows.

- Assume the problem at hand is of the form MP1a, i.e. constrained by (say *m*) inequalities only. Introduce a positive *m*-vector of slack variables ω to turn $c(\mathbf{x}) \geq \mathbf{0}$ into $c(\mathbf{x}) - \omega = \mathbf{0}$.
- But we know that at *any feasible* point $(\mathbf{x}, \omega)$ in the $(\mathbf{x}, \omega)$-space, the constraint $c(\mathbf{x}) - \omega = \mathbf{0}$ is, by definition, satisfied; and so is $[c(\mathbf{x}) - \omega]^T[c(\mathbf{x}) - \omega] = 0$, and therefore one can chose to adopt the latter restriction without affecting the original problem.
- Thus instead of directly solving MP1a, one can alternatively consider the extended M-program:

MPω: $\min_{\mathbf{x},\omega} f(\mathbf{x})$

Subject to: $[c(\mathbf{x}) - \omega]^T[c(\mathbf{x}) - \omega] = 0$;

$\mathbf{x} \in K_1 \subseteq \mathbf{R}^n$;

$\omega \in K_2 \subseteq \mathbf{R}^m_+$.

[16] The notation vec$\{z_i\}$ denotes a vector whose *i*-th element is z_i.

Copyright © 1997-2006: Ike's Research Ltd

- The solution process of any M-program may be considered as implicitly comprising two stages: (a) a delineation of the feasible set; (b) a search for those *particular* points in the feasible set that optimise the objective function. In practice these stages are codified as *necessary* and *sufficient* conditions that must be satisfied by any candidate solution.

- The constraints on ω in the M-program MPω point towards employing an NCP to solve the constraint satisfaction part of the solution process. To see this, let the caret '^' denote a variable that is "temporarily fixed" and define the following *m*-vector:

$$\mathbf{F}(\omega) = \text{vec}\{(\hat{c}_i - \omega_i)^2\}\,; \qquad i = \{1, 2, \cdots, m\}\,.$$

And since $[c(\mathbf{x}) - \omega]^T[c(\mathbf{x}) - \omega] = 0$ at the solution $(\mathbf{x}^*, \omega^*)$, it follows that $(\omega^*)^T\,\mathbf{F}(\omega^*) = \mathbf{0}$. Therefore, relative to the quasi-constant $\hat{c}$ (which can be assumed to evolve towards its optimal value $\hat{c} = c(\mathbf{x}^*)$ via a separate search process), the search for ω* in MPω may be stated thus:

NCPω: Find ω* such that: $\omega^* \geq 0$; $\mathbf{F}(\omega^*) \geq \mathbf{0}$; $(\omega^*)^T\,\mathbf{F}(\omega^*) = \mathbf{0}$.

- As mentioned above, one way of solving NCPω is to turn it into a *minimisation* problem in ω based to some *merit function* $h(\omega)$. By this device, solving MPω is decomposed into two inter-linked processes: (a) a search for $\mathbf{x}^*$ using the original criterion (and constraint) and (b) a search for ω* using the merit function. For the latter, the method Faccenei and Soares (1997) is well suited and easy to implement. The Facchenei-Soares the merit function $h(\omega)$ is defined as

$$h(\omega) = \sum_{i=1}^{n} \phi\big(\omega_i, F_i(\omega)\big)^2$$

where the function ϕ is the Fischer-Burmeister NCP-function is defined by

$$\phi(a, b) = \sqrt{a^2 + b^2} - (a + b)\,.$$

Note: The Fischer-Burmeister NCP-function is non-positive on the positive orthant $\mathbf{R}^2_+$, only attaining its maximum value of 0, when either $a = 0$ or $b = 0$, or when $ab = 0$, i.e. at the solution of the generic NCP

NCP: Find 'a' and 'b' such that: $\text{vec}\{a\} \geq \mathbf{0}$; $\text{vec}\{b\} \geq \mathbf{0}$; $\text{vec}\{a\}^T\text{vec}\{b\} = 0$.

- In view of the properties of the Fischer-Burmeister NCP-function noted above, it follows immediately that the Facchenei-Soares merit function $h(\omega)$ has a *minimum value* of 0, and that any algorithm that optimises this towards its minimum value also solves NCPω and thus contributes towards the solution of the extended M-program MPω.

Implementation

The process decomposition explained above means that the extended M-program MPω may be re-formulated as a bi-objective optimisation problem which may be solved for the *Nash equilibrium solution* $(\mathbf{x}^*, \omega^*)$, thus:

MPb:

$$\min_{\mathbf{x}} f(\mathbf{x}) \qquad\qquad \min_{\omega} h(\omega)$$

$$\text{Subject to:} \quad c(\mathbf{x}) \geq 0 \qquad\qquad \text{Subject to:} \quad [\hat{c} - \omega]^T[\hat{c} - \omega] = 0$$

$$\mathbf{x} \in K_1 \subseteq \mathbf{R}^n \qquad\qquad \omega \in K_2 \subseteq \mathbf{R}^m_+$$

Note 1: The inequality constraint in the first program is retained to ensure feasibility of the final solution; the equality constraint in the second program is there to ensure that ω actually approaches the optimal solution $\omega^* = c(\mathbf{x}^*)$ and avoids the degenerate case $\omega = \mathbf{0}$.

Note 2: The introduction of an extra M-program in ω certainly slows down GENO, but this is mitigated by the fact that the auxiliary program only one constraint. Furthermore, we know that this constraint is active at the solution, and we can therefore take advantage of that fact in our design of algorithms. Numerical experience indicates that the bi-objective scheme is beneficial in the majority of cases; the auxiliary program has the effect of "nudging" the genetic population towards the optimal solution via $\hat{c} = c(\mathbf{x})$.

Among the examples programs listed in Appendix II, the names of those coded using this approach are post-fixed with the letter 'b'.

Copyright © 1997-2006: Ike's Research Ltd

3.4.3 The Elimination Approach I: Equality Constraints

By far, the most efficient way of treating the m-vector of equality constraints $s(\mathbf{x}) = 0$ is to "solve it" for particular components of $\mathbf{x}$, and then substitute these out of the M-program altogether. Assume this is possible for the whole constraint vector s; "solving for" m different components of the state vector leads to m equations, say $\mathbf{p} = q(\mathbf{x})$. Using the notation $\mathbf{x}_{-m}$ to signify the vector $\mathbf{x}$ but with m components removed, the modified M-program may be stated thus:

MPc: Given $\mathbf{p} \in K_3 \subseteq \mathbf{R}^m$ and $\mathbf{p} = q(\mathbf{x})$

$$\min_{\mathbf{x}} f(\mathbf{x}_{-m}, \mathbf{p})$$

$$\text{Subject to:} \quad c(\mathbf{x}_{-m}, \mathbf{p}) \geq 0\,; \qquad \mathbf{x}_{-m} \in K_3 \subseteq \mathbf{R}^{n-m}$$

The success of this approach much depends on the ingenuity of the user in carrying out the problem pre-processing mentioned above. Of examples programs listed in Appendix II, the names of those coded using this approach are post-fixed with the letter 'c'.

3.4.4 The Elimination Approach II: Inequality Constraints

In principle, the equation elimination technique described in § 3.4.3 may equally be applied to equality constraints that follow the introduction of ω—an m-vector of the slack variables. But unless the range of ω is limited, and its evolution controlled by some optimality condition, numerical experience shows that tuning the algorithm may be difficult. To circumvent this, GENO employs an auxiliary M-program using the following argument.

- Once ω is introduced, then f effectively becomes an implicit function of a *positive variable*. And assuming a *minimisation* problem (but without loss of generality), it easy to show that the following conditions hold at the optimal solution:

$$\omega \geq 0\,; \qquad \partial f/\partial\omega \geq 0\,; \qquad (\omega)^T(\partial f/\partial\omega) = 0\,.$$

- The corresponding conditions in the case of a maximisation problem are:

$$\omega \geq 0\,; \qquad \partial f/\partial\omega \leq 0\,; \qquad (\omega)^T(\partial f/\partial\omega) = 0\,.$$

- The above conditions constitute an NCP which suggests that the bi-objective approach outlined in § 3.4.2 may be used. However, in order to account for both maximisation and minimisation (as well as control the genetic search process), one adopts the following equivalent NCP instead:

$$\omega_i \geq 0\,; \qquad (\partial f/\partial\omega_i)^2 \geq 0\,; \qquad \omega_i(\partial f/\partial\omega_i)^2 = 0\,; \qquad i = \{1, 2, \cdots, m\}\,.$$

- Thus, an alternative formulation of the problem is the following pair of M-programs:

MPd: $\min_{\mathbf{x}} f(\mathbf{x}_{-\omega}, \omega)$ $\qquad\qquad$ $\min_{\omega} h(\omega)$

Subject to: $c(\mathbf{x}_{-\omega}, \omega) \geq 0$, $\mathbf{x}_{-\omega} \in K_3 \subseteq \mathbf{R}^{n-\omega}$ $\qquad\qquad$ Subject to: $\omega \in K_2 \subseteq \mathbf{R}^m_+$

in which the merit function $h(\omega)$ is defined as

$$h(\omega) = \sum_{i=1}^{n} \phi\left(\omega_i, (\partial f/\partial\omega_i)^2\right)^2$$

and the function ϕ is the Fischer-Burmeister NCP-function $\phi(a, b) = \sqrt{a^2 + b^2} - (a + b)$.

- This approach is can be rather cumbersome to implement because the gradient vector $\partial f/\partial\omega$ may not be that easy to derive due to inter-dependencies of the resulting equality constraints. Furthermore, this method seems to be most effective when the inequalities selected for elimination are actually active at the final solution. Unfortunately, one cannot determine a priori which constraints are active.

Among the examples listed in the index of GENO programs provided in Appendix II, the names of those coded using this approach are post-fixed with the letter 'd'.

Copyright © 1997-2006: Ike's Research Ltd

3.5 Some Practical Examples

Example 3.1: Static Unconstrained Optimisation: The Colville #4 Function[17]

$$\min_{\mathbf{x}} f(\mathbf{x}) = 100\left(x_2 - x_1^2\right)^2 + \left(1 - x_1\right)^2 + 90\left(x_4 - x_3^2\right)^2 + \left(1 - x_3^2\right)^2 + 10.1\left(\left(x_2 - 1\right)^2 + \left(x_4 - 1\right)^2\right) + 19.8\left(x_2 - 1\right)\left(x_4 - 1\right)$$

Subject to: $x_i \in [-10, 10],\ \forall i$

I. GENO Output

Generation	*Function Value*
0	9794.900000
10	0.000000
20	0.000000
40	0.000000
60	0.000000
80	0.000000
90	0.000000
100	0.000000

Optimal State Vector: [18] $\mathbf{x}$ = (1.000000, 1.000000, 1.000000, 1.000000)^T

Objective Function Value: $f(\mathbf{x})$ = 0.000000

II. General Remarks

The original source for this example is Colville (1968): it is a variation on a well known test problem called Rosenbrock's Banana Function.[19] The Colville #4 function has a banana-shaped valley in 4 dimensions as well as a dense Hessian matrix as opposed to the sparse Hessian of the 4-dimensional Rosenbrock function.[20] It is one of the functions which Michalewicz uses to demonstrate the effect of his implementation of the Heuristic crossover operator. The global minimum is 0.000, and it is located at (1, 1, 1, 1).

Although Michalewicz's GENECOP algorithm produces a very good approximate solution, namely:

$$\mathbf{x} = (1.000581,\ 1.001166,\ 0.999441,\ 0.998879)^T$$

with the corresponding function value 0.0000012, it requires at least 10,000 generations to reach this level of accuracy. By contrast, GENO converges to the true global solution, and in far fewer generations.

Source	Generations	Best Variable Value	Best Function Value	CPU Time
GENECOP	Over 10000	See above	0.0000012	-
GENO	10	See above	0.000000	-

[17] **Source**: Michalewicz (1994, p.157).

[18] The variables x_k, $\mathbf{x}_k$ and $\overline{\mathbf{x}}_k$ should not be confused: x_1 denotes the *k*th component of the state vector; $\mathbf{x}_k$ denotes the entire state vector evaluated at time t = k; and finally, $\overline{\mathbf{x}}_k$ is a matrix whose columns are the states vector at times t = 0, 1, 2, . . ., k. On static problems, the variable of interest is normally the final state vector $\mathbf{x}_T$.

[19] Rosenbrock, H. H. (1960). An Automatic Method for Finding the Greatest and Least Value of a Function. *Computer Journal*, **3**, pp175-184.

[20] Van Iwaarden, R. J. (1996). *An Improved Unconstrained Global Optimization Algorithm*. Unpublished PhD Thesis, University of Colorado, Denver, U.S.A.

Copyright © 1997-2006: Ike's Research Ltd

III. **The gep File**: Example 3.1

```
// An unconstrained uni-objective static optimization problem
// Source: Michalewicz (1994, p.157)

#definecs p_maxgens     200
#definecs p_popsize     40
#definecs p_agents      1
#definecs p_order       4
#definecs p_plan        1

#include static_gep_defaults.src

let vars[p_agents, p_order]   = 1   1  1  1;

adj_mode        = "s";
solution_type   = "e";
maximise        = false;

p_s_xover   = 0.00;
p_a_xover   = 0.55;
p_b_xover   = 0.00;
p_h_xover   = 0.55;
p_d_xover   = 0.55;
p_shuffle   = 0.00;

d_factor    = 0.80;
quantum_0   = 1.0;
rand_seed   = 240658;


proc (1) = m_rate(i,d); retp(0.05); endp;

proc (1) = bm_rate(d); retp(0); endp;


proc (2) = f(i,d,v_array);
local c,fv,u,x;
   u = matinit(order, plan, 0);
   x = matinit(order, horizon, 0);
  {u,x} = assign_sequences(i,d,u,x);[21]

   //evaluate objective
   fv = objective(x,horizon);
   retp (fv,v_array);
endp;


proc (1) = objective(x,k);
local fv;
   fv = 100*(x[2,k] - x[1,k]^2)^2 + (1 - x[1,k])^2 + 90*(x[4,k] - x[3,k]^2)^2 \
        + (1 - x[3,k])^2 + 10.1*((x[2,k] - 1)^2 + (x[4,k] - 1)^2) \
        + 19.8*(x[2,k] - 1)*(x[4,k] - 1);
   if (maximise); fv = fv; else; fv = -fv; endif;
   retp(fv);
endp;
```

[21] Infra page 5, footnote 10.

Copyright © 1997-2006: Ike's Research Ltd

VI. **The dat File**: Example 3.1

Table 3.1: Input Data for the Colville #4 Function

VARIABLE NAME		X1	X2	X3	X4
UCB		10	10	10	10
LCB		-10	-10	-10	-10
USB		10	10	10	10
LSB		-10	-10	-10	-10
Initial State Vector		0	0	0	0
Final State Vector		0	0	0	0
Discrete Values		0	0	0	0
Inter-connexion Matrix	X1	1	0	0	0
	X2	0	1	0	0
	X3	0	0	1	0
	X4	0	0	0	1

V. **Points to Note**: Example 3.1

1. `p_popsize` — This problem required a larger size of population than the default value of 30

2. `adj_mode` — This parameter should always be set to `"s"` for unconstrained optimisation problems

3. `solution_type` — This parameter should always be set to `"e"` for unconstrained optimisation problems

4. `boundary_mutation` — As with the parameter p_popsize, the most effective boundary mutation rate was the non-default value '0'

5. `f(i,d,v_array)` — For unconstrained optimisation problems the form of this procedure is standard as repeated below: it needn't be changed.

```
proc (2) = f(i,d,v_array);
local c,fv,u,x;
   u = matinit(order, plan, 0);
   x = matinit(order, horizon, 0);
  {u,x} = assign_sequences(i,d,u,x);[22]

   //evaluate objective
   fv = objective(x,horizon);
   retp (fv,v_array);
endp;
```

6. `tuning` — Recourse to a 'larger-than-normal' population size and the switching off the boundary mutation operator were required to effectively tune the algorithm in this case.

[22] Infra page 5, footnote 10.

Copyright © 1997-2006: Ike's Research Ltd

Example 3.2 Static Unconstrained Optimisation: The 2-Dimensional Rastrigin Function[23]

$$\min_{\mathbf{x}} f(\mathbf{x}) = x_1^2 + x_2^2 - \cos(18x_1) - \cos(18x_2)$$
$$\text{Subject to:} \quad x_1, x_2 \in [-1, 1]$$

I. GENO Output

Generation	*Function Value*
0	-1.544693
10	-2.000000
20	-2.000000
30	-2.000000
40	-2.000000
50	-2.000000

Optimal State Vector: [24] $\mathbf{x} = (0.00000, 0.00000)^T$

Objective Function Value: $f(\mathbf{x}) = -2.00000$

II. General Remarks [25]

This is a very popular test function in the Russian literature on global optimisation.[26] The global minimum is -2, and its location is (0, 0). There are about 50 local minima arranged in a lattice configuration.

The table below compares results obtained using various methods, and as can be seen, GENO converges to the global solution relatively quickly. It is reasonable to suggest that it would out-perform other algorithms of the same type in well-designed efficiency tests.

Source	Generations	Best Variable Value	Best Function Value	CPU Time
Koon and Sebald	2010	(0.00034, -0.00055)	-2.000	689
Torn and Žilinskas	295*	(0.08378, -0.00761)	-1.988	-
Cetin, et al.	77*	(0.00000, 0.00000)	-2.000	-
GENO	10	(0.00000, 0.00000)	-2.000	-

[23] **Source**: Torn and Žilinskas (1989).

[24] The variables x_k, $\mathbf{x}_k$ and $\overline{\mathbf{x}}_k$ should not be confused: x_l denotes the *k*th component of the state vector; $\mathbf{x}_k$ denotes the entire state vector evaluated at time t = k; and finally, $\overline{\mathbf{x}}_k$ is a matrix whose columns are the states vector at times t = 0, 1, 2, . . ., k. On static problems, the variable of interest is normally the final state vector $\mathbf{x}_T$.

[25] The algorithm of Cetin, *et al* is a deterministic method; that of Koon and Sebald is an evolutionary technique; whilst that of Torn-Žilinskas is a hybrid method.

[26] Torn and Žilinskas (1989, p.190).

* Number of function evaluations.

Copyright © 1997-2006: Ike's Research Ltd

III. **The gep File**: Example 3.2

```
// An unconstrained uni-objective static optimization problem
// Source: Torn and Žilinskas (1989)

#definecs p_maxgens     50
#definecs p_popsize     30
#definecs p_agents      1
#definecs p_order       2
#definecs p_plan        1

#include static_gep_defaults.src

let vars[p_agents, p_order]   = 1   1;

adj_mode        = "s";
solution_type   = "e";
maximise        = false;

p_s_xover   = 0.00;
p_a_xover   = 0.55;
p_b_xover   = 0.00;
p_h_xover   = 0.55;
p_d_xover   = 0.55;
p_shuffle   = 0.00;

d_factor    = 0.80;
quantum_0   = 0.1;
rand_seed   = 2406525;


proc (1) = m_rate(i,d); retp(0.05); endp;

proc (1) = bm_rate(d); retp(0.005); endp;

proc (2) = f(i,d,v_array);
local c,fv,u,x;
   u = matinit(order, plan, 0);
   x = matinit(order, horizon, 0);
  {u,x} = assign_sequences(i,d,u,x);[27]

   //evaluate objective
   fv = objective(x,horizon);
   retp (fv,v_array);
endp;


proc (1) = objective(x,k);
local fv;
   fv = x[1,k]^2 + x[2,k]^2 - cos(18*x[1,k]) - cos(18*x[2,k]);
   if (maximise); fv = fv; else; fv = -fv; endif;
   retp(fv);
endp;
```

[27] Infra page 5, footnote 10.

Copyright © 1997-2006: Ike's Research Ltd

VI. **The dat File**: Example 3.2

Table 3.2: Input Data for the 2-Dimensional Rastrigin Function

VARIABLE NAME		X1	X2
UCB		1	1
LCB		-1	-1
USB		1	1
LSB		-1	-1
Initial State Vector		0	0
Final State Vector		0	0
Discrete Values		0	0
Inter-connexion Matrix	X1	1	0
	X2	0	1

V. **Points to Note**: Example 3.2

1. `adj_mode` — This parameter should always be set to `"s"` for unconstrained optimisation problems

2. `solution_type` — This parameter should always be set to `"e"` for unconstrained optimisation problems

3. `f(i,d,v_array)` — For unconstrained optimisation problems the form of this procedure is standard as repeated below: it needn't be changed.

```
proc (2) = f(i,d,v_array);
local c,fv,u,x;
   u = matinit(order, plan, 0);
   x = matinit(order, horizon, 0);
  {u,x} = assign_sequences(i,d,u,x);[28]

   //evaluate objective
   fv = objective(x,horizon);
   retp (fv,v_array);
endp;
```

4. `tuning` — Apart from a judicious choice of the parameter `quantum_0`, no special tuning was required for this problem.

[28] Infra page 5, footnote 10.

Copyright © 1997-2006: Ike's Research Ltd

Example 3.3: Static Constrained Optimisation: The Economic Dispatch Problem[29]

$$\min_{\mathbf{x}} J(\mathbf{x}) = 3x_1 + 0.000001x_1^3 + 2x_2 + 0.000002/3x_2^3$$

Subject to:

$$x_4 - x_3 + 0.55 \geq 0$$

$$x_3 - x_4 + 0.55 \geq 0$$

$$1000\sin(-x_3 - 0.25) + 1000\sin(-x_4 - 0.25) + 894.8 - x_1 = 0$$

$$1000\sin(x_3 - 0.25) + 1000\sin(x_3 - x_4 - 0.25) + 894.8 - x_2 = 0$$

$$1000\sin(x_4 - 0.25) + 1000\sin(x_4 - x_3 - 0.25) + 1294.8 = 0$$

$$x_i \in [0, 1200],\ i = 1,2$$

$$x_i \in [-0.55, 0.55],\ i = 3,4$$

I. GENO Output

Generation	*Function Value*
0	4246.144086
50	4221.986797
100	4221.956534
150	4221.956534
200	4221.956534
220	4221.956534
240	4221.956534
260	4221.956529
270	4221.956525
280	4221.956525
290	4221.956525
300	4221.956525

Optimal State Vector:[30] $\mathbf{x}$ = (589.265856, 1124.772836, 0.184406, -0.365594)^T

Objective Function Value: $J(\mathbf{x})$ = 4221.956525

II. General Remarks

This problem appears in Joines and Houck (1994), as well as Michalewicz and Fogel (2000), and according to the latter, the best known solution remains that reported by Hock and Schittkowski (1981), namely 5126.4981. And as regards the experiments of Joines and Houck, Michalewicz and Fogel write:

> "The best result reported was evaluated at 5126.6653. Interestingly, no solution was fully feasible due to the equality constraints, but the sum of the violated constraints was quite small (10^{-4})." (Paraphrased from p.252)

As can be seen from the results above, GENO finds a significantly better solution in comparison. In fact not only is the solution better, it is totally feasible as well. Interestingly, the latest research by others[31] has not unearthed a better solution than that found by GENO. This applies to at least two other GAUSS examples.[32]

---

[29] **Source**: Hock and Schittkowski (1981, p.95).

[30] The variables x_k, $\mathbf{x}_k$ and $\overline{\mathbf{x}}_k$ should not be confused: x_l denotes the *k*th component of the state vector; $\mathbf{x}_k$ denotes the entire state vector evaluated at time t = k; and finally, $\overline{\mathbf{x}}_k$ is a matrix whose columns are the states vector at times t = 0, 1, 2, . . ., k. On static problems, the variable of interest is normally the final state vector $\mathbf{x}_T$.

[31] See Hedar and Fukushima (2005).

[32] See Examples `sc_sp8a.e` or `sc_sp8b.e`; also `sc_sp9b.e` which appears in this document as Example 3.2 below.

Copyright © 1997-2006: Ike's Research Ltd

III. **The gep File**: Example 3.3

```
// A constrained uni-objective static optimization problem
// Source: Hock and Schittkowski (1981, p.95).

#definecs p_maxgens  300
#definecs p_popsize  20
#definecs p_agents   1
#definecs p_order    1
#definecs p_plan     1

#include static_gep_defaults.src

let vars[p_agents, p_order]   = 1;

adj_mode        = "s";
solution_type   = "e";
maximise        = false;
pos_orth        = true;

p_s_xover       = 0.00;
p_a_xover       = 0.55;
p_b_xover       = 0.00;
p_h_xover       = 0.55;
p_d_xover       = 0.55;
p_shuffle       = 0.00;

d_factor        = 0.250;
quantum_0       = 0.01;
rand_seed       = 2406525;

proc (1) = m_rate(i,d); retp(0.05); endp;

proc (1) = bm_rate(d); retp(0.005); endp;


proc (2) = f(i,d,v_array);
local c,fv,u,v,w,x,z;
   v = zeros(2,1);
   u = matinit(order,plan,0);
   x = matinit(order,horizon,0);
  {u,x} = assign_sequences(i,d,u,x);[33]

   //evaluate constraints
   z = equations(x,0,horizon);
   c = constraints(z,x,horizon);
   v_array  = evaluate_constraints(c,v_array);
   v_array[2] = v_array[2] + distance_to_set(-1,1,z[4]);[34]

   //evaluate objective
   fv = objective(z,x,horizon);
   retp (fv,v_array);
endp;


proc (1) = objective(z,x,k);
local fv;
   fv = 3*z[1] + 0.000001*(z[1]^3) + 2*z[2] + (0.000002/(3*(z[2]^3)));
   if (maximise); fv = fv; else; fv = - fv; endif;
   retp(fv);
endp;


proc (1) = constraints(z,x,k);
local c; c = zeros(2,1);
   c[1] = x[1,k] - z[3] + 0.55;
   c[2] = z[3] - x[1,k] + 0.55;
   retp (c);
endp;
```

[33] Infra page 5, footnote 10.

[34] The procedure `proc distance_to_set(a,b,c)` is internal to GENO. It returns the shortest distance from c to the interval [a, b].

Copyright © 1997-2006: Ike's Research Ltd

```
proc (1) = equations(x,y,k);
local z; z = zeros(4,1);
   z[4] = -1.2948 - sin(x[1,k] - 0.25);
   z[3] = x[1,k] - 0.25 - asin(z[4]);
   z[2] = 1000*sin( z[3] - 0.25) + 1000*sin(z[3] - x[1,k] - 0.25) + 894.8;
   z[1] = 1000*sin(-z[3] - 0.25) + 1000*sin(-x[1,k] - 0.25) + 894.8;
   retp (z);
endp;
```

VI. **The dat File**: Example 3.3

Table 3.3: Input Data for the Economic Dispatch Problem

VARIABLE NAME		X1
UCB		1e20
LCB		-1e20
USB		-0.299
LSB		-0.55
Initial State Vector		0
Final State Vector		0
Discrete Values		0
Inter-connexion Matrix	X1	1

V. **Points to Note**: Example 3.3

1. `Method` — The gep and dat files presented above implement version MPc of the original mathematical program. The equations constraints were eliminated as follows. First we note that, if either x_3 or x_4 were given, would have three equations in three unknowns which we can then "solve" sequentially in terms of the given variable. To that end, assume x_4 is known, and make the following definitions:

$$z_4 \equiv \sin(x_4 - x_3 - 0.25)\,; \quad z_3 \equiv x_3\,; \quad z_2 \equiv x_2\,; \quad z_1 \equiv x_1$$

Then, using the last of the equation constraints, we can write,

$$z_4 = -\sin(x_4 - 0.25) - 1.2948$$

And using the definitions of z_3 and z_4, we can write:

$$z_3 = x_4 - 0.25 - \arcsin(z_4)$$

And from the first two equation constraints, we have that:

$$z_2 = 1000\sin(z_3 - 0.25) + 1000\sin(z_3 - x_4 - 0.25) + 894.8$$

$$z_1 = 1000\sin(-z_3 - 0.25) + 1000\sin(-x_4 - 0.25) + 894.8$$

The above set of equation is what is implemented by **proc** equations. Note that the mutual dependency of the z-equations dictates that they are evaluated in the order presented above.

2. `Solution Control` — By the substitutions described above, the size of the state vector is reduced to one, namely the variable x_4 (which is implemented as x[1, k] in the program). This variable is controlled by the assumed dynamic process itself. But the compound variable z_4 is another variable whose evolution needs to be controlled. This is achieved by noting that the range of z_4 (being the trigonometric sine function) is [-1, 1]; hence the statement in **proc** `f(i,d,v_array)` which evaluates the distance of z[4] from [-1, 1], viz.:

```
v_array[2] = v_array[2] + distance_to_set(-1,1,z[4]);
```

3. `Algorithm Tuning` — As in most examples presented here, GENO was effectively tuned using only four parameters: `d_factor`, `quantum_0` and the return values of `m_rate`, and `bm_rate`.

Copyright © 1997-2006: Ike's Research Ltd

Example 3.4: Static Constrained Optimisation: A Pressure Vessel Design Problem[35]

$$\min_{\mathbf{x}} J(\mathbf{x}) = 0.6224x_1x_3x_4 + 1.7781x_2x_3^2 + 3.1661x_4x_1^2 + 19.84x_3x_1^2$$

Subject to:

$$x_4 - 240 \le 0$$

$$-x_1 + 0.0193x_3 \le 0$$

$$-x_2 + 0.00954x_3 \le 0$$

$$-\pi x_3^2 x_4 - \frac{4}{3}\pi x_3^3 + 1{,}296{,}000 \le 0$$

$$\mathrm{x}_i \in [0.0625, 99] \cap \{\mathrm{x}_i : \mathrm{x}_i = 0.0625\mathrm{N}, \mathrm{N} \in \mathbf{Z}\},\ i = 1, 2;\quad x_i \in [10.0, 200], i = 3, 4$$

I. GENO Output [36]

Generation	*Objective 1*	*Objective 2*
0	1500647.737740	0.000000
100	6059.828502	0.000000
200	6059.715394	0.000000
320	6059.714347	0.000000
340	6059.714347	0.000000
360	6059.714336	0.000000
380	6059.714336	0.000000
400	6059.714336	0.000000
440	6059.714336	0.000000
480	6059.714336	0.000000
500	6059.714336	0.000000

Optimal State Vector: $\mathbf{x}$ = (0.812500, 0.437500, 42.098446, 176.636596) ^{T}

Objective Function Value: $J(\mathbf{x})$ = 6059.714336

II. General Remarks

This problem has previously been tackled by Deb (1997) using GeneAS (Genetic Adaptive Search); by Kannan and Kramer (1994) using an augmented Lagrangian multiplier method; and by Sandgren (1988) using a branch and bound technique; and by Coello Coello (2000) using a genetic algorithm. Coello Coello (2000, p.18) presents a comparison of these methods together with his technique: the table below is an extract from there to which has been appended the result by GENO.

	Coello Coello	Deb (1997)	Kannan, *et al.*	Sandgren	GENO
Best Function Value	6069.3267	6410.3811	7198.0428	8129.1036	6059.714336

As can be seen, the solution by GENO is by far the best amongst those considered; in fact, as of this writing, it is the best currently known solution. Note also that in the final solution vector, x_1 and x_2 are integer multiples of 0.0625 as required. Although Hedar and Fukushima (2005, p.19) claim to have found a better solution valued 5868.764836, it should be noted that their solution ignores the discreteness restriction on x_1 and x_2, and so their algorithm cannot, strictly speaking, be compared to GENO.

[35] **Source**: Coello Coello (2000).

[36] **Legend**: *Objective 1* is the actual function being minimised; *Objective 2* is a merit function for an auxiliary NCP.

Copyright © 1997-2006: Ike's Research Ltd

III. **The gep File**: Example 3.4

```
// A constrained uni-objective static optimization problem
// Source: Coello Coello (2000).

#definecs p_maxgens  500
#definecs p_popsize  20
#definecs p_agents   2
#definecs p_order    4
#definecs p_plan     1

#include static_gep_defaults.src

let vars[p_agents, p_order]   = {1   1   1   1,
                                 1   1   1   1};

let discrete_var[p_order]     = 1   1   0   0; //An indicator array for discrete variables

adj_mode        = "s";
solution_type   = "e";
maximise        = false;

p_s_xover       = 0.00;
p_a_xover       = 0.55;
p_b_xover       = 0.00;
p_h_xover       = 0.55;
p_d_xover       = 0.55;
p_shuffle       = 0.00;

d_factor        = 0.80;
quantum_0       = 0.1;
rand_seed       = 240657;

proc (1) = m_rate(i,d); retp(0.05); endp;

proc (1) = bm_rate(d); retp(0.005); endp;

proc (2) = dynamic_state_bounds(i,j,k,low,upp);
local fv,fv1,fv2,u,x;
   k = horizon;
   fv1 = low; fv2 = upp;
   u = matinit(order, plan, 0);
   x = matinit(order, horizon, 0);
   {u,x} = assign_sequences(i,1,u,x);[37]
   if(j == 3);  fv2 = min((x[1,k] / 0.0193),(x[2,k] / 0.00954));
   elseif(j == 4); fv2 = 240;
   endif;
   {low,upp} = common_set(low,upp,fv1,fv2);[38]
   retp(low, upp);
endp;


//
// Note: An alternative specification of the above procedure is as follows
//
// proc (2) = dynamic_state_bounds(i,j,k,low,upp);
// local fv,fv1,fv2;
//    k = horizon;
//    fv1 = low; fv2 = upp;
//    if(j == 3);  fv2 = min((state_fn(i,1,k)[39] / 0.0193),(state_fn(i,2,k) / 0.00954));
//    elseif(j == 4); fv2 = 240;
//    endif;
//    {low,upp} = common_set(low,upp,fv1,fv2);
//    retp(low, upp);
//  endp;
//
```

[37] Infra page 5, footnote 10.

[38] The procedure **proc** `common_set(a,b,c,d)` is internal to GENO. It returns the intersection of the two intervals [a, b] and [c, d].

[39] The procedure **proc** `state_fn(i,j,k)` is internal to GENO. It returns the value of the *j*-th component of the state vector, which resides in the *i*-th chromosome, at the *k*-th time instant. A similarly defined control variable value is returned by **proc** `control_fn(i,j,k)`.

Copyright © 1997-2006: Ike's Research Ltd

```
proc (2) = f(i,d,v_array);
local c, fv, u, x;
   u = matinit(order,plan,0);
   x = matinit(order,horizon,0);
   {u,x} = assign_sequences(i,d,u,x);[40]

   //evaluate constraints
   c = constraints(0, x, horizon);
   v_array = evaluate_constraints(c, v_array);

   //evaluate objectives
   if(d == 1); fv = objective(0, x, horizon); retp (fv, v_array);
   elseif(d == 2); retp(merit_fcn("b", c, c, v_array));[41]
   endif;
endp;


proc (1) = objective(z,x,k);
local fv;
   fv = 0.6224*x[1,k]*x[3,k]*x[4,k] + 1.7781*x[2,k]*(x[3,k]^2) + 3.1661*x[4,k]*(x[1,k]^2) \
        + 19.84*x[3,k]*(x[1,k]^2);
   if (maximise); fv = fv; else; fv = - fv; endif;
   retp(fv);
endp;


proc (1) = constraints(z,x,k);
local c; c = zeros(4,1);
   c[1] = x[4,k] - 240;
   c[2] = x[3,k]*0.0193 - x[1,k];
   c[3] = 0.00954*x[3,k] - x[2,k];
   c[4] = 1296000 - pi*(x[3,k]^2)*x[4,k] - 4*pi*(x[3,k]^3)/3.0;
   retp(c);
endp;
```

VI. The dat File: Example 3.4

Table 3.4: Input Data for the Pressure Vessel Design Problem

VARIABLE NAME		X1	X2	X3	X4
UCB		99	99	200	200
LCB		0.0625	0.0625	10	10
USB		99	99	200	200
LSB		0.0625	0.0625	10	10
Initial State Vector		0	0	0	0
Final State Vector		0	0	0	0
Discrete Values		0.0625	0.0625	0	0
Inter-connexion Matrix	X1	1	0	0	0
	X2	0	1	0	0
	X3	0	0	1	0
	X4	0	0	0	1

[40] Infra page 5, footnote 10.

[41] The procedure **proc** `merit_fcn` is internal to GENO. It returns the relative merit of each chromosome as evaluated by an auxiliary M-program that aids the search for the optimal solution, **x***. It should be called exactly in the manner shown if the auxiliary M-program is of type "b". Refer to page 12, § 3.4.2 for more information.

Copyright © 1997-2006: Ike's Research Ltd

V. **Points to Note**: Example 3.4

1. `Method` — The gep and dat files presented above implement of version MPb of the original mathematical program. As explained in § 3.4.2 above, MPb is a bi-objective optimisation problem: in addition to the original optimisation problem, a second mathematical program is defined such that its solution coincides with the value of the constraint functions at **x*** — the optimal vector as evaluated by the original objective. Assuming a problem with *m* inequality constraints, the solution of the second program may be obtained by minimising a merit function associated with an auxiliary NCP defined as follows: given any **x**, find the set of all *m*-vectors ω such that:

$$\omega \geq 0\,; \qquad [c_i(x) - \omega_i]^2 \geq 0\,; \qquad \omega_i[c_i(x) - \omega_i]^2 = 0\,; \qquad i = \{1, 2, \cdots, m\}$$

This logic explains the following code fragment in **proc** `f(i,d,v_array)`:

```
if(d == 1); fv = objective(0, x, horizon); retp (fv,v_array);
elseif(d == 2); retp(merit_fcn("b", c, c, v_array));
```

The **proc** `merit_fcn` itself, this is explained in the next note.

2. `Merit Function` — GENO's NCP approach is based on the merit function (Facchenei and Soares: 1997):

$$\Psi(x) = \sum_{i=1}^{n} \phi(x_i, F_i(x))^2$$

where ϕ is the Fischer-Burmeister NCP-function defined as

$$\phi(a, b) = \sqrt{a^2 + b^2} - (a + b)\,.$$

The procedure **proc** `merit_fcn` is internal to GENO and is a direct implementation of Ψ as defined above, but with x_i replaced by ω_i, and $F_i(x)$ replaced by $[c_i(\hat{x}) - \omega_i]^2$.

Clearly, there are two solutions: ω = 0 and ω = c(x). The first is not useful for our purposes since it does not in any way contribute to the search for *x**. Therefore, in the search for a solution to the auxiliary NCP throughout GENO's evolution, there should be a mechanism that encourages the latter solution.

3. `Discrete Variable` — Problems with discrete variables are easily dealt with by GENO. In this example, x_1 and x_2 are restricted to integer multiples of 0.065. Accordingly, the indicator matrix `discrete_var` is re-defined in the gep file; and the required entries are made in the dat file as shown above.

4. `Solution Control` — The procedure **proc** `dynamic_state_bounds` contributes towards the evolution of the variables x_3 and x_4 by evaluating the conjunction of the *static* state variable bounds (USB and LSB) and the *dynamic* state bounds dictated by the first three constraints, viz.

$$x_4 \leq 240$$

$$x_3 \leq \min\left(\frac{x_2}{0.00954}, \frac{x_1}{0.0193}\right) \Leftarrow [-x_1 + 0.0193x_3 \leq 0] \wedge [-x_2 + 0.00954x_3 \leq 0]$$

As in most examples, the GENO was effectively tuned using only the parameters: `d_factor`, `quantum_0` and the return values of `m_rate`, and `bm_rate`.

Copyright © 1997-2006: Ike's Research Ltd

Example 3.5: Static Constrained Optimisation: An Alkylation Process [42]

$$\min_{\mathbf{x}} J(\mathbf{x}) = 5.04x_1 + 0.035x_2 + 10x_3 + 3.36x_5 - 0.063x_4x_7$$

Subject to:

$$35.82 - 0.222x_{10} - 0.9x_9 \geq 0$$

$$-133 + 3x_7 - 0.99x_{10} \geq 0$$

$$(1/0.9 - 0.9)x_9 - (35.82 - 0.222x_{10} - 0.9x_9) \geq 0$$

$$(1/0.99 - 0.99)x_{10} - (-133 + 3x_7 - 0.99x_{10}) \geq 0$$

$$1.12x_1 + 0.13167x_1x_8 - 0.00667x_1x_8^2 - 0.99x_4 \geq 0$$

$$57.425 + 1.098x_8 - 0.038x_8^2 + 0.325x_6 - 0.99x_7 \geq 0$$

$$(1/0.99 - 0.99)x_4 - (1.12x_1 + 0.13167x_1x_8 - 0.00667x_1x_8^2 - 0.99x_4) \geq 0$$

$$(1/0.99 - 0.99)x_7 - (57.425 + 1.098x_8 - 0.038x_8^2 + 0.325x_6 - 0.99x_7) \geq 0$$

$$1.22x_4 - x_1 - x_5 = 0$$

$$-x_6 + 98000x_3/(x_4x_9 + 1000x_3) = 0$$

$$-x_8 + (x_2 + x_5)/x_1 = 0$$

$x_1 \in [0.00001, 2000]$; $x_2 \in [0.00001, 16000]$; $x_3 \in [0.00001, 120]$; $x_4 \in [0.00001, 5000]$

$x_5 \in [0.00001, 2000]$; $x_6 \in [85, 93]$; $x_7 \in [90, 95]$; $x_8 \in [3, 12]$; $x_9 \in [1.2, 4]$; $x_{10} \in [145, 162]$

I. GENO Output

Generation	*Objective 1*	*Objective 2*
0	-1006.767778	0.000000
20	-2080.765400	0.000000
80	-2192.491058	0.000000
160	-4269.990026	0.000000
240	-1763.232099	0.000000
300	-1763.233924	0.000000
340	-1763.233924	0.000000
380	-1763.235308	0.000000
400	-1763.235308	0.000000
480	-1763.745393	0.000000
500	-1763.745454	0.000000

Solution Vector:

$\mathbf{x}$ = (1695.747560, 15766.851262, 54.144966, 3027.278310, 1997.531978, 90.084472, 94.984529, 10.475842, 1.571577, 153.485638)^T

Objective Function Value: $J(\mathbf{x})$ = -1763.745454

II. General Remarks

The original source for this example appears to be Bracken and McCormick (1968), and the best known solution thus far is that reported by Hock and Schittkowski, namely: –1768.80696. This problem has a rather complex feasible region and the search is made more difficult by the fact that six of the constraints are active at the solution. But, as can be seen from the results presented above, GENO's solution is very close to the best known so far.

[42] **Source**: Hock and Schittkowski (1981, p.123).

Copyright © 1997-2006: Ike's Research Ltd

III. **The gep File**: Example 3.5

```
// A constrained uni-objective static optimization problem
// Source: Hock and Schittkowski (1981, p.123)

#definecs p_maxgens     500
#definecs p_popsize     20
#definecs p_agents      2
#definecs p_order       7
#definecs p_plan        1

#include static_gep_defaults.src

let vars[p_agents, p_order]   =  1     1     1     1     1     1     1;

let q_mod[p_order]            =  1     1     1     1   0.1   0.1     1;

adj_mode          = "g";
solution_type     = "m";
sequential_solve  = true;
maximise          = false;
pos_orth          = true;

p_s_xover   = 0.55;
p_a_xover   = 0.55;
p_b_xover   = 0.00;
p_h_xover   = 0.55;
p_d_xover   = 0.55;
p_shuffle   = 0.005;

d_factor    = 0.80;
quantum_0   = 10;
rand_seed   = 240657;

proc (1) = m_rate(i,d); retp(0.05); endp;

proc (1) = bm_rate(d); retp(0.005); endp;

proc (2) = f(i,d,v_array);
local c,fv,u,x;
   u = matinit(order,plan,0);
   x = matinit(order,horizon,0);
  {u,x} = assign_sequences(i,d,u,x);[43]

   //evaluate constraints
   z = equations(x,0,horizon);

   //evaluate objectives
   if(d == 1); fv = objective(z,x,horizon);
   elseif(d == 2);
      c = constraints(z, x, horizon);
      v_array = evaluate_constraints(c, v_array);
      fv = -v_array[2];
   endif;
   retp (fv,v_array);
endp;

proc (1) = objective(z,x,k);
local fv;
   fv = 5.04*z[1] + 0.035*z[2] + 10*z[3] + 3.36*x[2,k] - 0.063*x[1,k]*x[4,k];
   if (maximise); fv = fv; else; fv = -fv; endif;
   retp(fv);
endp;

proc (1) = equations(x,y,k);
local z; z = zeros(3,1);
   z[1] = 1.22*x[1,k] - x[2,k];
   z[2] = z[1]*x[5,k] - x[2,k];
   z[3] = x[1,k]*x[3,k]*x[6,k] / (98000 - 1000*x[3,k]);
   retp (z);
endp;
```

[43] Infra page 5, footnote 10.

Copyright © 1997-2006: Ike's Research Ltd

```
proc (1) = constraints(z,x,k);
local c; c = zeros(8,1);
   c[1] = 3*x[4,k] - 133 - 0.99*x[7,k];
   c[2] = 35.82 - 0.222*x[7,k] - 0.9*x[6,k];
   c[3] = ((1.0/0.99) - 0.99)*x[7,k] - c[1];
   c[4] = ((1.0/0.9) - 0.9)*x[6,k] - c[2];
   c[5] = 1.12*z[1] + 0.13167*z[1]*x[5,k] - 0.00667*z[1]*(x[5,k]^2) - 0.99*x[1,k];
   c[6] = 57.425 + 1.098*x[5,k] - 0.038*(x[5,k]^2) + 0.325*x[3,k] - 0.99*x[4,k];
   c[7] = ((1.0/0.99) - 0.99)*x[1,k] - c[5];
   c[8] = ((1.0/0.99) - 0.99)*x[4,k] - c[6];
   retp (c);
endp;
```

VI. **The dat File**: Example 3.5

Table 3.5: Input Data for the Alkylation Problem

VARIABLE NAME		X1	X2	X3	X4	X5	X6	X7
UCB		1e20	1e20	1e20	1e20	1e20	1e20	1e20
LCB		-1e20	-1e20	-1e20	-1e20	-1e20	-1e20	-1e20
USB		5000	2000	93	95	12	4	162
LSB		0.00001	0.00001	85	90	3	1.2	145
Initial State Vector		0	0	0	0	0	0	0
Final State Vector		0	0	0	0	0	0	0
Discrete Values		0	0	0	0	0	0	0
Inter-connexion Matrix	X1	1	0	0	0	0	0	0
	X2	0	1	0	0	0	0	0
	X3	0	0	1	0	0	0	0
	X4	0	0	0	1	0	0	0
	X5	0	0	0	0	1	0	0
	X6	0	0	0	0	0	1	0
	X7	0	0	0	0	0	0	1

V. **Points to Note**: Example 3.5

1. `Method` — The gep and dat files presented above implement of version MPa of the original mathematical program as a bi-objective problem, i.e. gep type 'aa'.

2. `q_mod` — This example illustrates how the quantum modular array is intended to be used. The initial size of quanta are set as 10 for all variables by the assignment: `quantum_0 = 10` (as opposed to the normal 0.1 because of the large variable ranges); however, for x_5 and x_6 the size of quanta is further "modulated" by the array entries `q_mod` [5] = 0.1 and `q_mod` [6] = 0.1 because these two variables has a relatively narrower in range. However, this devise is currently not very effective; the associated GENO mechanisms are still under development for optimum performance.

3. `sequential_solve` — This example illustrates a bi-objective implementation of MPa (denoted as type 'aa') in which the solution process is explicitly decomposed into its two constituent part, i.e. constraint satisfaction and function optimisation. The two processes act in concert by virtue of the selected adjustment mode (i.e. `adj_mode = "g"`) to reach a point that is Pareto-efficient (see page 54 footnote 71 for further explanation).

4. `adj_mode and solution_type` — Note the options selected for these two parameters when the implementation of M-program MPa is of type 'aa'. These parameter settings must be accompanied by the assignment: `sequential_solve = true`.

Copyright © 1997-2006: Ike's Research Ltd

Example 3.6: Static Constrained Optimisation: Decentralised Economic Planning [44]

$$\min_{\mathbf{x}} J(\mathbf{x}) = (x_1 - 10)^2 + 5(x_2 - 12)^2 + 3(x_4 - 11)^2 + x_3^4 + 10x_5^6 + 7x_6^2 + x_7^4 - 4x_6x_7 - 10x_6 - 8x_7$$

Subject to:

$$127 - 2x_1^2 - 3x_2^4 - x_3 - 4x_4^2 - 5x_5 \geq 0$$

$$282 - 7x_1 - 3x_2 - 10x_3^2 - x_4 - x_5 \geq 0$$

$$196 - 23x_1 - x_2^2 - 6x_6^2 + 8x_7 \geq 0$$

$$-4x_1^2 - x_2^2 + 3x_1x_2 - 2x_3^2 - 5x_6 + 11x_7 \geq 0$$

$$x_i \in [-10, 10], \quad i = 1, \ldots, 7$$

I. GENO Output

Generation	*Objective 1*	*Objective 2*
0	7864498.421740	2.224884
20	106428.689759	0.000000
40	1301.777022	0.000000
60	682.475649	0.000000
80	680.994100	0.000000
100	680.872077	0.000000
120	680.630739	0.000000
140	680.630285	0.000000
160	680.630070	0.000000
180	680.630061	0.000000
200	680.630060	0.000000
220	680.630059	0.000000
240	680.630059	0.000000
260	680.630059	0.000000
280	680.630058	0.000000
300	680.630057	0.000000
340	680.630057	0.000000
380	680.630057	0.000000
420	680.630057	0.000000
460	680.630057	0.000000
500	680.630057	0.000000

Solution Vector: $\mathbf{x}$ = (2.330500, 1.951372, -0.477541, 4.365727, -0.624487, 1.038131, 1.594227) ^{T}

Objective Function Value: $J(\mathbf{x})$ = 680. 630057

II. General Remarks

The original reference for this problem appears to be K. P. Wong's PhD Dissertation, and one of the earliest references to it is Asaadi (1973). It is featured in Michalewicz's 1995 comparative study of evolutionary algorithms, and appears again in Michalewicz and Fogel (2000). Thus far, the best known solution has been that reported by Hock and Schittkowski (1981), namely:

$\mathbf{x}$ = (2.330500, 1.951372, -0.477541, 4.365727, -0.624487, 1.038131, 1.594227)^T with $J(\mathbf{x})$ = 680.630057

Most of the methods tested by Michalewicz (1995) generate reasonable solutions. The method of Michalewicz and Attia (1994) exhibits the best overall performance; their best solution is 680.642. In comparison, GENO finds a solution that is as good as Hock and Schittkowski's.

[44] **Source**: Hock and Schittkowski (1981, p.111).

Copyright © 1997-2006: Ike's Research Ltd

III. **The gep File**: Example 3.6

```
// A constrained uni-objective static optimization problem
// Source: Hock and Schittkowski (1981, p.111)

#definecs p_maxgens     500
#definecs p_popsize     20
#definecs p_agents      1
#definecs p_order       7
#definecs p_plan        1

#include static_gep_defaults.src

let vars[p_agents, p_order]   = {1   1   1   1   1   0   0,
                                 0   0   0   0   0   1   1};
adj_mode        = "s";
solution_type   = "e";
maximise        = false;
pos_orth        = true;

p_s_xover       = 0.55;
p_a_xover       = 0.55;
p_b_xover       = 0.00;
p_h_xover       = 0.55;
p_d_xover       = 0.55;
p_shuffle       = 0.00;

d_factor        = 0.80;
quantum_0       = 0.1;
rand_seed       = 240657;

proc (1) = m_rate(i,d); retp(0.05); endp;

proc (1) = bm_rate(d); retp(0.005); endp;

proc (2) = f(i,d,v_array);
local c, fv, g, u, x, z;
   u = matinit(order,plan,0);
   x = matinit(order,horizon,0);
   {u, x} = assign_sequences(i,d,u,x);[45]

   //evaluate constraints
   z = equations(x, 0, horizon);
   g = gradients(z, x, horizon);
   c = constraints(z, x, horizon);
   v_array = evaluate_constraints(c, v_array);

   //evaluate objectives
   if(d == 1); fv = objective(z, x, horizon); retp (fv, v_array);
   elseif(d == 2); retp(merit_fcn("d", g, x, v_array));[46]
   endif;
endp;


proc (1) = objective(z,x,k);
local fv;
   fv = (x[1,k] - 10)^2 + 5*((x[2,k] - 12)^2) + z[1]^4 + 3*((x[3,k] - 11)^2) \
        + 10*(x[4,k]^6) + 7*(z[2]^2) + x[5,k]^4 - 4*x[5,k]*z[2] - 10*z[2] - 8*x[5,k];
   if (maximise); fv = fv; else; fv = - fv; endif;
   retp(fv);
endp;

proc (1) = constraints(z,x,k);
local c; c = zeros(2,1);
   c[1] = 282 - 7*x[1,k] - 3*x[2,k] - 10*z[1]^2 - x[3,k] + x[4,k];
   c[2] = 196 - 23*x[1,k] - x[2,k]^2 - 6*z[2]^2 + 8*x[5,k];
   retp (c);
endp;
```

[45] Infra page 5, footnote 10.

[46] The procedure **proc** `merit_fcn` is internal to GENO. It returns the relative merit of each chromosome as evaluated by an auxiliary M-program that aids the search for the optimal solution, **x***. It should be called exactly in the manner shown if the auxiliary M-program is of type "d". Refer to page 15, § 3.4.4 for more information.

Copyright © 1997-2006: Ike's Research Ltd

```
proc (1) = equations(x,y,k);
local z; z = zeros(2,1);
   z[1] = 127 - 2*x[1,k]^2 - 3*(x[2,k]^4) - 4*(x[3,k]^2) - 5*x[4,k] - y[6,k];
   z[2] = (-4*(x[1,k]^2) - x[2,k]^2 + 3*x[1,k]*x[2,k] - 2*(z[1]^2) \
            + 11*x[5,k] - y[7,k])/5.0;
   retp (z);
endp;


proc (1) = gradients(z,x,k);
local g; g = zeros(2,1);
   g[1] = (4*z[1]/5)*(14*z[2] - 4*x[5,k] - 10) - 4*(z[1]^3);
   g[2] = (10 + 4*x[5,k] - 14*z[2])/5;
   retp(g);
endp;
```

VI. **The dat File**: Example 3.6

Table 3.6: Input Data for the Decentralised Economic Planning Problem

VARIABLE NAME		X1	X2	X3	X4	X5	X6	X7
UCB		10	10	10	10	10	1e20	1e20
LCB		-10	-10	-10	-10	-10	0	0
USB		10	10	10	10	10	10	10
LSB		-10	-10	-10	-10	-10	0	0
Initial State Vector		0	0	0	0	0	0	0
Final State Vector		0	0	0	0	0	0	0
Discrete Values		0	0	0	0	0	0	0
Inter-connexion Matrix	X1	1	0	0	0	0	0	0
	X2	0	1	0	0	0	0	0
	X3	0	0	1	0	0	0	0
	X4	0	0	0	1	0	0	0
	X5	0	0	0	0	1	0	0
	X6	0	0	0	0	0	1	0
	X7	0	0	0	0	0	0	1

Copyright © 1997-2006: Ike's Research Ltd

V. **Points to Note**: Example 3.6

1. `Method` — The gep and dat files presented above implement version MPd of the original mathematical program. This was derived as follows. First, lay out the *incidence matrix* that indicates the presence or non-presence of each variable in each constraint, i.e.

CONSTRAINT NO	X1	X2	X3	X4	X5	X6	X7
1	✓	✓	✓	✓	✓		
2	✓	✓	✓	✓	✓		
3	✓	✓				✓	✓
4	✓	✓	✓			✓	✓

Select a particular variable to eliminate from a particular constraint based on (a) whether the said variable appears in the objective function, and (b) whether it is easy to manipulate the selected constraint so as to make the said variable the subject of any subsequent equation. These heuristics are merely so as to make the problem re-formulation easy. For this particular example, constraint #1 was used to eliminate x_3 and constraint #4 was used to eliminate x_6 as follows.

In constraint #1, introduce a positive slack variable ω_1 to obtain,

$$127 - 2x_1^2 - 3x_2^4 - x_3 - 4x_4^2 - 5x_5 - \omega_1 = 0 \tag{3.6a}$$

Introduce a "new" variable $z_1 \equiv x_3$ and solve the equation above for z_1, i.e.

$$z_1 \equiv x_3 = 127 - 2x_1^2 - 3x_2^4 - 4x_4^2 - 5x_5 - \omega_1 \tag{3.6b}$$

Similarly, one can eliminate x_6 using constraint #4 to obtain

$$z_2 \equiv x_6 = \left(-4x_1^2 - x_2^2 + 3x_1x_2 - 2x_3^2 + 11x_7 - \omega_2\right)/5 \tag{3.6c}$$

Note that no further elimination can be done without introducing cyclical interdependencies in the system of constraints, and the current version of GENO cannot handle cyclicality. In order to see this, rename the variables as follows:

OLD NAME	X1	X2	X3	X4	X5	X6	X7	-	-
Intermediate	-	-	Z1	-	-	Z2	-	ω_1	ω_2
NEW NAME	X1	X2	Z1	X3	X4	Z2	X5	ω_1	ω_2

The modified program is thus:

Given:

$$z_1 = 127 - 2x_1^2 - 3x_2^4 - 4x_3^2 - 5x_4 - \omega_1$$

$$z_2 = \left(-4x_1^2 - x_2^2 + 3x_1x_2 - 2z_1^2 + 11x_7 - \omega_2\right)/5$$

$$\omega \in \mathbf{R}_+$$

$$\min_{\mathbf{x}} J(\mathbf{x}) = (x_1 - 10)^2 + 5(x_2 - 12)^2 + 3(x_3 - 11)^2 + z_1^4 + 10x_4^6 + 7z_6^2 + x_5^4 - 4z_2x_5 - 10z_2 - 8x_5$$

Subject to:

$$282 - 7x_1 - 3x_2 - 10z_1^2 - x_3 - x_4 \geq 0$$

$$196 - 23x_1 - x_2^2 - 6z_2^2 + 8x_5 \geq 0$$

$$x_i \in [-10, 10], \quad i = 1, \ldots, 5$$

Copyright © 1997-2006: Ike's Research Ltd

To formulate the equivalent bi-objective program outlined in section 1.4.4 above, we need the gradient vector $\partial f/\partial\omega$. And in this particular case the components of the gradient vector are easy to derive. Using the chain rule of differential calculus,

$$\frac{\partial f}{\partial \omega_m} = \sum_{n=1}^{2} \frac{\partial f}{\partial z_n} \cdot \frac{\partial z_n}{\partial \omega_m}, \quad m = 1, 2$$

we have that:

$$g_1 \equiv \frac{\partial f}{\partial \omega_1} = (4z_1/5)\left[14z_2 - 4x_5 - 10\right] - 4z_1^3$$

$$g_2 \equiv \frac{\partial f}{\partial \omega_2} = (10 + 4x_5 - 14z_2)/5$$

The g-equations are implemented by **proc** `gradients`; the z-equations are implemented by **proc** `equations`. Note that the mutual dependency of the z-equations dictates that they are evaluated in the order presented above.

2. `vars` — The `vars` matrix indicates the variables in each sub-program. In the NCP implementation, ω_1 and ω_2 are co-opted into the state vector as its final two components: the structure of the vars matrix implements the fact that these two components pertain to the second program of the bi-objective scheme, viz.:

$$\min_{\omega} h(\omega)$$

$$\text{Subject to:} \quad \omega \in \mathbf{R}_+;$$

3. `v_array` — The z-vector is required to be feasible, hence the statements:

```
v_array[2] = v_array[2] + distance_to_set(-10,10,z[1]);[47]
v_array[2] = v_array[2] + distance_to_set(-10,10,z[2]);
```

[47] Infra page 23, footnote 34.

Copyright © 1997-2006: Ike's Research Ltd

Example 3.7: Static Constrained Optimisation: Heat Exchanger Optimisation [48]

$$\min_{\mathbf{x}} J(\mathbf{x}) = x_1 + x_2 + x_3$$

Subject to:

$$1 - 0.0025(x_4 + x_6) \geq 0$$

$$1 - 0.0025(x_5 + x_7 - x_4) \geq 0$$

$$1 - 0.01(x_8 - x_5) \geq 0$$

$$x_1 x_6 - 833.33252 x_4 - 100 x_1 + 83333.333 \geq 0$$

$$x_2 x_7 - 1250 x_5 - x_2 x_4 + 1250 x_4 \geq 0$$

$$x_3 x_8 - 1250000 - x_3 x_5 + 2500 x_5 \geq 0$$

$$x_1 \in [100, 10000]\,;\; x_i \in [1000, 10000], \quad i = 2,3\,;\; x_i \in [10, 1000], \quad i = 4, \ldots, 8$$

I. GENO Output

Generation	*Objective 1*	*Objective 2*
0	7516.821758	0.738992
10	7055.554338	0.000000
20	7049.332889	0.000000
30	7049.249409	0.000000
40	7049.248050	0.000000
50	7049.248022	0.000000
60	7049.248022	0.000000
70	7049.248021	0.000000
80	7049.248021	0.000000
90	7049.248021	0.000000
100	7049.248021	0.000000

Solution Vector: x = (579.306680, 1359.970648, 5109.970692, 182.017699, 295.601172, 217.982301, 286.416527, 395.601172) ^{T}

Objective Function Value: $J(\mathbf{x})$ = 7049.248021

II. General Remarks

This is a real-life engineering design problem whose object is to optimise the performance of a heat exchanger. The problem was first tackled using the geometric programming approach of Avriel and Williams (1971) and it has been subjected to various other techniques ever since. It is featured in Michalewicz's 1995 comparative study of evolutionary algorithms and again in Michalewicz and Fogel (2000). According to Michalewicz and Fogel (2000), the best known solution is still that reported by Hock and Schittkowski (1981), namely:

$\mathbf{x}$ = (579.3167, 1359.943, 5110.071, 182.0174, 295.5985, 217.9799, 286.4162, 395.5979)^T with $J(\mathbf{x})$ = 7049.330923

None of the methods tested by Michalewicz (1995) comes close to the solution above. The best evolutionary solution appears to be that computed by Michalewicz's GENECOP III system, namely 7286.650.

In contrast and without any problem pre-processing effort, GENO's NCP method computes a better solution at 7063.918114.[49] The solution reported above (which is even better than that computed by Hock and Schittkowski, albeit marginally) was found after a moderate pre-processing effort that eliminated all the inequality constraints by the method described in section 1.4.4 (see § **Points to Note**: Example 3.7 below).

[48] **Source**: Hock and Schittkowski (1981, p.115)

[49] Test run Example sc_sp1b.e

Copyright © 1997-2006: Ike's Research Ltd

III. **The gep File**: Example 3.7

```
// A constrained uni-objective static optimization problem
// Source: Hock and Schittkowski (1981, p.115)

#definecs p_maxgens  100
#definecs p_popsize  20
#definecs p_agents   2
#definecs p_order    8
#definecs p_plan     1

#include static_gep_defaults.src

let vars[p_agents, p_order]   = {1   1   0   0   0   0   0   0,
                                 0   0   1   1   1   1   1   1};

adj_mode       = "s";
solution_type  = "e";
maximise       = false;
pos_orth       = true;

p_s_xover      = 0.00;
p_a_xover      = 0.55;
p_b_xover      = 0.00;
p_h_xover      = 0.55;
p_d_xover      = 0.55;
p_shuffle      = 0.55;

d_factor       = 0.80;
quantum_0      = 1.0;
rand_seed      = 240657;

proc (1) = m_rate(i,d); retp(0.05); endp;

proc (1) = bm_rate(d); retp(0.005); endp;


proc (2) = f(i, d, v_array);
local c,fv,g,u,x,z;
   u = matinit(order,plan,0);
   x = matinit(order,horizon,0);
  {u, x} = assign_sequences( i,d,u,x);[50]

   //evaluate constraints
   z = equations(x,0,horizon);
   g = gradients(z,x,horizon);
   c = constraints(0,x,horizon);
   v_array  = evaluate_constraints(c,v_array);
   v_array[2] = v_array[2] + distance_to_set( 100,10000,z[1]);[51]
   v_array[2] = v_array[2] + distance_to_set(1000,10000,z[2]);
   v_array[2] = v_array[2] + distance_to_set(1000,10000,z[3]);
   v_array[2] = v_array[2] + distance_to_set(10,1000,z[4]);
   v_array[2] = v_array[2] + distance_to_set(10,1000,z[5]);
   v_array[2] = v_array[2] + distance_to_set(10,1000,z[6]);

   //evaluate objectives
   if(d == 1); fv = objective(z, x, horizon); retp (fv, v_array);
   elseif(d == 2); retp(merit_fcn("d", g, x, v_array));[52]
   endif;
endp;


proc (1) = objective(z,horizon);
local fv;
   fv = (z[1] + z[2] + z[3]);
   if (maximise); fv = fv; else; fv = - fv; endif;
   retp(fv);
endp;
```

[50] Infra page 5, footnote 10.

[51] Infra page 23, footnote 34.

[52] Infra page 33, foot note 46.

Copyright © 1997-2006: Ike's Research Ltd

```
proc (1) = equations(x,y,k);
local z; z = zeros(6,1);
   z[6] = 100 + x[2,k] - x[8,k];
   z[5] = 400 + x[1,k] - x[2,k] - x[7,k];
   z[4] = 400 - x[1,k] - x[6,k];
   z[3] = (x[5,k] - 2500*x[2,k] + 1250000) / (z[6] - x[2,k]);
   z[2] = (x[4,k] + 1250*x[2,k] - 1250*x[1,k]) / (z[5] - x[1,k]);
   z[1] = (x[3,k] + 833.33252*x[1,k] - 83333.333) / (z[4] - 100);
   retp(z);
endp;


proc (1) = gradients(z,x,k);
local g; g = zeros(6,1);
   g[1] = 1/(z[4] - 100);
   g[2] = 1/(z[5] - x[1,k]);
   g[3] = 1/(z[6] - x[2,k]);
   g[4] = (x[3,k] + 833.33252*x[1,k] - 83333.333) / (z[4] - 100)^2;
   g[5] = (x[4,k] + 1250*x[2,k] - 1250*x[1,k]) / (z[5] - x[1,k])^2;
   g[6] = (x[5,k] - 2500*x[2,k] + 1250000) / (z[6] - x[2,k])^2;
   retp(g);
endp;
```

VI. **The dat File**: Example 3.7

Table 3.7: Input Data for the Heat Exchanger Optimisation Problem

VARIABLE NAME		X1	X2	X3	X4	X5	X6	X7	X8
UCB		1000	1000	10	10	10	10	10	10
LCB		10	10	0	0	0	0	0	0
USB		1000	1000	10	10	10	10	10	10
LSB		10	10	0	0	0	0	0	0
Initial State Vector		0	0	0	0	0	0	0	0
Final State Vector		0	0	0	0	0	0	0	0
Discrete Values		0	0	0	0	0	0	0	0
Inter-connexion Matrix	X1	1	0	0	0	0	0	0	0
	X2	0	1	0	0	0	0	0	0
	X3	0	0	1	0	0	0	0	0
	X4	0	0	0	1	0	0	0	0
	X5	0	0	0	0	1	0	0	0
	X6	0	0	0	0	0	1	0	0
	X7	0	0	0	0	0	0	1	0
	X8	0	0	0	0	0	0	0	1

Copyright © 1997-2006: Ike's Research Ltd

V. **Points to Note**: Example 3.7

1. `Method` — The gep and dat files presented above implement version MPd of the original mathematical program, and this was derived as follows. The incidence matrix below "points to" inequality 4, 5 and 6 as prime targets for elimination using variables x_1, x_2 and x_3 (because these variables do not appear in more than one constraint and thus one avoids "circular dependencies")

CONSTRAINT NO	X1	X2	X3	X4	X5	X6	X7	X8
1				✓		✓		
2				✓	✓		✓	
3					✓			✓
4	✓			✓		✓		
5		✓		✓	✓		✓	
6			✓		✓			✓

Into constraint #4, #5 and #6 respectively, introduce slack variables ω_1, ω_2 and ω_3 and define the following equations.

$$z_1 \equiv x_1 = [\omega_1 + 833.33252x_4 - 83333.333]/[x_6 - 100] \tag{3.7a}$$

$$z_2 \equiv x_2 = [\omega_2 + 1250x_5 - 1250x_4]/[x_7 - x_4] \tag{3.7b}$$

$$z_3 \equiv x_3 = [\omega_3 + 1250000 - 2500x_5]/[x_8 - x_5] \tag{3.7c}$$

Rename the variables as follows:

OLD NAME	X1	X2	X3	X4	X5	X6	X7	X8	-	-	-
Intermediate	Z1	Z2	Z3	-	-	-	-	-	ω_1	ω_2	ω_3
NEW NAME	Z1	Z2	Z3	X1	X2	X3	X4	X5	ω_1	ω_2	ω_3

Then an intermediate version of the original mathematical program following the above operations is:

Given:

$$z_1 = [\omega_1 + 833.33252x_1 - 83333.333]/[x_3 - 100]$$

$$z_2 = [\omega_2 + 1250x_2 - 1250x_1]/[x_4 - x_1]$$

$$z_3 = [\omega_3 + 1250000 - 2500x_2]/[x_5 - x_2]$$

$$\omega \in \mathbf{R}_+$$

$$\min_{\mathbf{x}} J(\mathbf{x}) = z_1 + z_2 + z_3$$

Subject to:

$$400 - x_1 - x_3 \geq 0$$

$$400 + x_1 - x_2 - x_4 \geq 0$$

$$100 + x_2 - x_5 \geq 0$$

$$x_i \in [10, 1000], \quad i = 1, \ldots, 5$$

The program above is still amenable to further inequality elimination as the new incidence matrix below indicates

Copyright © 1997-2006: Ike's Research Ltd

CONSTRAINT NO	X1	X2	X3	X4	X5
1	✓		✓		
2	✓	✓		✓	
3		✓			✓

Into constraint #1, #2 and #3 respectively, introduce slack variables ω_4, ω_5 and ω_6 and let:

$$z_4 \equiv x_3 = 400 - x_1 - \omega_4 \tag{3.7d}$$

$$z_5 \equiv x_4 = 400 + x_1 - x_2 - \omega_5 \tag{3.7e}$$

$$z_6 \equiv x_5 = 100 + x_2 - \omega_6 \tag{3.7f}$$

Rename the variables once more as follows:

OLD NAME	X1	X2	X3	X4	X5	-	-	-	-	-	-
Intermediate	-	-	Z4	Z5	Z6	ω_1	ω_2	ω_3	ω_4	ω_5	ω_6
NEW NAME	X1	X2	Z4	Z5	Z6	ω_1	ω_2	ω_3	ω_4	ω_5	ω_6

The final version of the program following the above operations is:

$$\min_{\mathbf{x}} J(\mathbf{x}) = z_1 + z_2 + z_3$$

$$\begin{aligned}
\text{subject to:}\quad & z_6 = 100 + x_2 - \omega_6 \\
& z_5 = 400 + x_1 - x_2 - \omega_5 \\
& z_4 = 400 - x_1 - \omega_4 \\
& z_3 = [\omega_3 + 1250000 - 2500x_2]/[z_6 - x_2] \\
& z_2 = [\omega_2 + 1250x_2 - 1250x_1]/[z_5 - x_1] \\
& z_1 = [\omega_1 + 833.33252x_1 - 83333.333]/[z_4 - 100] \\
& \omega \in \mathbf{R}_+ \quad x_i \in [10, 1000], \quad i = 1, 2
\end{aligned}$$

To formulate the equivalent bi-objective M-program as outlined in § 3.4.4 above, we need the gradient vector $\partial f / \partial \omega$. Using the chain rule of differential calculus, it is easy to show that:

$$\begin{aligned}
g_1 &\equiv \partial f / \partial \omega_1 = 1/(z_4 - 100) \\
g_2 &\equiv \partial f / \partial \omega_2 = 1/(z_5 - x_1) \\
g_3 &\equiv \partial f / \partial \omega_3 = 1/(z_6 - x_2) \\
g_4 &\equiv \partial f / \partial \omega_4 = (x_3 + 833.33252x_1 - 83333.333)/(z_4 - 100)^2 \\
g_5 &\equiv \partial f / \partial \omega_5 = (x_4 + 1250x_2 - 1250x_1)/(z_5 - x_1)^2 \\
g_6 &\equiv \partial f / \partial \omega_6 = (x_5 + 2500x_2 - 1250000x_1)/(z_6 - x_2)^2
\end{aligned}$$

The g-equations are implemented by **proc** `gradients`; the z-equations are implemented by **proc** `equations`. Note that the mutual dependency of the z-equations dictates that they are evaluated in the order presented above. And the implementation assumes that the ω_i are co-opted into the state vector as its final six components, hence the problem order declaration: **`#definecs`** `p_order` 8.

Copyright © 1997-2006: Ike's Research Ltd

Example 3.8: Dynamic Optimisation: The Harvest Problem [53]

$$\max_{\bar{u}} J(\bar{x}, \bar{u}) = \sum_{k=0}^{T-1} \sqrt{u_k}$$

Subject to: $x_{k+1} = a \cdot x_k - u_k$

$x_0 = x_T$

$x_k \in \mathbf{R}$; $u_k \in \mathbf{R}_+$

Data: $T = 5$; $x_0 = x_4 = 100$; $a = 1.1$.

I. GENO Output

Generation	*Objective Value*
0	12.555874
20	12.721036
40	12.721037
60	12.721038
80	12.721038
90	12.721038
100	12.721038

Optimal Control Vector: $\bar{\mathbf{u}}_4$ = (7.513148, 9.090909, 11.000000, 13.31000)

Optimal State Vector: $\bar{\mathbf{x}}_5$ = (100.000000, 102.486852, 103.644628, 103.009091, 100.000000)

Objective Function Value: $J(\mathbf{x}, \mathbf{u})$ = 12.721038

II. General Remarks

The exact solution to this problem is given by the formula (Michalewicz: 1994, p.108):

$$J^* = \sqrt{\frac{x_0 \cdot (a^n - 1)^2}{a^{n-1} \cdot (a - 1)}}.$$

Based on the parameter values given above, J* = 12.721038. But whereas GENO takes only 60 generations to achieve six-decimal point accuracy using a population of only 10, the algorithm of Michalewicz (whose performance is listed below) requires at least 10,000 and using a population of size 70. [54]

Generation	*Objective Value*
100	12.712700
1000	12.720600
10,000	12.721000
20,000	12.721000
30,000	12.721000
40,000	12.721038

Source	Generations	Best Vector Value	Best Function Value	CPU Time
Michalewicz	Over 40000	-	12.721038	-
GENO	60	See above	12.721038	-

[53] **Source**: Michalewicz (1994)

[54] Extracted from Table 6.3 of Michalewicz (1994, p.112).

Copyright © 1997-2006: Ike's Research Ltd

III. **The gep File**: Example 3.8

```
// A constrained uni-objective dynamic optimization problem
// Source: Michalewicz (1994, p.108).

#definecs p_maxgens  200
#definecs p_popsize  10
#definecs p_agents   1
#definecs p_order    1
#definecs p_plan     4

#include dynamic_gep_defaults.src

let vars[p_agents, p_order]   = 1;

let ff_states[p_order]  = 1;
let cc_states[p_order]  = 1;

adj_mode        = "s";
solution_type   = "e";
maximise        = true;

p_s_xover       = 0.55;
p_a_xover       = 0.55;
p_b_xover       = 0.00;
p_h_xover       = 0.55;
p_d_xover       = 0.55;
p_shuffle       = 0.55;

d_factor        = 0.80;
quantum_0       = 0.1;
rand_seed       = 240657;

proc (1) = m_rate(i,d); retp(0.05); endp;

proc (1) = bm_rate(d); retp(0.005); endp;

proc (2) = f(i,d,v_array);
local u,x,fv,k;
   u = matinit(order, plan, 0);
   x = matinit(order, horizon, 0);
  {u,x} = assign_sequences(i,d,u,x);[55]

   //evaluate objective
   fv = 0;
   for k (1, plan, 1);
      fv = fv + u[k]^0.5;
   endfor;
   if (maximise); fv = fv; else; fv = -fv; endif;
   retp (fv,v_array);
endp;

proc (1) = a(i,j,k); retp(1.1*state_fn(i,1,k)[56]); endp;

proc (1) = b(i,j,k); retp(-1.0); endp;

proc (1) = aa(z,w,j,k); retp(1.1*z[k]); endp;

proc (1) = bb(z,w,j,k); retp(-1.0); endp;

proc (1) = ee(z,w,j,k); retp((z[k+1] + w[k])/1.1); endp;
```

[55] Infra page 5, footnote 10.

[56] Infra page 26, footnote 39.

Copyright © 1997-2006: Ike's Research Ltd

VI. **The dat File**: Example 3.8

Table 3.8: Input Data for the Harvest Problem

VARIABLE NAME		X1
UCB		1e20
LCB		0
USB		1e20
LSB		-1e20
Initial State Vector		100
Final State Vector		100
Discrete Values		0
Inter-connexion Matrix	X1	1

V. **Points to Note**: Example 3.8

1. ff_states — This example illustrates the coding of a dynamic problem in which the state vector is fixed at both ends of the optimisation period. The matrix `ff_states` is a binary indicator matrix that shows which components of the state vector are fixed at the final time; in this particular example, the state happens to be one-dimensional.

2. state vector entries — In the dat file, one records the *actual values* of the state vector at initial and final time.

3. dynamics — **proc** `a(i,j,k)` and **proc** `b(i,j,k)` are direct implementations of the coefficient expressions of the dynamic constraints in MP1, i.e.:

$$A_k^i(\bar{\mathbf{x}}_k, \bar{\mathbf{u}}_{k-1}) \text{ and } B_k^i(\bar{\mathbf{x}}_k, \bar{\mathbf{u}}_{k-1}).$$

   proc `aa(z,w,j,k)` and **proc** `bb(z,w,j,k)` are replica implementations of the same coefficient expressions, except that the state and control variables arguments, `z[k]` and `w[k]` respectfully, are static boundary values only.

   proc `ee(z,w,j,k)` on the other hand is significantly different: it is the state equation expressed in "reverse", i.e. it gives $\bar{\mathbf{x}}_k$ in terms of $\bar{\mathbf{x}}_{k+1}$ and $\bar{\mathbf{u}}_k$. This is procedure is crucial to GENO's ability to solve two-point boundary value problems. This perhaps indicates one major limitation of the current version of GENO in that "factoring out" $\bar{\mathbf{x}}_k$ from the *A* and *B* expressions of MP1 may not always be easy. This process is necessary for writing the equation required by **proc** `ee(z,w,j,k)`.

4. `Accessing variables` — The *j*-th component of the state vector in the *i*-th chromosome at time *k* may be accessed by the procedure **proc** `state_fn(i,j,k)`; the corresponding control variable is **proc** `control_fn(i,j,k)`. The same variables may be accessed by `x[j,k]` and `u[j,k]` respectively, after making the assignment `{u,x} = assign_sequences(i,d,u,x)`.

5. `Algorithm Tuning` — As in most examples, the GENO was effectively tuned using only the parameters: `d_factor`, `quantum_0` and the return values of `m_rate`, and `bm_rate`.

Copyright © 1997-2006: Ike's Research Ltd

Example 3.9: Dynamic Optimisation: A Non-linear Resource Allocation Problem [57]

$$\max_{\bar{u}} J(\bar{x}, \bar{u}) = \prod_{k=1}^{T} (1 + k u_{k-1})$$

Subject to: $x_{k+1} = x_k - u_k$

$$\sum_{k=1}^{T} u_k = 8$$

$$x_k \in \mathbf{R}; \quad x_0 = 8; \quad u_k \in [0, 5]$$

Data: $T = 5$;

I. GENO Output

Generation	*Integer-valued Solution*	*Real-valued Solution*
0	1287.000000	1590.148000
20	3780.000000	4085.716747
40	3960.000000	4398.498099
60	4158.000000	4407.119360
80	4158.000000	4415.391047
100	4158.000000	4415.659845
120	4158.000000	4415.692284
140	4158.000000	4415.692855
160	4158.000000	4415.710917
180	4158.000000	4415.711192
200	4158.000000	4415.711867

Optimal Control Vector [Integer variables]: $\bar{\mathbf{u}}_4$ = (1.000000, 1.000000, 2.000000, 2.000000, 2.000000)

Optimal State Vector [Integer variables]: $\bar{\mathbf{x}}_5$ = (8.000000, 7.000000, 6.000000, 4.000000, 2.000000, 0.000000)

Objective Value [Integer variables]: $J(\mathbf{x}, \mathbf{u})$ = 4158.000000

Optimal Control Vector [Real variables]: $\bar{\mathbf{u}}_4$ = (1.056100, 1.556540, 1.723350, 1.806410, 1.857600)

Optimal State Vector [Real variables]: $\bar{\mathbf{x}}_5$ = (8.000000, 6.943900, 5.387360, 3.664010, 1.857600, 0.000000)

Objective Value [Real variables]: $J(\mathbf{x}, \mathbf{u})$ = 4415.711867

II. General Remarks

Larson and Casti (1978, p.182) compute the integer-valued solution using a "hand simulation" of Bellman's dynamic programming method[58] as:

Best Objective Function Value: $J(\mathbf{x}, \mathbf{u})$ = 4158

Optimal Control Sequence: $\bar{\mathbf{u}}_4$ = (1, 1, 2, 2, 2).

As can be seen above, GENO easily found this solution. The real-valued solution is also presented, and both may easily be validated by noting that the optimal control sequence satisfies the equality constraint in either case.

Source	Generations	Best Variable Value	Best Function Value	CPU Time
Larson and Casti	NA	See above	4158.000	-
GENO	200	See above	4415.712	-

[57] **Source**: Larson and Casti (1978, p.181)

[58] Bellman, R. (1957). *Dynamic Programming*. Princeton University Press, Princeton NJ

Copyright © 1997-2006: Ike's Research Ltd

III. **The gep File**: Example 3.9

```
// A constrained uni-objective dynamic optimization problem
// Source: Larson and Casti (1978, p.181).

#definecs p_maxgens  200
#definecs p_popsize  10
#definecs p_agents   1
#definecs p_order    1
#definecs p_plan     5

#include dynamic_gep_defaults.src

let vars[p_agents, p_order]   = 1;

let discrete_var[p_order]     = 1;

let ff_states[p_order]  = 0;
let cc_states[p_order]  = 1;

adj_mode       = "s";
solution_type  = "e";

p_s_xover      = 0.55;
p_a_xover      = 0.55;
p_b_xover      = 0.00;
p_h_xover      = 0.55;
p_d_xover      = 0.55;
p_shuffle      = 0.55;

d_factor       = 0.80;
quantum_0      = 0.1;
rand_seed      = 240657;

proc (1) = m_rate(i,d); retp(0.05); endp;

proc (1) = bm_rate(d); retp(0.005); endp;


proc (2) = f(i,d,v_array);
local u,x,fv,k;
   u = matinit(order, plan, 0);
   x = matinit(order, horizon, 0);
  {u,x} = assign_sequences(i,d,u,x);[59]

   //evaluate objective
   fv = 1.0;
   for k (1, plan, 1);
      fv = fv*(1 + k*u[k]);
   endfor;
   if (maximise); fv = fv; else; fv = -fv; endif;
   retp (fv,v_array);
endp;

proc (1) = a(i,j,k); retp(state_fn(i,1,k));[60] endp;

proc (1) = b(i,j,k); retp(-1.0); endp;

proc (1) = aa(z,w,j,k); retp(z[k]); endp;

proc (1) = bb(z,w,j,k); retp(-1.0); endp;
```

[59] Infra page 5, footnote 10.

[60] Infra page 26, footnote 39.

Copyright © 1997-2006: Ike's Research Ltd

```
proc (2) = dynamic_control_bounds(i,j,k,low,upp);
local n, fv, budget, stop_at;
   budget = 8;
   if(k == 1); fv = budget;
   else;
      fv = 0;
      stop_at = k - 1;
      for n (1, stop_at, 1);
         fv = fv + control_fn(i,j,n);[61]
      endfor;
      if(fv > budget); fv = 0.0;
      else; fv = budget - fv;
      endif;
   endif;
   upp = min(fv, upp);
   low = max( 0, low);
   retp (low,upp);
endp;
```

VI. **The dat File**: Example 3.9

Table 3.9a: Input Data for integer-valued solution

VARIABLE NAME		X1
UCB		5
LCB		0
USB		8
LSB		0
Initial State Vector		8
Final State Vector		0
Discrete Values		1
Inter-connexion Matrix	X1	1

Table 3.9b: Input Data for real-valued solution

VARIABLE NAME		X1
UCB		5
LCB		0
USB		8
LSB		0
Initial State Vector		8
Final State Vector		0
Discrete Values		0
Inter-connexion Matrix	X1	1

V. **Points to Note**: Example 3.9

1. solution control — The procedure **proc** `dynamic_control_bounds` evaluates the conjunction of the *static* control variable bounds (UCB and LCB) and the *dynamic* control bounds dictated by the given constraint,

$$\sum_{k=1}^{T} u_k = 8 .$$

2. discrete values — This example further illustrates GENO's capability on discrete-valued programs. On these type of programs, the type of entries made in the dat file and the indicator matrix `discrete_var` should not be confused: the former is always a 0-1 indicator matrix with a '1' entry for each discrete variable, and a '0' entry otherwise; the later should contain the actual discrete values for each discrete variable, and an entry of '0' for all non-discrete variables. Note that in this particular case, the discrete value is just happens to be 1.

3. real-valued solution — The real-valued solution has been included for comparison purposes, and perhaps as a challenge to other algorithms since, as far as I know, it is the best known solution.

[61] Infra page 26, footnote 39.

Copyright © 1997-2006: Ike's Research Ltd

Example 3.10: Multi-objective Static Optimisation: Oligopolist Market Equilibrium Problem [62]

$$\underset{q}{\text{Opt}}\, \mathbf{J}(\boldsymbol{q}) = (J^1, J^2, \cdots, J^5)^{\mathrm{T}}$$

Subject to:

$$J^i(\boldsymbol{q}) = q_i\, p(\boldsymbol{q}) - f_i(q_i)\,; \quad f_i(q_i) = c_i q_i + \frac{(q_i)^{(1+\alpha_i)}}{(1+\alpha_i)(K_i)^{\alpha_i}}\,; \quad p(\boldsymbol{q}) = \left(\frac{5000}{\boldsymbol{Q}}\right)^{\beta}\,; \quad \boldsymbol{Q} = \sum_{i=1}^{n} q_i\,;\ i = \{1,2,\ldots,5\}$$

Problem-specific Data:

Firm	c_i	K_i	α_i	β
1	10	5	1/1.2	1/1.1
2	8	5	1/1.1	1/1.1
3	6	5	1.00	1/1.1
4	4	5	1/0.9	1/1.1
5	2	5	1/0.8	1/1.1

I. GENO Output

Generation	q_1^*	q_2^*	q_3^*	q_4^*	q_5^*
0	36.100000	63.500000	40.500000	40.600000	21.700000
20	36.930000	41.810000	43.710000	42.660000	39.180000
60	36.932500	41.818160	43.706590	42.659240	39.178900
80	36.932511	41.818141	43.706578	42.659240	39.178952
90	36.932511	41.818141	43.706579	42.659240	39.178953
100	36.932511	41.818141	43.706579	42.659240	39.178953

Equilibrium Solution: $\boldsymbol{q}^* = (36.932511, 41.818141, 43.706578, 42.659240, 39.178952)^{\mathrm{T}}$

II. General Remarks

This example was originally formulated by Murphy, et al. (1982) and has since been numerically solved by Harker (1984), Jörnsten (1991), as well as Kolstad and Mathiesen (1991). To provide a benchmark against which GENO may be compared to these other algorithms, a solution that is accurate to 18 decimal places was found (by solving the system of non-linear equations arising from the Karush-Kuhn-Tucker conditions) using the *FindRoot* facility of *Mathematica*. The results below clearly show that GENO computes the most accurate solution.

q_i	Benchmark	Murphy, *et al.*	Harker	Jörnsten	Kolstad, *et al.*	GENO
1	36.932510815735757481	36.9120	36.93180	36.9300	36.9350	36.932511
2	41.818141660437635128	41.8200	41.81755	41.8200	41.8182	41.818141
3	43.706578522274216542	43.7050	43.7060	43.7100	43.7066	43.706578
4	42.659239743305114839	42.6650	42.6588	42.6600	42.6593	42.659240
5	39.178952516625022418	39.1820	39.1786	39.1800	39.1790	39.178952

[62] **Source**: Murphy, Sherali and Soyster (1982).

Copyright © 1997-2006: Ike's Research Ltd

III. **The gep File**: Example 3.10

```
// A Nash solution of a multi-objective unconstrained static optimization problem
// Source: Murphy, Sherali and Soyster (1982)

#definecs p_maxgens  150
#definecs p_popsize  10
#definecs p_agents   5
#definecs p_order    5
#definecs p_plan     1

#include static_gep_defaults.src

let vars[p_agents,p_order] = {
               1   0   0   0   0,   //variables controlled by agent 1
               0   1   0   0   0,   //variables controlled by agent 2
               0   0   1   0   0,   //variables controlled by agent 3
               0   0   0   1   0,   //variables controlled by agent 4
               0   0   0   0   1    //variables controlled by agent 5
               };

adj_mode          = "s";
solution_type     = "e";
pos_orth          = true;
view_vars         = true;

p_s_xover         = 0.00;
p_a_xover         = 0.55;
p_b_xover         = 0.00;
p_h_xover         = 0.55;
p_d_xover         = 0.55;
p_shuffle         = 0.55;

d_factor          = 0.80;
quantum_0         = 0.1;
rand_seed         = 240658;

proc (1) =  m_rate(i,d); retp(0.05); endp;

proc (1) =  bm_rate(d); retp(0.005); endp;


//problem-specific data
let c_array[p_agents] = 10   8   6   4   2;
let k_array[p_agents] = 5    5   5   5   5;
let a_array[p_agents] = 0.83333333  0.90909091  1.00000000  1.11111111  1.25000000;
let b_array[p_agents] = 0.90909091  0.90909091  0.90909091  0.90909091  0.90909091;


proc (2) = f(i,d,v_array);
local u,x,fv,fv1,fv2,k;
   k = horizon;
   u = matinit(order, plan, 0);
   x = matinit(order, horizon, 0);
  {u,x} = assign_sequences(i,d,u,x);[63]

   //evaluate objectives
   fv1 = c_array[d]*x[d,k] + \
        (x[d,k]^(1 + a_array[d]))*(k_array[d]^(-a_array[d]))/(1 + a_array[d]);
   fv2 = x[d,k]*coupler(d,i);
   fv  = fv2 - fv1;
   if (maximise); fv =  fv; else; fv = -fv; endif;
   retp (fv, v_array);
endp;
```

[63] Infra page 5, footnote 10.

Copyright © 1997-2006: Ike's Research Ltd

```
proc (1) = coupler(d,i);
local u,x,fv,que,j,k;
   que = 0;
   k = horizon;
   u = matinit(order, plan, 0);
   x = matinit(order, horizon, 0);
  {u,x} = assign_sequences(i,d,u,x);[64]
   for j (1, order, 1);
      que = que + x[j,k];
   endfor;
   fv = (5000^b_array[d])*(que^(-b_array[d]));
   retp (fv);
endp;
```

VI. **The dat File**: Example 3.10

Table 3.10: Input Data for the Market Equilibrium Problem

VARIABLE NAME		X1	X2	X3	X4
UCB		100	100	100	100
LCB		1	1	1	1
USB		100	100	100	100
LSB		1	1	1	1
Initial State Vector		100	83	13	3
Final State Vector		0	0	0	0
Discrete Values		0	0	0	0
Inter-connexion Matrix	X1	1	0	0	0
	X2	0	1	0	0
	X3	0	0	1	0
	X4	0	0	0	1

V. **Points to Note**: Example 3.10

1. `view_vars` — For this example, the output is best viewed as variables values, hence set: `view_vars = true`.

2. `coupler` — This is merely an auxiliary procedure for evaluating the objective function.

[64] Infra page 5, footnote 10.

Copyright © 1997-2006: Ike's Research Ltd

Example 3.11 Multi-objective Static Optimisation: The Euclidean Compromise Solution I [65]

$$\underset{\mathbf{x}}{\text{Opt}}\ \mathbf{f}(\mathbf{x}) = \{f_1(\mathbf{x}), f_2(\mathbf{x})\}$$

Subject to: $x \in [-6,\ 6]$

Where: $f_1(\mathbf{x}) = x^2$

$f_2(\mathbf{x}) = (x-2)^2$

I. GENO Output

Generation	Objective [1]	Objective [2]
0	0.360000	1.960000
10	1.000000	1.000000
20	1.000000	1.000000
40	1.000000	1.000000
80	1.000000	1.000000
100	1.000000	1.000000

Solution Vector: $\mathbf{x}$ = 1.000000

Objective Function Value: $\mathbf{f}(\mathbf{x})$ = (1.000000, 1.000000) ^{T}

II. General Remarks

Preamble

For a multiple objective optimisation problem, the solution ideally ought to be a win-win situation in which each individual objective function attains its best possible value within the constraints of the problem. But, as the name implies, the *ideal solution* is usually unattainable; all one can hope for is to numerically approach such a point as closely as possible. The *Euclidean compromise* is a solution concept for the multi-objective optimisation problem that is based on the notion of 'being close to ideal'.[66] The Euclidean compromise solution (ECS) is that point on the Pareto frontier that is closest to the ideal solution, as measured by the Euclidean distance metric. This example serves to illustrate the effectiveness of GENO at computing the ECS of a multiple objective optimisation problem. The example is simple enough to afford an analytical determination of the ECS, against which the performance of GENO may be measured.

The Euclidean Compromise Solution: An Analytical Approach

The location of the ideal solution in the space of outcomes may be determined as follows. The ECS is a point that lies on the Pareto frontier, which is itself the set of points that minimise the auxiliary objective:

$$J(\mathbf{x}) = \alpha f_1(\mathbf{x}) + (1-\alpha) f_2(\mathbf{x}), \quad \alpha \in [0, 1]. \tag{3.11a}$$

At a minimum of (3.11a), the following first order optimality condition holds:

$$\frac{\partial J(\mathbf{x})}{\partial x} = 2\alpha x + 2(1-\alpha)(x-2) = 0 \quad \Leftrightarrow \quad x = 2 - 2\alpha \tag{3.11b}$$

Using elementary interval arithmetic (Kearfott, 1996), one can easily show the following logical implications.

[65] **Source**: Marco, Désidéri and Lanteri (1999). This is one of the test problems that Schaffer (1984) used to evaluate his multi-objective genetic algorithm called VEGA. His simple unconstrained two-objective functions comprised *the* test suite for validating most of the evolutionary multi-objective optimisation techniques developed in the ensuing years (Coello Coello, 2001, p.4).

[66] There are of course other notions of what constitutes a solution to the multi-objective optimisation problem, especially when viewed from the game theory perspective. In this regard, the Nash equilibrium concept is pervasive, and the ability of GENO at computing the Nash solution has been demonstrated elsewhere (see Example 3.8). The point of this example is to show that GENO is equally capable of computing solutions that one could associate with another major game-type, i.e. 'the bargaining game' (see concluding comments below).

Copyright © 1997-2006: Ike's Research Ltd

Given that: $\alpha \in [0, 1]$ then $x \in [0, 2]$ (3.11c)

hence: $f_1 \in [0, 4]$ and $f_2 \in [0, 4]$ (3.11d)

In this example, the operator 'Opt' denotes minimisation, hence the *ideal solution* (in the space of outcomes) is clearly the point (0, 0). Given a candidate solution, **x***, its Euclidean distance in the space of outcomes from the ideal solution is:

$$R(\mathbf{x}) = \sqrt{\left(f_1^2(\mathbf{x}) + f_2^2(\mathbf{x})\right)} = \sqrt{x^4 + (x-2)^4} \ . \tag{3.11e}$$

This distance is minimal when the following equation holds:

$$\partial R(\mathbf{x}) / \partial x = 0 \quad \Leftrightarrow \quad x^3 + (x-2)^3 = 0 \ . \tag{3.11f}$$

The *only* point on the Pareto frontier — whose domain is $x \in [0, 2]$ — that satisfies condition (3.11f) is $x = 1$, and at this 'Euclidean compromise solution', we have that:

$$f_1(1) = 1 \quad \text{and} \quad f_2(1) = 1 \tag{3.11g}$$

As can be seen from the results above, GENO easily finds this solution.

Concluding Remarks

The Euclidean Compromise Solution is a member of a class of solutions first suggested by Yu (1973). It is closely related to other single-point solutions of the so-called bargaining problem in the theory of games in that (a) they are all specific points on the Pareto-efficient frontier of multi-objective optimisation problems, (b) they are defined relative to some auxiliary points—the *ideal solution* in the case of ECS, and the *disagreement point* (or both) in the bargaining context.

The most common solution concepts in the bargaining context are the Nash Bargaining Solution, Kalai-Smorodinsky Solution and the Maschle-Perles Solution.[67] Though one can numerically compute the former directly by using standard non-linear programming algorithms, the same cannot be said of the others.[68] In any case, the use of non-linear programming methods relies on a weighted-sum representation of the Pareto frontier. This can be problematic because, as is well known, the weighted-sum approach is incapable of generating points from all parts of the Pareto set in all cases (Das and Dennis, 1997).

In contrast, GENO does not require "scalarization" of the objective vector. The algorithm is therefore capable of converging to any point on the Pareto frontier that is sufficiently characterised with respect to the ideal outcome. As can be seen from the results presented above, it converges to the Euclidean compromise solution within ten generations! Furthermore, provided the components of the solution vector are properly scaled in outcome space, the ECS as generated by GENO will always exhibit a "middling characteristic",[69] which is intuitively what is desired of a compromise solution.

---

[67] See the survey paper by Rosenmüller and Trockel (2001).

[68] Admittedly, bargaining solutions may be determined indirectly by formulating a fictitious non co-operative game whose Nash equilibrium coincides with the said solution. This research agenda is commonly called 'The Nash Program' and is closely related to another area of research called 'Mechanism Theory'; for details, see e.g. Trockel (1999).

[69] The term 'middling' was brought to the multi-objective programming lexicon by Schaffer (1984) who noted that his VEGA algorithm tended to produce solutions that excelled on one criterion, but performed poorly on others. He argues that what is desirable of a compromise solution is that it should have acceptable performance simultaneously on all criteria, i.e. a "middling performance".

Copyright © 1997-2006: Ike's Research Ltd

III. **The gep File**: Example 3.11

```
// A Pareto solution of a multi-objective unconstrained static optimization problem
// Source: Marco, Désidéri and Lanteri (1999)

#definecs p_maxgens     200
#definecs p_popsize     20
#definecs p_agents      2
#definecs p_order       1
#definecs p_plan        1

#include static_gep_defaults.src


let vars[p_agents, p_order]   = {1,     // variables controlled by agent 1
                                 1}     // variables controlled by agent 2

let i_point[p_agents]   = 0   0;

adj_mode        = "g";
solution_type   = "i";
maximise        = false;
proximity       = true;

p_s_xover    = 0.00;
p_a_xover    = 0.55;
p_b_xover    = 0.00;
p_h_xover    = 0.55;
p_d_xover    = 0.55;
p_shuffle    = 0.55;

d_factor     = 0.250;
quantum_0    = 0.1;
rand_seed    = 2406525;

proc (1) = m_rate(i,d); retp(0.05); endp;

proc (1) = bm_rate(d); retp(0.005); endp;


proc (2) = f(i, d, v_array);
local j,u,x,fv,fv1,fv2,k;
   k = horizon;
   u = matinit(order, plan, 0);
   x = matinit(order, horizon, 0);
  {u,x} = assign_sequences(i,d,u,x);[70]

   //evaluate objectives
   if(d == 1);      fv = x[1,k]^2;
   elseif(d == 2); fv = (x[1,k] - 2)^2;
   endif;
   if (maximise); fv =  fv; else; fv = -fv; endif;
   retp (fv, v_array);
endp;
```

[70] Infra page 5, footnote 10.

Copyright © 1997-2006: Ike's Research Ltd

VI. **The dat File**: Example 3.11

Table 3.11: Input Data for the Euclidean Compromise Problem I

VARIABLE NAME		X1
UCB		1e20
LCB		-1e20
USB		6
LSB		-6
Initial State Vector		0
Final State Vector		0
Discrete Values		0
Inter-connexion Matrix	X1	1

V. **Points to Note**: Example 3.11

1. i_point — The matrix `i_point` specifies the coordinates of the 'ideal solution' in the space of objective function values (see 'General Remarks' above).

2. adj_mode — The `adj_mode` parameter in this case should be set to `"g"`. This effectively switches GENO to a "group rational" mode of operation in the quest for a multi-objective solution to the problem.[71]

3. `solution_type` — The `solution_type` parameter in this case should be set to "i" (for 'ideal solution').

[71] For expositional purposes, let each component function of the vector $\mathbf{J}$ in MP1 be the concern of a decision-maker or player, and let each player derive some utility from values attained by his respective 'outcome' or 'payoff' function. Also, assume the pth player controls a subset, $\bar{\mathbf{x}}^{p}_{T+1} \in \mathbf{X}^{p_1}_{0} \times \cdots \times \mathbf{X}^{p_N}_{T+1}$, of the state vector via a set of decisions or actions, $\bar{\mathbf{u}}^{p}_{T} \in \mathbf{U}^{p_1}_{0} \times \cdots \times \mathbf{U}^{p_N}_{T}$.[71] Then it is immediately clear that MP1, in all its guises, can be treated essentially as a dynamic game. One may therefore borrow some relevant concepts and analytical tools from the theory of games in order to deal with the general multi-criteria problem and vice versa. And as noted above, the 'Euclidean compromise' is one of a class of solutions for the multi-agent optimisation problem that are based on Vilfredo Pareto's notion of 'efficiency', a typical explanation of which may be found in e.g. Fonseca and Fleming (1995). In Game-theoretic terms, a Pareto-efficient solution has the property that no player can improve his outcome (i.e. objective function value) without worsening the situation of the other player. It formalises what is called *group rationality* since it would be irrational for the players as a group to settle for a solution that is *not* Pareto-efficient: by moving to an efficient solution (in the sense of Pareto), the outcome for at least one player can improve without damaging the situation for the other.

Copyright © 1997-2006: Ike's Research Ltd

Example 3.12 Multi-objective Static Optimisation: The Euclidean Compromise Solution II [72]

$$\underset{\mathbf{z}}{\text{Opt}}\ \mathbf{f}(\mathbf{z}) = \{f_1(\mathbf{z}), f_2(\mathbf{z})\}$$

$$\text{subject to: } f_1(\mathbf{z}) = (x-1)^2 + (y-3)^2;\quad f_2(\mathbf{z}) = (x-4)^2 + (y-2)^2$$
$$\mathbf{z} = (x, y)^T \in [-5, 5] \times [-5, 5]$$

I. GENO Output

Generation	Objective [1]	Objective [2]
0	1.730000	7.930000
10	2.500000	2.500000
20	2.500000	2.500000
40	2.500000	2.500000
80	2.500000	2.500000
100	2.500000	2.500000

Solution Vector: **z** = (2.500000, 2.500000) ^{T}

Objective Function Value: **f** (**z**) = (2.500000, 2.500000) ^{T}

II. General Remarks

The Pareto frontier is that set of points that minimise the auxiliary objective:

$$J(\mathbf{z}) = \alpha f_1(\mathbf{z}) + (1-\alpha) f_2(\mathbf{z}), \quad \alpha \in [0, 1]. \tag{3.12a}$$

The first order optimality conditions are:

$$\partial J(\mathbf{z})/\partial x = 0 \quad \Leftrightarrow \quad x = 4 - 3\alpha;\ \partial J(\mathbf{z})/\partial y = 0 \quad \Leftrightarrow \quad y = 2 + \alpha \tag{3.12b}$$

Given that $\alpha \in [0, 1]$, then $x \in [1, 4]$, $y \in [2, 3]$, and hence $f_1 \in [0, 10]$, $f_2 \in [0, 10]$. In this example, the operator 'Opt' denotes minimisation, therefore the *ideal solution* in the space of outcomes is the point (0, 0). Given a candidate solution, **z***, its Euclidean distance from the *ideal point* (in the space of outcomes) is:

$$R(\mathbf{z}) = \sqrt{\left(f_1^2(\mathbf{z}) + f_2^2(\mathbf{z})\right)} = \sqrt{[(x-1)^2 + (y-3)^2]^2 + [(x-4)^2 + (y-2)^2]^2}. \tag{3.12c}$$

This distance is least when the following holds:

$$(\partial R(\mathbf{z})/\partial x = 0) \wedge (\partial R(\mathbf{z})/\partial y = 0) \quad \Leftrightarrow \quad x + 3y = 10. \tag{3.12d}$$

There are many points that satisfy condition 3.12d but by evaluating 3.12c at points near the candidate solution, **z*** = (2.5, 2.5)^T that are also on the frontier,[73] one may verify that **z*** is indeed a minimum point, viz.:

	Z^T = (2.53, 2.49)	Z^T = (2.47, 2.51)	Z^T = (2.5, 2.5)	Z^T = (2.49, 2.503)	Z^T = (2.51, 2.497)
f_1	2.601	2.401	2.5	2.467	2.533
f_2	2.401	2.601	2.5	2.533	2.467
Distance Function	3.53977	3.53977	3.53553	3.53592	3.53592

The point **z*** = (2.5, 2.5)^T is thus the Euclidean compromise solution, at least up to the second decimal place. And as can be seen, GENO easily computes this solution.

[72] **Source**: Marco, Désidéri and Lanteri (1999).

[73] Note that for the 'new point' to be on the Pareto-efficient frontier, the perturbations in the variables must satisfy $\delta x + 3\delta y = 0$.

Copyright © 1997-2006: Ike's Research Ltd

III. **The gep File**: Example 3.12

```
// A Pareto solution of a multi-objective unconstrained static optimization problem
// Source: Marco, Désidéri and Lanteri (1999)

#definecs p_maxgens     200
#definecs p_popsize     20
#definecs p_agents      2
#definecs p_order       2
#definecs p_plan        1

#include static_gep_defaults.src

let vars[p_agents, p_order]   = {1  1,     // variables controlled by agent 1
                                 1  1}     // variables controlled by agent 2

let i_point[p_agents]   = 0   0;

adj_mode        = "g";
solution_type   = "i";
maximise        = false;
proximity       = true;

p_s_xover    = 0.00;
p_a_xover    = 0.55;
p_b_xover    = 0.00;
p_h_xover    = 0.55;
p_d_xover    = 0.55;
p_shuffle    = 0.55;

d_factor     = 0.250;
quantum_0    = 0.1;
rand_seed    = 240658;

proc (1) = m_rate(i,d); retp(0.05); endp;

proc (1) = bm_rate(d); retp(0.005); endp;

proc (2) = f(i, d, v_array);
local j,u,x,fv,fv1,fv2,k;
   k = horizon;
   u = matinit(order, plan, 0);
   x = matinit(order, horizon, 0);
  {u,x} = assign_sequences(i,d,u,x);[74]

   //evaluate objectives
   if(d == 1);      fv = (x[1,k] - 1)^2 + (x[1,k] - 3)^2;
   elseif(d == 2); fv = (x[1,k] - 4)^2 + (x[1,k] - 2)^2;
   endif;
   if (maximise); fv =  fv; else; fv = -fv; endif;
   retp (fv, v_array);
endp;
```

[74] Infra page 5, footnote 10.

Copyright © 1997-2006: Ike's Research Ltd

VI. **The dat File**: Example 3.12

Table 3.12: Input Data for the Euclidean Compromise Problem II

VARIABLE NAME		X1	X2
UCB		5	5
LCB		-5	-5
USB		5	5
LSB		-5	-5
Initial State Vector		0	0
Final State Vector		0	0
Discrete Values		0	0
Inter-connexion Matrix	X1	1	0
	X2	0	1

V. **Points to Note**: Example 3.12

1. i_point — The matrix `i_point` specifies the coordinates of the 'ideal solution' in the space of objective function values (see 'General Remarks' above).

2. adj_mode — The `adj_mode` parameter in this case should be set to `"g"`. This effectively switches GENO to a "group rational" mode of operation in the quest for a multi-objective solution to the problem.

3. `solution_type` — The `solution_type` parameter in this case should be set to "i" (for 'ideal solution').

Copyright © 1997-2006: Ike's Research Ltd

Example 3.13 Multi-objective Dynamic Optimisation: Efficient Portfolio Selection [75]

$$\underset{\bar{\mathbf{u}}}{\text{Opt}}\ J(\bar{\mathbf{x}}_{T+1}, \bar{\mathbf{u}}_T) = \{ f_1(\bar{\mathbf{u}}_T),\ f_2(\bar{\mathbf{u}}_T) \}$$

Subject to: $x_{k+1} = x_k - u_k$

$$\sum_{k=1}^{T} u_k = 1$$

$$x_1 = 1; \quad x_{T+1} = 0; \quad u_k \in [0,1]; \quad T = 6$$

Where: $f_1(\bar{\mathbf{u}}_T) = \sum_{k=1}^{T} \bar{r}_k u_k$; $f_2(\bar{\mathbf{u}}_T) = \sum_{t=1}^{T} \sum_{k=1}^{t-1} u_t u_k \rho_{tk} \sigma_t \sigma_k$

Other data:

	Merrill Lynch	Paine Webber	Digital Equipment	Microsoft	Silicon Graphics	Hewlett-Packard
Mean Return	15.852	14.262	31.336	25.775	50.228	14.842
Std Deviation	37.215	41.773	33.165	62.009	60.720	23.757

Correlation Matrix	Merrill Lynch	Paine Webber	Digital Equipment	Microsoft	Silicon Graphics	Hewlett-Packard
Merrill Lynch	1.000	0.944	0.146	0.231	0.379	0.258
Paine Webber	0.944	1.000	0.109	0.239	0.413	0.223
Digital Equipment	0.146	0.109	1.000	-0.169	-0.229	0.691
Microsoft	0.231	0.239	-0.169	1.000	0.882	-0.256
Silicon Graphics	0.379	0.413	-0.229	0.882	1.000	-0.284
Hewlett-Packard	0.258	0.223	0.691	-0.256	-0.284	1.000

I. GENO Output

Generation	*Portfolio Return*	*Standard Deviation*
0	33.588800	26.346978
20	37.483200	29.805179
60	37.483200	29.805179
80	37.483200	29.805179
100	37.483200	29.805179

	Merril Lynch	Paine Webber	Digital Equip	Microsoft	Silicon Graph	Hewlett-Packard	
Allocation $\bar{\mathbf{u}}_6$:	0.000000	0.000000	0.300000	0.000000	0.500000	0.200000	
Investment Fund $\bar{\mathbf{x}}_7$:	1.000000	1.000000	1.000000	0.700000	0.700000	0.200000	0.000000

Portfolio Return: 37.483200

Standard Deviation: 29.805179

[75] **Source**: The portfolio selection model is due to Markowitz (1952).

Copyright © 1997-2006: Ike's Research Ltd

II. General Remarks

Part I: On the Markowitz Investment Model

The Markowitz model[76] is concerned with how a rational investor in the capital markets *would* select and invest his wealth in financial securities. If the individual has a utility function *U* and there are *N* securities with random single-period returns r^i available for investing in, the Markowitz portfolio selection model has it that the investor's optimal portfolio should be the solution to the programming problem:

MP13: $$\max_x Z_1(\mathbf{x}) = \mathbf{E}\{U(w_0, r^i, x^i)\}$$

Subject to: $x^i \geq 0$

$$\sum_i x^i = 1$$

where:

- Z_1 is the investor's expected utility;
- $\mathbf{E}$ is the mathematical expectation operator;
- w_0 is the investor's initial wealth;
- x^i is the proportion of wealth allocated to the *i*-th asset;
- r^i is the rate of return of the *i*-th asset.

If the investor has a negative exponential utility function for wealth, and the returns are joint-normally distributed, the expected utility maximisation problem above is equivalent to the Mean-Variance optimisation problem given below:[77]

MP14: $$\max_x Z_2(\mathbf{x}) = \sum_i x^i r^i - \varphi^{-1} \sum_i \sum_{j<i} x^i x^j c^{ij}$$

Subject to: $x^i \geq 0$

$$\sum_i x^i = 1$$

where:

- r^i is the expected return for the *i*-th asset;
- c^{ij} is the covariance of returns between the *i*-th and *j*-th assets;
- x^i is the proportion of wealth invested in the *i*-th asset;
- φ is the investor's risk tolerance.

Even if the investor had any other utility function (other than the negative exponential) and/or the returns were not joint-normally distributed, Levy and Markowitz (1979) provide empirical evidence which shows that the mean-variance maximand Z_2 is a good approximation of Z_1 for a wide variety of utility functions.

Historically, the Mean-Varience model MP14 has usually been solved by either selecting and fixing a risk level and then calculating the best achievable portfolio return; or fixing the portfolio return at some arbitrary level and then searching for a feasible portfolio of minimum variance that has this rate of return. In either case, one normally employs optimization methods to compute the solution. For example, the latter approach would involve solving the M-program:

[76] A brief review of the model is in Markowitz (1991) and Sharpe (1991). Some management consulting firms such as Marakon Associates and Strategic Planning Associates have in fact proposed employing the equilibrium version of the Markowitz model (the Capital Asset Pricing Model or CAPM) not only as a decision making tool for investors with a portfolio of financial assets, but also as a planning tool for corporations that manage a portfolio of businesses, SBUs, etc. (Naylor and Tapon: 1982, p.1166).

[77] For a recent review of the Mean-Variance reformulation of Markowitz's normative model, see e.g. Sharpe (1991).

Copyright © 1997-2006: Ike's Research Ltd

MP15: $$\min_{x} Z_3(\mathbf{x}) = \sum_i \sum_{j<i} x^i x^j c^{ij}$$

Subject to: $x^i \geq 0$

$$\sum_i x^i r^i = \bar{r}$$

$$\sum_i x^i = 1$$

where:

$\bar{r}$	is the required portfolio return;
r^i	is the expected return for the *i*-th asset;
c^{ij}	is the covariance of returns between the *i*-th and *j*-th assets;
x^i	is the proportion of wealth invested in the *i*-th asset;
φ	is the investor's risk tolerance.

The method of solution employed by GENO in Example 3.13 is clearly different.[78] The re-formulation of the Markowitz model as a bi-objective dynamic optimisation problem exploits the fact that GENO can easily compute the Euclidean compromise solution — a point that is on the efficient frontier of the portfolio and is closest to the ideal outcome. Since, by convexity of the minimum-variance set, the portfolio return cannot exceed the maximum of the individual returns of the assets comprising the portfolio;[79] since, by logic, the typical investor can rationally be expected to exhibit non-satiated risk-averse behaviour;[80] and since, by design, GENO converges to the Pareto-efficient frontier, the ideal outcome should always be stated as comprising the maximum available return (i.e. Silicon Graphics in the example above), and the maximum risk (as measured by the portfolio standard deviation) that the investor can bear.

Better still the problem should be re-scaled such that, (for any *rational* investor), the ideal solution *is* the origin of the outcome space. In this particular case for example, instead of seeking to *maximise* the portfolio return, one might consider *minimising* the reciprocal of the portfolio return.[81] That is, one considers the model:

MP16: $$\underset{\bar{\mathbf{u}}}{\text{Opt}}\ J(\bar{\mathbf{x}}_{T+1}, \bar{\mathbf{u}}_T) = \{ f_1(\bar{\mathbf{u}}_T),\ f_2(\bar{\mathbf{u}}_T) \}$$

Subject to: $x_{k+1} = x_k - u_k$

$$\sum_{k=1}^{T} u_k = 1$$

$$x_1 = 1; \quad x_{T+1} = 0; \quad u_k \in [0,1]; \quad T = 6$$

Where: $$f_1(\bar{\mathbf{u}}_T) = \sum_{k=1}^{T} (\bar{r}_k)^{-1} u_k\ ; \quad f_2(\bar{\mathbf{u}}_T) = \sum_{t=1}^{T} \sum_{k=1}^{t-1} u_t u_k \rho_{tk} \sigma_t \sigma_k$$

Clearly, logic dictates that the ideal outcome for MP16 is (0, 0).

[78] I am unaware of a re-formulation of the Markowitz model that is similar to the bi-objective *pseudo-dynamic* optimisation model presented here, although latterly Armañanzas and Lozano (2005), and Streichert, et al. (2003) have suggested bi-objective static optimisation models that are somewhat similar.

[79] See e.g. Luenberger (1998, Chap.6).

[80] Ibid.

[81] This approach is valid provided the returns are *positive*.

Copyright © 1997-2006: Ike's Research Ltd

II. General Remarks

Part II: On Robust Optimisation

Preamble

All real-life optimal design or control problems involve some system parameters whose true values can never be known precisely due to measurement errors, extraneous random noise, or some other technical difficulties. And yet these parameters must be assigned some numerical values before optimisation techniques can be applied to generate solutions. In many applications, good estimates of system parameters are all that is needed — the ensuing variations in the "optimal solution" induced by the uncertainty in parameter values being within acceptable margins. But there exist instances where accounting for "system shocks" may be critical, and to this end, a research agenda called *Robust Optimisation* has recently emerged to address the issue. The following is a presentation of a decision model (inspired by Markowitz's Portfolio Theory) that may be used to design robust policies for the control of national economies, and on which GENO's EC solution approach may be applied.

Robust Economic Policy Design

Most *realistic* economic models involve some stochastic variables. However, for purposes of formulating practical policies, the computational difficulties associated with stochastic models, coupled with other conceptual considerations strongly favour the adoption of a suitable deterministic approximation. A standard approach in this regard is to assume the deterministic model that results when all the random variables in the stochastic model are fixed at their most likely values. This procedure is completely adequate for *LQG* problems:[82] in this case, a separation of the problem into a deterministic part and a stochastic part can easily be derived, and the *Certainty Equivalence Principle* applies.[83]

Unfortunately, the certainty equivalence principle cannot always be applied in the general non-LQG case. To partly compensate for this therefore, one needs to formulate the deterministic approximation such that the solutions it generates would, to some extent, be unaffected by perturbations in the system. Such solutions are termed *robust* and the need for robust solutions in applied policy-design work is evidently crucial because policy-makers are, in general, poorly informed about prospective disturbances that might impinge on the system, and it is therefore desirable to design policies that would perform reasonably well under different types of possible shocks. This requirement becomes even more important when it is considered that often the solutions proposed are of the *open-loop* type, and hence the disturbance-rejection advantage of feedback is not available. Accordingly, Rustem (1989) proposes a technique for generating robust economic policies based on Control Theory's notion of *sensitivity* (for an alternative approach, see Zhang (2005) and references cited therein).

The Sensitivity Approach

Consider the single-agent stochastic policy optimisation problem below.

MP17: $$\min_{\mathbf{U}} \ \mathbf{E}\left\{ J(\mathbf{Z},\mathbf{U}) \,\middle|\, \mathrm{F}(\mathbf{Z},\mathbf{U},\varepsilon)=\mathbf{0};\ \mathrm{G}(\mathbf{Z},\mathbf{U})\le\mathbf{0} \right\}$$

where: $\mathbf{E}$ denotes mathematical expectation;

ε is the vector of random disturbances;

$\mathbf{Z}$ is the vector of endogenous variables;

$\mathbf{U}$ is the vector of decision variables;

J is the policy objective function;

F is a vector function denoting the model equations;

G is a vector function of the constraints imposed on **Z** and **U**.

---

[82] These are problems where one seeks to optimise a *Q*uadratic objective functional subject to *L*inear constraints with additive random disturbances drawn from a *G*aussian distribution.

[83] The Certainty Equivalent Principle states that, when optimising under uncertainty, one simply adopts the policy rule which is optimal for the deterministic part. The principle is *universally* applicable in LQG problems; and where it applies, a useful result is that the optimal rule is universal, i.e. independent of the initial conditions of the problem and the disturbances in the model (Currie and Levine: 1993, p.98)

Copyright © 1997-2006: Ike's Research Ltd

In order to generate optimum policies that are also robust, Rustem (1989) suggests appending a sensitivity measure to the original objective function and then solving a deterministic version of the problem. The basic idea of the sensitivity technique is to re-formulate the problem such that the sensitivity term so introduced is optimised simultaneously with the original objective function. Two types of sensitivity measures may be considered: (a) the sensitivity of the optimal objective function with respect to random disturbances; this is given by the gradient vector $J_\varepsilon = \partial J / \partial \varepsilon$; (b) the sensitivity of the optimum path of the endogenous variables with respect to random disturbances; this is similarly defined as the *matrix* $\mathbf{Z}_\varepsilon = \partial \mathbf{Z}_\varepsilon / \partial \varepsilon$.

In terms of sensitivity measure (a), robust policies may be obtained by solving the deterministic problem:

MP18: $$\min_{\mathbf{U}} \left\{ \bar{J}(\mathbf{Z},\mathbf{U},\alpha) \,\middle|\, \mathrm{F}(\mathbf{Z},\mathbf{U},\mathbf{0}) = \mathbf{0};\ \mathrm{G}(\mathbf{Z},\mathbf{U}) \le \mathbf{0} \right\}$$

where: $\bar{J} = (1-\alpha)J + \alpha \left\langle J_\varepsilon, \Sigma J_\varepsilon \right\rangle \big|_{\varepsilon=\mathbf{0}}$, $\alpha \in [0,1]$;

Σ is a positive semi-definite weighting matrix.

Similarly, in terms of measure (b), the applicable deterministic problem is (Rustem: 1989, p.198):

MP19: $$\min_{\mathbf{U}} \left\{ \bar{J}(\mathbf{Z},\mathbf{U},\alpha) \,\middle|\, \mathrm{F}(\mathbf{Z},\mathbf{U},\mathbf{0}) = \mathbf{0};\ \mathrm{G}(\mathbf{Z},\mathbf{U}) \le \mathbf{0} \right\}$$

where: $\bar{J} = (1-\alpha)J + \alpha \left\| \mathbf{Z}_\varepsilon^{\mathrm{T}} \Sigma \mathbf{Z}_\varepsilon \right\|_{\mathrm{F}}^2$, $\alpha \in [0,1]$;

$\|\mathbf{A}\|_{\mathrm{F}} = \sqrt{\sum_i \sum_j |a_{ij}|^2}$ denotes the Frobenous norm of the matrix **A**.

The function $\bar{J}$ is thus a convex combination of the original objective function, *J*, and a sensitivity measure evaluated at the most likely value of the disturbance vector (here taken to be ε = 0 with no loss in generality). The weighting matrix Σ depends on the type of uncertainty considered, and its elements determine the relative importance attached to the various sources of this class of uncertainty in the system. To compute solutions that are robust with respect to perturbations due to the stochastic nature of the behavioural equations, Σ would be represented by the variance-covariance matrix of the error terms; for robustness against uncertainty in the forecasts of exogenous variables, Σ would be represented by the variance-covariance matrix of the forecast errors; and for robustness against uncertainty in estimated model parameters, Σ would be represented by the variance-covariance matrix of the estimates.

The sensitivity components of $\bar{J}$ can be shown to be related to $\mathbf{var}\{J(\mathbf{Z}^*, \mathbf{U}^*, \varepsilon)\}$. With reference to the Mean-Variance objective function

$$\bar{G} = (1-\beta)\,\mathbf{E}\left\{J(\mathbf{Z},\mathbf{U},\varepsilon)\right\} + \beta \mathbf{var}\left\{J(\mathbf{Z},\mathbf{U},\varepsilon)\right\},\ \beta \in [0,1],$$

Rustem (1989, pp.198-199) shows that if the disturbance vector is $\varepsilon \sim N(0,\Omega)$, then an exact equivalence between the robust policy objective $\bar{J}$ and the Mean-Variance objective $\bar{G}$ is easily established for *LQG* problems by simply defining the weighting matrix Σ as the covariance matrix of the disturbance the vector, ε;[84] Paraskevopolulos, et al. (1991) empirically show that the *unperturbed* objective function $J(\mathbf{Z},\mathbf{U},0)$ and the associated sensitivity function $\left\langle J_\varepsilon, \Sigma J_\varepsilon \right\rangle \big|_{\varepsilon=\mathbf{0}}$ are adequate proxy measures of the expected value and variance of the random variable $J(\mathbf{Z}^*, \mathbf{U}^*, \varepsilon)$ respectively.

[84] For a non-linear models however, this equivalence does not hold. This is due to the fact that the non-linearity introduces a bias term in $\mathbf{E}\{\mathrm{J}(\cdot)\}$ which renders the assumption that $\mathbf{E}\{\mathrm{J}(\cdot)\}$ is equal to the deterministic version of the original objective function *J* to be untrue; furthermore, equivalence of $\mathbf{var}\{\mathrm{J}(\cdot)\}$ to the sensitivity terms introduced can only be established if the coefficient of the first-order Taylor series approximation of the nonlinear model at $\varepsilon = \mathbf{0}$ is a constant. However, see Becker, et al. (1994) for a correction that may be used to alleviate this.

Copyright © 1997-2006: Ike's Research Ltd

The parameter α of the augmented objective function $\bar{J}$ may be therefore interpreted as a risk-aversion factor that signifies a trade-off between the size of the expected payoff and the uncertainty associated with the payoff. The act of choosing any particular value for α implies a cost-benefit trade-off of some sort: the "cost of α" is the *detriment* in the expected payoff, and the "benefit of α" is the *reduction* in the degree of uncertainty that is associated with the expected payoff. And in this risk-aversion interpretation, α signifies the degree of uncertainty in the solution which the decision maker can tolerate: a risk-taker can be expected to exhibit a value of α that is close to zero, whereas a more risk-averse decision-maker would be characterised by an α which is close to one.

As regards the determination of the particular value of α that applies to a given implementation, it has been suggested that this should be done implicitly. That is say, the analyst should generate a series of solutions corresponding to a finite set of robustness parameters, $\mathbf{A}: \alpha^k \in [0,1]$, $k = 1, 2, ..., m$, i.e.:

$$\left\{\alpha^k, \mathbf{E}\{J(\mathbf{Z}^*, \mathbf{U}^*, \varepsilon)\}, \mathbf{var}\{J(\mathbf{Z}^*, \mathbf{U}^*, \varepsilon)\}\right\}_{\alpha^k \in \mathbf{A}}$$

The set of solutions should be presented (in tabular or graphical form) to the user who then selects the combination that most appeals to him or her. This solution procedure is what is termed *a posteriori articulation of preferences* in the multi-objective optimisation lexicon; it is seemingly viable, but it can be problematic.[85] An alternative method would be to adopt an *a priori articulation of preferences* approach such as GENO's EC solution method illustrated by Examples 3.10 through 3.12. In the robust optimisation context, one solves the deterministic bi-objective problem:

MP20:

$$\underset{\bar{\mathbf{u}}}{\text{Opt}}\ J(\mathbf{Z},\mathbf{U}) = \{f_1(\mathbf{Z},\mathbf{U}),\ f_2(\mathbf{Z},\mathbf{U})\}$$

Subject to: $F(\mathbf{Z},\mathbf{U},\varepsilon) = \mathbf{0}$; $G(\mathbf{Z},\mathbf{U}) \le \mathbf{0}$; $\varepsilon = \mathbf{0}$

Where: $f_1(\mathbf{Z},\mathbf{U}) = J(\mathbf{Z},\mathbf{U},\varepsilon)$; $f_2(\mathbf{Z},\mathbf{U}) = \langle J_\varepsilon, \Sigma J_\varepsilon \rangle\big|_{\varepsilon=\mathbf{0}}$

The challenge now is provide a simple mechanism for determining (the model user's) 'ideal outcome' that GENO's EC method requires. One approach may be to adapt *knowledge elicitation techniques* that are common to developers of computer-based *Expert Systems*. One could proceed as follows: (1) first solve MP18 (or MP19) to establish the two extreme outcomes J_s and J_r corresponding to $\alpha \approx 0$ (the "almost sure outcome") and $\alpha \approx 1$ (the "risky outcome") respectively, together with the accompanying sensitivity values S_s and S_r; (2) next, obtain the model user's preference by posing the following question: "On a scale of 1 to 10, given that 1 yields (J_s, S_s) and 10 yields (J_r, S_r), what is your preference?"; (3) the stated preference—a number on $[0,10]$—can then be translated back to the α-scale as α_p; (4) MP18 (or MP19) should then be solved to obtain S_{α_p}; (5) the 'ideal point' in outcome space as required by MP20 is then $[J_r, S_{\alpha_p}]$. Alternatively, the problem should be re-formulated such that the ideal solution *is logically* the origin of the outcome space.

Closure

The idea of replacing a stochastic programming problem with an approximate deterministic equivalent that includes a compensating term to ensure robust solutions affords a method of dealing with the uncertainty that is ubiquitous in practical systems. GENO's EC solution procedure offers a computationally efficient method of generating solutions in a given setting.

---

[85] Solving multi-objective problems generally comprises two fairly distinct stages: a *search procedure* and a *decision-making* process. Depending on how the two are combined, three classes of methods are discernable, namely:

1. *A priori Articulation of Preferences*. [Decide-then-Search] The model-user expresses his preferences in terms of an aggregate utility function *prior to optimisation*.
2. *A posteriori Articulation of Preferences*. [Search-then-Decide] The model-user is presented with a set of non-inferior solutions; he proceeds to select a solution from the given set.
3. *Progressive Articulation of Preferences*. [Decide-and-Search] Preference articulation by the decision maker and solution generation proceed in parallel at inter-leafed steps.

Most current research on the numerical solution of multi-objective problems has tended to concentrate on devising techniques for generating the entire set of non-inferior solutions, in a 'search-then-decide' solution strategy. This approach has major drawbacks: if the number of objectives is large, then the computational effort required to generate the entire non-inferior set can be very substantial; worse still, presenting a visually simple solution set is impossible except in the two dimensional case. Hence the user may not be able to articulate his preferences after all.

Copyright © 1997-2006: Ike's Research Ltd

III. **The gep File**: Example 3.13

```
// A Pareto solution of a constrained multi-objective dynamic optimization problem
// Source: See Footnote 75

#definecs p_maxgens     200
#definecs p_popsize     20
#definecs p_agents      2
#definecs p_order       1
#definecs p_plan        6

#include dynamic_gep_defaults.src


let vars[p_agents, p_order]   = {1,     // variables controlled by agent 1
                                 1}     // variables controlled by agent 2

let i_point[p_agents]   = 51   25;

adj_mode        = "g";
solution_type   = "i";
pos_orth        = true;
proximity       = true;

p_s_xover   = 0.55;
p_a_xover   = 0.55;
p_b_xover   = 0.00;
p_h_xover   = 0.55;
p_d_xover   = 0.55;
p_shuffle   = 0.55;

d_factor    = 0.80;
quantum_0   = 0.1;
rand_seed   = 240658;

proc (1) = m_rate(i,d); retp(0.05); endp;

proc (1) = bm_rate(d); retp(0.005); endp;


//problem-specific data
//source: Schoenberg (1994).
let ave_ret[p_plan] = 15.852    14.262   31.336   25.775   50.228   14.842;
let std_dev[p_plan] = 37.215    41.773   33.165   62.009   60.720   23.757;

let corr_mtx[p_plan,p_plan] = {1.000   0.944   0.146   0.231   0.379   0.258,
                               0.944   1.000   0.109   0.239   0.413   0.223,
                               0.146   0.109   1.000  -0.169  -0.229   0.691,
                               0.231   0.239  -0.169   1.000   0.882  -0.256,
                               0.379   0.413  -0.229   0.882   1.000  -0.284,
                               0.258   0.223   0.691  -0.256  -0.284   1.000};

proc (2) = f(i, d, v_array);
local u,x,fv,fv1,fv2;
   u = matinit(order, plan, 0);
   x = matinit(order, horizon, 0);
  {u,x} = assign_sequences(i,d,u,x);[86]

   if(d == 1);       fv = portfolio_return(u);
   elseif(d == 2);   fv = portfolio_risk(u);
   endif;
   retp (fv, v_array);
endp;
```

[86] Infra page 5, footnote 10.

Copyright © 1997-2006: Ike's Research Ltd

```
proc (1) = portfolio_return(u);
local k,fv;
   fv = 0;
   for k (1,plan,1);
      fv = fv + ave_ret[k]*u[k];
   endfor;
   if (maximise); fv = fv; else; fv = - fv; endif;
   retp(fv);
endp;


proc (1) = portfolio_risk(u);
local k,t,fv,stop_at,sum1,sum2;
   fv = 0; sum2 = 0;
   sum1 = (u[1]^2)*(std_dev[1]^2);
   for t (2,plan,1);
      stop_at = t - 1;
      sum1 = sum1 + (u[t]^2)*(std_dev[t]^2);
      for k (1,stop_at,1);
         sum2 = sum2 + corr_mx[t,k]*u[t]*u[k]*std_dev[t]*std_dev[k];
      endfor;
   endfor;
   fv = sqrt(sum1 + 2*sum2);
   if (maximise); fv = fv; else; fv = - fv; endif;
   retp(fv);
endp;


proc (1) = a(i,j,k); retp(state_fn(i,1,k)[87]); endp;

proc (1) = b(i,j,k); retp(-1.0); endp;

proc (1) = aa(z,w,j,k); retp(z[k]); endp;

proc (1) = bb(z,w,j,k); retp(-1.0); endp;

proc (1) = ee(z,w,j,k); retp((z[k+1] + w[k])); endp;


proc (2) = dynamic_control_bounds(i,j,k,low,upp);
local n, fv, budget, stop_at;
   budget = 1;
   if(k == 1); fv = budget;
   else;
      fv = 0;
      stop_at = k - 1;
      for n (1, stop_at, 1);
         fv = fv + control_fn(i,j,n);[88]
      endfor;
      if(fv > budget); fv = 0.0;
      else; fv = budget - fv;
      endif;
   endif;
   upp = min(fv, upp);
   low = max( 0, low);
   retp (low,upp);
endp;
```

[87] Infra page 26, footnote 39.

[88] Infra page 26, footnote 39.

Copyright © 1997-2006: Ike's Research Ltd

VI. **The dat File**: Example 3.13

Table 3.13: Input Data for the Portfolio Selection Problem

VARIABLE NAME		X1
UCB		1
LCB		0
USB		1
LSB		0
Initial State Vector		0
Final State Vector		0
Discrete Values		0
Inter-connexion Matrix	X1	1

V. **Points to Note**: Example 3.13

1. i_point — On this type of problem, the components of the `i_point` should always be (a) the largest asset return in the portfolio; (b) the maximum risk (as measured by the standard deviation) that the user is willing to bear.

2. proximity — The `proximity` parameter effectively switches GENO to a "next-to-ideal" mode of operation in the quest for a multi-objective solution to the problem.

3. `ee(z,w,j,k)` — The state vector is fixed at the final "time", hence the need for the **`proc`** `ee(z,w,j,k)`. Note that the final value needn't be '0'; it can be any other value between '0' and '1'. The '0' ensures that *all* the available funds are invested in the chosen assets.

4. `risk-free asset` — If the portfolio includes a risk-free asset, one can easily add this to the computation by merely including its rate return and a standard deviation of '0'.

5. `allocation bounds` — If the investment in a particular asset is required to be within certain bounds, this may easily be accommodated via **`proc`** `dynamic_state_bounds`. For instance, in the example above, if the funds allocated to Microsoft — the 4th asset in the sequence — are required to be at least 15% but less than 20% the total, one need to add code that dynamically modulates of the bounds affecting the 5th column of the state matrix as shown below:

```
proc (2) = dynamic_state_bounds(i,j,k,low,upp);
local fv1,fv2;
   fv1 = low;
   fv2 = upp;
   if(k == 4);
      fv1 = state_fn(i,j,k)[89] - 0.20;
      fv2 = state_fn(i,j,k) - 0.15;
   endif;
upp = min(fv2, upp);
low = max(fv1, low);
retp (low,upp);
endp;
```

The code above follows directly from the dynamic constraint $x_{k+1} = x_k - u_k$ because if u_4 is restricted to the set [0.15, 0.20], then by elementary interval arithmetic, the variable x_5 clearly must lie within the interval [(x_4 - 0.20), (x_4 - 0.15)]. However, this technique is most effective when the restricted asset is located early in the sequence of assets, i.e. $k = 1$ or 2. A larger population size may also be necessary.

[89] Infra page 26, footnote 39.

Copyright © 1997-2006: Ike's Research Ltd

Example 3.14 Multi-objective Optimisation: A Dynamic Non-cooperative Game [90]

Problem 1: $\max_{\mathbf{u}} J_1(\mathbf{x},\mathbf{u}) = x_T^1 - (2T)^{-1}\sum_{k=0}^{T-1} u_k^2 - Q(\mathbf{x})$

Subject to: $x_{k+1}^1 = x_k^2$

$x_{k+1}^2 = 2x_k^2 - x_k^1 + (T)^{-2} u_k^2$

$\mathbf{x}_k \in \mathbf{R}$; $u_k \in [-100, 100]$; $T = 5$, $\mathbf{x}_0 = (0,0)$.

Problem 2: $\min_{\mathbf{u}} J_2(\mathbf{x},\mathbf{u}) = |x_T^3| + |x_T^4| + \sum_{k=0}^{T-1} |x_k^3| + |x_k^4| + 0.5|u_k^3| + 0.5|u_k^4| + Q(\mathbf{x})$

Subject to: $x_{k+1}^3 = x_k^3 + x_k^4 + u_k^3$

$x_{k+1}^4 = x_k^4 - x_k^3 + u_k^4$

$x_k^i \in [-2, 2]$, $i = 3, 4$; $u_k^i \in [-1, 1]$, $i = 3, 4$; $T = 5$, $\mathbf{x}_0 = (1,2)$.

Problem 3: $\max_{\mathbf{u}} J_3(\mathbf{x},\mathbf{u}) = s(q)^N (x_N^5)^{1-v} + \sum_{k=0}^{T-1} [(q)^k (u_k^5)^{1-v}] - Q(\mathbf{x})$

Subject to: $x_{k+1}^5 = a(x_k^5 - u_k^5)$

$(u_k^5 - x_k^5) \le 0$

$x_k^5 \in \mathbf{R}_+$; $u_k^5 \in \mathbf{R}_+$; $T = 5$, $x_0^5 = 1.0$, $a = 1.02$, $q = 0.95$, $s = v = 0.5$.

Problem 4: $\min_{\mathbf{u}} J_4(\mathbf{x},\mathbf{u}) = q(x_N^6)^2 + \sum_{k=0}^{T-1} [s(x_k^6)^2 + r(u_k^6)^2] + Q(\mathbf{x})$

Subject to: $x_{k+1}^6 = ax_k^6 + bu_k^6$

$x_k^6 \in \mathbf{R}$; $u_k^6 \in \mathbf{R}$; $T = 5$, $x_0^6 = 100$, $a = 0.01$, $q = 0.95$, $b = s = q = r = 1$.

Problem 5: $\max_{\mathbf{u}} J_5(\mathbf{x},\mathbf{u}) = \prod_{k=1}^{T} [1 + ku_{k-1}^7] - Q(\mathbf{x})$

Subject to: $x_{k+1}^7 = x_k^7 - u_k^7$

$\sum_{k=1}^{T} u_k^7 = 8$

$x_k^7 \in \mathbf{R}$; $u_k^7 \in [0, 5]$; $T = 5$, $x_0^7 = 8$.

Problem 6: $\min_{\mathbf{u}} J_6(\mathbf{x},\mathbf{u}) = |x_T^8 - 1| + \sum_{k=0}^{T-1} [2 + u_k^8](3)^{-x_k^8} + Q(\mathbf{x})$

Subject to: $x_{k+1}^8 = x_k^8 + [x_k^8 - 2x_k^8 + 1.25(x_k^8)^2 - 0.25(x_k^8)^3]u_k^8$

$x_k^8 \in [0, 3]$; $u_k^8 \in [-1, 1]$; $T = 5$, $x_0^8 = 2$.

Problem 7: $\max_{\mathbf{u}} J_7(\mathbf{x},\mathbf{u}) = \sum_{k=0}^{T-1} \sqrt{u_k^9} - Q(\mathbf{x})$

Subject to: $x_{k+1}^9 = ax_k^9 - u_k^9$

$x_0^9 = x_T^9$

$x_k^9 \in \mathbf{R}$; $u_k^9 \in \mathbf{R}_+$; $T = 5$, $a = 1.1$, $x_0^9 = x_T^9 = 100$;

The Coupler: $Q(\mathbf{x}) = \sum_{j=1}^{9} \left(\sum_{k=0}^{T-1} (u_k^{*j} - u_k^j)^2 + \sum_{k=0}^{T} (x_k^{*j} - x_k^j)^2 \right)$

[90] **Source**: Problems 1, 3, 4 and 7 are from Michalewicz (1994); Problems 2, 5 and 6 are from Larson and Casti (1978); and the dynamic game formed by introduction of the coupler Q(**x**) is original to Siwale (2002).

Copyright © 1997-2006: Ike's Research Ltd

I. **GENO** Output Case I: Simultaneous Solution of De-coupled System

Generation	Object [1]	Object [2]	Object [3]	Object [4]	Object [5]	Object [6]	Object [7]
0	-5.351000	-14.500000	1.874277	-20928.379923	2024.055000	-0.883203	15.640603
40	0.119994	-11.500000	2.105092	-10000.500013	4413.550436	-0.647443	15.955390
80	0.120000	-11.500000	2.105094	-10000.500012	4415.711874	-0.632810	15.955390
100	0.120000	-11.500000	2.105094	-10000.500012	4415.712306	-0.632385	15.955390
120	0.120000	-11.500000	2.105094	-10000.500012	4415.712306	-0.630116	15.955390
140	0.120000	-11.500000	2.105094	-10000.500012	4415.712306	-0.630041	15.955390
160	0.120000	-11.500000	2.105094	-10000.500012	4415.712306	-0.629934	15.955390
180	0.120000	-11.500000	2.105094	-10000.500012	4415.712517	-0.629107	15.955390
200	0.120000	-11.500000	2.105094	-10000.500012	4415.712518	-0.629050	15.955390
220	0.120000	-11.500000	2.105094	-10000.500012	4415.712520	-0.629036	15.955390
240	0.120000	-11.500000	2.105094	-10000.500012	4415.712520	-0.629028	15.955390
260	0.120000	-11.500000	2.105094	-10000.500012	4415.712521	-0.629025	15.955390
280	0.120000	-11.500000	2.105094	-10000.500012	4415.712521	-0.629016	15.955390
300	0.120000	-11.500000	2.105094	-10000.500012	4415.712521	-0.629013	15.955390

Solution to Problem 1: Optimum Objective Function Value = 0.120000

Control 1:	0.000000	0.000000	0.000000	0.000000	0.000000	
Control 2:	0.800000	0.600000	0.400000	0.200000	0.000000	
State 1:	0.000000	0.000000	0.032000	0.088000	0.160000	0.240000
State 2:	0.000000	0.032000	0.088000	0.160000	0.240000	0.320000

Solution to Problem 2: Optimum Objective Function Value = -11.500000

Control 3:	-1.000000	-1.000000	0.000000	1.000000	0.000000	
Control 4:	-1.000000	1.000000	1.000000	1.000000	0.000000	
State 3:	1.000000	2.000000	1.000000	0.000000	0.000000	0.000000
State 4:	2.000000	0.000000	-1.000000	-1.000000	0.000000	0.000000

Solution to Problem 3: Optimum Objective Function Value = 2.105094

Control 5:	0.225661	0.211887	0.198954	0.186810	0.175407	
State 5:	1.000000	0.789825	0.589497	0.398354	0.215775	0.041175

Solution to Problem 4: Optimum Objective Function Value = -10000.500012

Control 6:	-0.500013	-0.002500	-0.000012	0.000000	0.000000	
State 6:	100.000000	0.499987	0.002499	0.000013	0.000000	0.000000

Solution to Problem 5: Optimum Objective Function Value = 4415.712521

Control 7:	1.056631	1.556632	1.723301	1.806636	1.856800	
State 7:	8.000000	6.943369	5.386737	3.663436	1.856800	0.000000

Solution to Problem 6: Optimum Objective Function Value = -0.629013

Control 8:	0.963664	0.022570	0.001489	-0.571663	-1.005097	
State 8:	2.000000	2.963664	2.997695	2.999929	2.142424	1.000000

Solution to Problem 7: Optimum Objective Function Value = 15.955390

Control 9:	6.830134	8.264463	10.000000	12.100000	14.641000	
State 9:	100.000000	103.169866	105.222389	105.744628	104.219091	100.000000

Copyright © 1997-2006: Ike's Research Ltd

I. **GENO Output** Case II: Nash Equilibrium Solution of Coupled System

Generation	Object [1]	Object [2]	Object [3]	Object [4]	Object [5]	Object [6]	Object [7]
0	-11177.654097	-11186.803097	-11170.428820	-32100.683020	-9148.248097	-11173.416805	-11156.662494
40	0.079364	-11.540636	2.064451	-10000.540649	4406.813592	-0.679876	15.914752
80	0.119376	-11.500624	2.104470	-10000.500636	4415.646404	-0.633010	15.954766
100	0.119756	-11.500244	2.104849	-10000.500257	4415.698417	-0.632335	15.955145
120	0.119774	-11.500226	2.104868	-10000.500238	4415.701403	-0.632316	15.955164
140	0.119788	-11.500212	2.104881	-10000.500225	4415.701590	-0.632152	15.955178
160	0.119804	-11.500196	2.104898	-10000.500208	4415.701831	-0.632116	15.955194
180	0.119799	-11.500201	2.104893	-10000.500213	4415.703547	-0.632038	15.955189
200	0.119780	-11.500220	2.104874	-10000.500232	4415.703528	-0.631963	15.955170
220	0.119776	-11.500224	2.104869	-10000.500237	4415.703527	-0.631939	15.955165
240	0.119778	-11.500222	2.104871	-10000.500235	4415.703528	-0.631937	15.955167
260	0.119777	-11.500223	2.104871	-10000.500235	4415.703530	-0.631934	15.955167
280	0.119963	-11.500037	2.105057	-10000.500049	4415.703716	-0.631283	15.955353
300	0.119960	-11.500040	2.105054	-10000.500052	4415.703713	-0.631024	15.955350

Solution to Problem 1: Optimum Objective Function Value = 0.119960

Control 1:	0.000000	0.000000	0.000000	0.000000	0.000000	
Control 2:	0.800000	0.600000	0.400000	0.200000	0.000000	
State 1:	0.000000	0.000000	0.032000	0.088000	0.160000	0.240000
State 2:	0.000000	0.032000	0.088000	0.160000	0.240000	0.320000

Solution to Problem 2: Optimum Objective Function Value = -11.500040

Control 3:	-1.000000	-1.000000	0.000000	1.000000	0.000000	
Control 4:	-1.000000	1.000000	1.000000	1.000000	0.000000	
State 3:	1.000000	2.000000	1.000000	0.000000	0.000000	0.000000
State 4:	2.000000	0.000000	-1.000000	-1.000000	0.000000	0.000000

Solution to Problem 3: Optimum Objective Function Value = 2.105054

Control 5:	0.225661	0.211887	0.198954	0.186810	0.175407	
State 5:	1.000000	0.789825	0.589497	0.398354	0.215775	0.041175

Solution to Problem 4: Optimum Objective Function Value = -10000.500052

Control 6:	-0.500013	-0.002501	-0.000013	0.000000	0.000000	
State 6:	100.000000	0.499987	0.002499	0.000012	0.000000	0.000000

Solution to Problem 5: Optimum Objective Function Value = 4415.703713

Control 7:	1.057573	1.557582	1.724254	1.807591	1.853000	
State 7:	8.000000	6.942427	5.384845	3.660591	1.853000	0.000000

Solution to Problem 6: Optimum Objective Function Value = -0.631024

Control 8:	0.965796	0.022102	0.000538	-0.587641	-1.003520	
State 8:	2.000000	2.965796	2.999112	2.999919	2.118446	1.000000

Solution to Problem 7: Optimum Objective Function Value = 15.955350

Control 9:	6.830135	8.264463	10.000000	12.100000	14.640999	
State 9:	100.000000	103.169865	105.222389	105.744628	104.219090	100.000000

Copyright © 1997-2006: Ike's Research Ltd

II. General Remarks

This example was formulated for two reasons: (a) to demonstrate an additional capability of GENO, namely that of simultaneously solving a set of dynamic optimisation problems; (b) to test GENO's ability at computing the Nash equilibrium solution of a dynamic non-cooperative game. To this end, a 'coupler function' Q(**x**) was added to a collection of optimal control problems: when "switched on", the coupler essentially turns the disparate group of optimisation problems into a dynamic game expressed in extensive form (Basar and Olsder: 1999, Chap.5).

It has long been known that a discrete-time optimal control problem is essentially a static mathematical program. This immediately becomes apparent when one adopts the 'variable stacking' technique wherein each sampling instant is interpreted as simply a label for an independent variable.[91] The individual problems comprising Example 13 can therefore be regarded as static mathematical programs, and GENO's performance on the static optimisation problem has been amply demonstrated by Examples 3.1 through 3.6. By the same token, the coupled system following the introduction of the coupler function can be converted into a static game, and Examples 3.9 through 3.10 confirm GENO's efficiency on this type of problem. However, because GENO does not explicitly use 'variable stacking',[92] and although the algorithm has been shown to perform well on the single-objective dynamic problem (via Examples 3.7 and 3.8), and separately on the static games, a direct evaluation using a truly dynamic game would still be informative.

In order to validate the Nash solution, the coupler term was designed such that the Nash equilibrium solution coincides with that of the de-coupled system. That is to say, if **x*** denotes the Nash equilibrium solution and $J_i^c(\mathbf{x}), J_i^d(\mathbf{x})$ the ith payoff when the system is coupled and de-coupled respectively, then by design:[93]

$$\mathbf{x}^* : \; Q(\mathbf{x}^*) = 0 \quad \Leftrightarrow \quad J_i^d(\mathbf{x}^*) = J_i^c(\mathbf{x}^*), \;\; \forall i \,. \tag{13a}$$

By this means, one can therefore ascertain whether GENO manages to find the Nash equilibrium or not.

Apart from serving as a reference for the Nash computation, the de-coupled system also provides further tests of GENO on dynamic optimisation problems. In all cases, the quality of the GENO solution is at least as good as that computed by others: and in cases where the solutions are the same, GENO is always more efficient compared to other numerical techniques of the same type, i.e. it uses a smaller population and fewer generations.[94] Furthermore, GENO solves all the problems simultaneously, using a single population.

It is not hard to imagine that the induced Nash equilibrium problem has a complex landscape in outcome space. Fonseca and Flemming (1995) graphically show that, even for a simple bi-objective problem, the introduction of constraints can result in landscapes that are very difficult to search. They write:

> "Despite the underlying objectives being continuous, smooth and uni-modal, the landscapes can be seen to exhibit features such as discontinuities, non-smoothness and flat regions. Optimisers capable of coping with such features are [therefore] necessary." (p.18)

For a problem with a large number of objectives, the situation can only be a lot worse. Yet, even in this case where there are seven objectives to be optimised GENO manages to compute a close approximation to the Nash solution. The results labelled 'Nash solution of coupled system' show the computed Nash solution to be accurate up to the third decimal place on a majority of the criteria at generation 300.

---

[91] See e.g. Whittle (1982).

[92] Recall that the design of the algorithm assumes a dynamic formulation, with the static problem being treated as the degenerate case.

[93] The implicit assumption underlying 13a is that **x*** is the global solution (or nearly so) for each sub-problem. Clearly this must be the case for otherwise there would exist opportunities to unilaterally improve one payoff by adopting a different strategy from that which obtains at the equilibrium point contrary to the definition of an equilibrium point as defined by John Nash (Nash, 1951).

[94] See Michalewicz, *et al*. (1992); Michalewicz (1994, Chapter 6), and Larson and Casti (1978, Chapter 3).

Copyright © 1997-2006: Ike's Research Ltd

III. **The gep File**: Example 3.14

```
// Nash solution of a constrained multi-objective dynamic optimization problem
// Source: Larson and Casti(1978); Michalewicz(1994); Siwale(2002)

#definecs p_maxgens 300
#definecs p_popsize 30
#definecs p_agents  7
#definecs p_order   9
#definecs p_plan    5

#definecs p_coupler  true  // Options: true   for the coupled system
                           //          false  for the de-coupled system

#include dynamic_gep_defaults.src

let vars[p_agents, p_order]   = {
               1   1   0   0   0   0   0   0   0,    //variables controlled by agent 1
               0   0   1   1   0   0   0   0   0,    //variables controlled by agent 2
               0   0   0   0   1   0   0   0   0,    //variables controlled by agent 3
               0   0   0   0   0   1   0   0   0,    //variables controlled by agent 4
               0   0   0   0   0   0   1   0   0,    //variables controlled by agent 5
               0   0   0   0   0   0   0   1   0,    //variables controlled by agent 6
               0   0   0   0   0   0   0   0   1     //variables controlled by agent 7
               };

let ff_states[p_order]  = 0   0   0   0   0   0   0   0   1;
let cc_states[p_order]  = 0   1   1   1   1   1   1   1   1;

adj_mode       = "s";
solution_type  = "e";

p_s_xover      = 0.55;
p_a_xover      = 0.55;
p_b_xover      = 0.00;
p_h_xover      = 0.55;
p_d_xover      = 0.55;
p_shuffle      = 0.55;

d_factor       = 0.80;
quantum_0      = 0.1;
rand_seed      = 240658;

proc (1) = m_rate(i,d); retp(0.05); endp;

proc (1) = bm_rate(d); retp(0.005); endp;


// Problem-specific data: The optimal control matrix as computed by the de-coupled system
let nash_c_mtx[p_order, 5] = {
    0.000000    0.000000    0.000000    0.000000    0.000000,
    0.800000    0.600000    0.400000    0.200000    0.000000,
   -1.000000   -1.000000    0.000000    1.000000    0.000000,
   -1.000000    1.000000    1.000000    1.000000    0.000000,
    0.225661    0.211887    0.198954    0.186810    0.175407,
   -0.500013   -0.002501   -0.000013    0.000000    0.000000,
    1.057077    1.557078    1.723745    1.807100    1.855000,
    0.965827    0.021971   -0.000323   -0.584176   -1.003733,
    6.830135    8.264463   10.000000   12.100000   14.640999
    };

// Problem-specific data: The optimal state matrix as computed by the de-coupled system
let nash_x_mtx[p_order, 6] = {
       0.000000      0.000000      0.032000      0.088000      0.160000      0.240000,
       0.000000      0.032000      0.088000      0.160000      0.240000      0.320000,
       1.000000      2.000000      1.000000      0.000000      0.000000      0.000000,
       2.000000      0.000000     -1.000000     -1.000000      0.000000      0.000000,
       1.000000      0.789825      0.589497      0.398354      0.215775      0.041175,
     100.000000      0.499987      0.002499      0.000012      0.000000      0.000000,
       8.000000      6.942923      5.385845      3.662100      1.855000      0.000000,
       2.000000      2.965827      2.998945      2.998461      2.121974      1.000000,
     100.000000    103.169866    105.222389    105.744628    104.219090    100.000000
    };
```


Copyright © 1997-2006: Ike's Research Ltd

```
proc (2) = f(i, d, v_array);
local q1,q2,s1,s2,v1,r2;
local x,fv,fv1,j,k,m,u;
   u = matinit(order, plan, 0);
   x = matinit(order, horizon, 0);
   { q1, s1, v1 } = (0.95, 0.5, 0.50);
   { q2, s2, r2 } = (1.00, 1.0, 1.00);
   { u, x } = assign_sequences(i,d,u,x);[95]

   //list of objectives
   fv = 0.0;
   if(d == 1);
      for k (1, plan, 1);
         fv = fv + u[2,k]*u[2,k];
      endfor;
      fv = x[1,horizon] - (fv/(2*horizon));
   elseif(d == 2);
      for k (1, plan, 1);
         fv = fv + abs(x[3,k]) + abs(x[4,k]) + 0.5*abs(u[3,k]) + 0.5*abs(u[4,k]);
      endfor;
      fv = fv + abs(x[3,horizon]) + abs(x[4,horizon]);
      fv = -fv;
   elseif(d == 3);
      for k (1, plan, 1);
         fv = fv + (q1^(k-1))*(u[5,k]^(1-v1));
      endfor;
      fv = fv + s1*(x[5,horizon]^(1-v1))*(q1^plan);
   elseif(d == 4);
      for k (1, plan, 1);
         fv = fv + s2*x[6,k]*x[6,k] + r2*u[6,k]*u[6,k];
      endfor;
      fv = fv + q2*x[6,horizon]*x[6,horizon];
      fv = -fv;
   elseif(d == 5);
      fv =  1.0;
      for k (1, plan, 1);
         fv = fv*(1 + k*u[7,k]);
      endfor;
   elseif(d == 6);
      for k (1, plan, 1);
         fv = fv + (2 + u[8,k])*(3.0^(-x[8,k]));
      endfor;
      fv = fv + abs(x[8,horizon] - 1);
      fv = -fv;
   elseif(d == 7);
      for k (1, plan, 1);
         fv = fv + (u[9,k]^0.5);
      endfor;
   endif;
   if (maximise); fv = fv - coupler(d,x); else; fv = - fv + coupler(d,x); endif;
   retp (fv, v_array);
endp;


proc (1) = a(i,j,k);
local fv;
   if(j == 1);        fv = state_fn(i,2,k);[96]
   elseif(j == 2);    fv = 2*state_fn(i,2,k) - state_fn(i,1,k);
   elseif(j == 3);    fv = state_fn(i,3,k) + state_fn(i,4,k);
   elseif(j == 4);    fv = state_fn(i,4,k) - state_fn(i,3,k);
   elseif(j == 5);    fv = 1.02*state_fn(i,j,k);
   elseif(j == 6);    fv = 0.01*state_fn(i,j,k);
   elseif(j == 7);    fv = state_fn(i,j,k);
   elseif(j == 8);    fv = state_fn(i,j,k);
   elseif(j == 9);    fv = 1.1*state_fn(i,j,k);
   endif;
   retp(fv);
endp;
```

[95] Infra page 5, footnote 10.

[96] Infra page 26, footnote 39.

Copyright © 1997-2006: Ike's Research Ltd

```
proc (1) = b(i,j,k);
local fv;
   if(j == 1);        fv =  0.00;
   elseif(j == 2);    fv =  1.00/(horizon*horizon);
   elseif(j == 3);    fv =  1.00;
   elseif(j == 4);    fv =  1.00;
   elseif(j == 5);    fv = -1.02;
   elseif(j == 6);    fv =  1.00;
   elseif(j == 7);    fv = -1.00;
   elseif(j == 8);    fv =  state_fn(i,j,k) - 2*state_fn(i,j,k) + \
                            1.25*(state_fn(i,j,k)^2.0) - 0.25*(state_fn(i,j,k)^3.0);
   elseif(j == 9);    fv = -1.00;
   endif;
   retp(fv);
endp;

proc (1) = aa(z,w,j,k);
local fv;
   if(j == 1);        fv = z[2,k];
   elseif(j == 2);    fv = 2*z[2,k] - z[1,k];
   elseif(j == 3);    fv = z[3,k] + z[4,k];
   elseif(j == 4);    fv = z[4,k] - z[3,k];
   elseif(j == 5);    fv = 1.02*z[j,k];
   elseif(j == 6);    fv = 0.01*z[j,k];
   elseif(j == 7);    fv = z[j,k];
   elseif(j == 8);    fv = z[j,k];
   elseif(j == 9);    fv = 1.1*z[j,k];
   endif;
   retp(fv);
endp;

proc (1) = bb(z,w,j,k);
local fv;
   if(j == 1);        fv =  0.00;
   elseif(j == 2);    fv =  1.00/(horizon*horizon);
   elseif(j == 3);    fv =  1.00;
   elseif(j == 4);    fv =  1.00;
   elseif(j == 5);    fv = -1.02;
   elseif(j == 6);    fv =  1.00;
   elseif(j == 7);    fv = -1.00;
   elseif(j == 8);    fv =  z[8,k] - 2*z[8,k] + 1.25*(z[8,k]^2.0) - 0.25*(z[8,k]^3.0);
   elseif(j == 9);    fv = -1.00;
   endif;
   retp(fv);
endp;

proc (1) = ee(z,w,j,k);
local fv;
   if(j == 9); fv = (z[9,k+1] + w[9,k])/1.1;
   else;       fv = 0.0;
   endif;
   retp(fv);
endp;

proc (2) = dynamic_control_bounds(i,j,k,low,upp);
local fv, budget;
   budget = 8;
   if(j == 5); fv = state_fn(i,j,k);
   elseif(j == 7);
      if(k == 1); fv = budget;
      else;
         fv = 0;
         for n (1, (k - 1), 1);
            fv = fv + control_fn(i,j,n);
         endfor;
         fv = budget - fv;
         if(fv < 0); fv = 0.0; endif;
      endif;
   else; fv = large;
   endif;
   upp = min( fv, upp);
   low = max(-large, low);
   retp (low, upp);
endp;
```


Copyright © 1997-2006: Ike's Research Ltd

```
proc (1) = coupler(d,u,x);
local j,k,fv1,fv2;
   fv1 = 0; fv2 = 0;
   if(p_coupler);
      for j (1, order, 1);
         for k (1, plan, 1);
            fv1 = fv1 + (nash_c_mtx[j,k] - u[j,k])^2;
            fv2 = fv2 + (nash_x_mtx[j,k] - x[j,k])^2;
         endfor;
         fv2 = fv2 + (nash_x_mtx[j,horizon] - x[j,horizon])^2;
      endfor;
   endif;
   retp (fv1 + fv2);
endp;
```

VI. **The dat File**: Example 3.14

Table 3.14: Input Data for the Dynamic Non-cooperative Game

VARIABLE NAME		X1	X2	X3	X4	X5	X6	X7	X8	X9
UCB		10	10	1	1	1e20	100	5	1	1e20
LCB		-10	-10	-1	-1	0	-100	0	-1	0
USB		1e20	1e20	2	2	1e20	1e20	8	3	1e20
LSB		-1e20	0	-2	-2	0	-1e20	0	0	-1e20
Initial State Vector		0	0	1	2	1	100	8	2	100
Final State Vector		0	0	0	0	0	0	0	0	100
Discrete Values		0	0	1	1	0	0	0	0	0
Inter-connexion Matrix	X1	0	1	0	0	0	0	0	0	0
	X2	1	1	0	0	0	0	0	0	0
	X3	0	0	1	1	0	0	0	0	0
	X4	0	0	1	1	0	0	0	0	0
	X5	0	0	0	0	1	0	0	0	0
	X6	0	0	0	0	0	1	0	0	0
	X7	0	0	0	0	0	0	1	0	0
	X8	0	0	0	0	0	0	0	1	0
	X9	0	0	0	0	0	0	0	0	1

V. **Points to Note**: Example 3.14

1. p_coupler — A problem-specific Boolean matrix that couples or de-couples the system

Copyright © 1997-2006: Ike's Research Ltd

Example 3.15: Mixed-integer Non-linear Programming: Process Synthesis I [97]

$$\min_{\mathbf{x},\mathbf{y}} J(\mathbf{x},\mathbf{y}) = (y_1-1)^2 + (y_2-1)^2 + (y_3-1)^2 - \ln(y_4+1) + (x_1-1)^2 + (x_2-2)^2 + (x_3-3)^2$$

Subject to:

$$y_1 + y_2 + y_3 + x_1 + x_2 + x_3 - 5 \le 0 \qquad y_3^2 + x_1^2 + x_2^2 + x_3^2 - 5.5 \le 0$$

$$x_1 + y_1 - 1.2 \le 0 \qquad x_2 + y_2 - 1.8 \le 0$$

$$x_3 + y_3 - 2.5 \le 0 \qquad x_1 + y_4 - 1.2 \le 0$$

$$x_2^2 + y_2^2 - 1.64 \le 0 \qquad x_3^2 + y_3^2 - 4.25 \le 0$$

$$x_3^2 + y_2^2 - 4.64 \le 0 \qquad \mathbf{x} \in [0,\infty)$$

$$\mathbf{y} \in \{0,1\}^4$$

I. GENO Output

Generation	*Objective*
0	10.360000
50	4.811778
100	4.636855
150	4.636853
200	3.714389
250	3.714353
300	3.714353
350	3.714353
400	3.714353
450	3.714353
460	3.714353
470	3.557803
480	3.557471
490	3.557466
500	3.557466

Optimal State Vector: $\mathbf{x}$ = (0.200000, 1.280557, 1.954526, 1.000000, 0.000000, 0.000000, 1.000000) ^{T}

Objective Function Value: $J(\mathbf{x}, \mathbf{y})$ = 3.557466

II. General Remarks

This problem was originally proposed by Floudas, et al. (1989) and was subsequently tackled by others using various techniques; the latest effort appears to be that by Angira and Babu (2002) who used a differential evolution algorithm.

The best known solution as reported by Angira and Babu (2002) (and presumably all previous authors cited therein) is as follows:

Best Objective Function Value: $J(\mathbf{x}, \mathbf{y})$ = 3.557466

Optimal Control Sequence: $\mathbf{x}$ = (0.200000, 1.280557, 1.954526, 1.000000, 0.000000, 0.000000, 1.000000) ^{T}

As can be seen above, the solution by GENO is better than the best currently known solution, albeit marginally.

[97] **Source**: Floudas, Aggarwal and Ciric (1989).

Copyright © 1997-2006: Ike's Research Ltd

III. **The gep File**: Example 3.15

```
// Single-objective constrained static optimization problem
// Source: Floudas, Aggarwal and Ciric (1989).

#definecs p_maxgens  500
#definecs p_popsize  20
#definecs p_agents   1
#definecs p_order    7
#definecs p_plan     1

#include static_gep_defaults.src

let vars[p_agents, p_order]   = 1   1   1   1   1   1   1;
let discrete_var[p_order]     = 0   0   0   1   1   1   1;

adj_mode          = "s";
solution_type     = "e";
maximise          = false;

timer             = true;
sol_mtx_check     = true;
constraints_check = true;

//cross-over probabilities
p_s_xover         = 0.55;
p_a_xover         = 0.55;
p_b_xover         = 0.00;
p_h_xover         = 0.55;
p_d_xover         = 0.55;
p_shuffle         = 0.00;

d_factor          = 0.80;
quantum_0         = 0.1;
rand_seed         = 2406525;


proc (1) = m_rate(i,d); retp(0.05); endp;

proc (1) = bm_rate(d); retp(0.005); endp;


proc (2) = f(i, d, v_array);
local c_vector,fv,u,x,z;
   u = matinit(order, plan, 0);
   x = matinit(order, horizon, 0);
  {u,x} = assign_sequences(i,d,u,x);

   //evaluate constraints
   c_vector = constraints(0,x,horizon);
   v_array  = evaluate_constraints(c_vector,v_array);

   //evaluate objective
   fv = objective(0,x,horizon);
   retp (fv,v_array);
endp;


proc (1) = objective(z,x,k);
local fv;
   fv = (x[1,k] - 1)^2 + (x[2,k] - 2)^2 + (x[3,k] - 3)^2 + (x[4,k] - 1)^2 + (x[5,k] - 1)^2 +
(x[6,k] - 1)^2 - ln(x[7,k] + 1);
   if (maximise); fv = fv; else; fv = -fv; endif;
   retp(fv);
endp;
```


Copyright © 1997-2006: Ike's Research Ltd

```
proc (1) = constraints(z,x,k);
local c; c = zeros(9,1);
   c[1] = x[1,k] + x[2,k] + x[3,k] + x[4,k] + x[5,k] + x[6,k] - 5;
   c[2] = x[1,k]^2 + x[2,k]^2 + x[3,k]^2 + x[6,k]^2 - 5.5;
   c[3] = x[1,k] + x[4,k] - 1.2;
   c[4] = x[2,k] + x[5,k] - 1.8;
   c[5] = x[3,k] + x[6,k] - 2.5;
   c[6] = x[1,k] + x[7,k] - 1.2;
   c[7] = x[2,k]^2 + x[5,k]^2 - 1.64;
   c[8] = x[3,k]^2 + x[6,k]^2 - 4.25;
   c[9] = x[3,k]^2 + x[5,k]^2 - 4.64;
   retp (c);
endp;
```

VI. **The dat File**: Example 3.15

Table 3.15: Input Data for Process Synthesis I

VARIABLE NAME		X1	X2	X3	X4	X5	X6	X7
UCB		10	10	10	1	1	1	1
LCB		0	0	0	0	0	0	0
USB		10	10	10	1	1	1	1
LSB		0	0	0	0	0	0	0
Initial State Vector		0	0	0	0	0	0	0
Final State Vector		0	0	0	0	0	0	0
Discrete Values		0	0	0	1	1	1	1
Inter-connexion Matrix	X1	1	0	0	0	0	0	0
	X2	0	1	0	0	0	0	0
	X3	0	0	1	0	0	0	0
	X4	0	0	0	1	0	0	0
	X5	0	0	0	0	1	0	0
	X6	0	0	0	0	0	1	0
	X7	0	0	0	0	0	0	1

V. **Points to Note**: Example 3.15

1. `Method` — This is a standard implementation of version MPa of the original mathematical program. The 0-1 integer variables are easily accommodated by declaring the matrix `discrete_var` and making the appropriate entries in the dat file as illustrated above. But note that on other type of programs involving discrete variables (e.g. Example 3.4 in this Manual), the type of entries made in the dat file and the indicator matrix `discrete_var` should not be confused: the former is always a 0-1 indicator matrix with a '1' entry for each discrete variable, and a '0' entry otherwise; the later should contain the actual discrete values for each discrete variable, and an entry of '0' for all non-discrete variables. Note that in this particular case, the discrete value just happens to be 1.

Copyright © 1997-2006: Ike's Research Ltd

Example 3.16: Mixed-integer Non-linear Programming: Process Synthesis II [98]

$$\min_{\mathbf{x}} J(\mathbf{x}) = 7.5x_3 + 5.5(1-x_3) + 7x_1 + 6x_2 + 50\frac{(1-x_3)}{0.8[1-\exp(-0.4x_2)]} + 50\frac{x_3}{0.9[1-\exp(-0.5x_1)]}$$

Subject to: $0.9[1-\exp(-0.5x_1)] - 2x_3 \le 0$

$$0.8[1-\exp(-0.4x_2)] - 2(1-x_3) \le 0$$

$$x_1 \le 10x_3$$

$$x_2 \le 10(1-x_3)$$

$$x_1 \in [0,\infty)\,;\quad x_2 \in [0,\infty)\,;\quad x_3 \in \{0,1\}$$

I. GENO Output

Generation	*Objective*
0	107.966428
10	99.240140
20	99.239680
30	99.239680
40	99.239635
50	99.239635
60	99.239635
70	99.239635
80	99.239635
90	99.239635
100	99.239635
110	99.239635
120	99.239635
130	99.239635
140	99.239635
150	99.239635
160	99.239635
170	99.239635
180	99.239635
190	99.239635
200	99.239635

Optimal State Vector: $\mathbf{x} = (3.514237, 0.000000, 1.000000)^T$

Objective Function Value: $J(\mathbf{x}, \mathbf{y}) = 99.239635$

II. General Remarks

This problem was originally proposed by Kocis and Grossmann (1989). It is a process synthesis model in which the objective is to select two candidate reactors in order to minimise the production cost. The version solved here is that in which the equation constraints have been eliminated using the algebraic manipulations as alluded to in § 3.4.3 above.

The problem was subsequently tackled by others using various techniques, and the latest effort appears to be that by Angira and Babu (2002) who used a differential evolution algorithm. The best function value (as reported by Angira and Babu (2002) and, presumably all previous authors cited therein) is 99.245209. As can be seen above, GENO finds a better solution within 50 generations.

[98] **Source**: Babu and Angira (2002)

Copyright © 1997-2006: Ike's Research Ltd

III. **The gep File**: Example 3.16

```
// Single-objective constrained static optimization problem
// source: Babu and Angira (2002)

#definecs p_maxgens  500
#definecs p_popsize  20
#definecs p_agents   1
#definecs p_order    3
#definecs p_plan     1

#include static_gep_defaults.src

let vars[p_agents, p_order]   = 1   1   1;
let discrete_var[p_order]     = 0   0   1;

adj_mode          = "s";
solution_type     = "e";
maximise          = false;

timer             = true;
sol_mtx_check     = true;
constraints_check = true;

//cross-over probabilities
p_s_xover         = 0.55;
p_a_xover         = 0.55;
p_b_xover         = 0.00;
p_h_xover         = 0.55;
p_d_xover         = 0.55;
p_shuffle         = 0.00;

d_factor          = 0.80;
quantum_0         = 0.1;
rand_seed         = 240657;

proc (1) = m_rate(i,d); retp(0.05); endp;

proc (1) = bm_rate(d); retp(0.005); endp;

proc (2) = f(i,d,v_array);
local c,fv,u,x,z;
   u = matinit(order, plan, 0);
   x = matinit(order, horizon, 0);
  {u,x} = assign_sequences(i,d,u,x);

   //evaluate constraints
   c = constraints(0,x,horizon);
   v_array = evaluate_constraints(c,v_array);

   //evaluate objective
   fv = objective(0,x,horizon);
   retp (fv,v_array);
endp;

proc (1) = objective(z,x,k);
local fv,fv1,fv2;
   fv1 = 0.8*(1 - exp(-0.4*x[2,k]));
   fv2 = 0.9*(1 - exp(-0.5*x[1,k]));
   fv = 7.5*x[3,k] + 5.5*(1 - x[3,k]) + 7*x[1,k] + 6*x[2,k]\
        + 50*(1 - x[3,k])/max(small,fv1) + 50*x[3,k]/max(small,fv2);
   if (maximise); fv = fv; else; fv = -fv; endif;
   retp(fv);
endp;

proc (1) = constraints(z,x,k);
local c; c = zeros(5,1);
   c[1] = 0.9*(1 - exp(-0.5*x[1,k])) - 2*x[3,k];
   c[2] = 0.8*(1 - exp(-0.4*x[2,k])) - 2*(1 - x[3,k]);
   c[3] = x[1,k] - 10*x[3,k];
   c[4] = x[2,k] - 10*(1 - x[3,k]);
   c[5] = x[3,k] - 1;
   retp (c);
endp;
```


Copyright © 1997-2006: Ike's Research Ltd

VI. **The dat File**: Example 3.16

Table 3.16: Input Data for Process Synthesis II

VARIABLE NAME		X1	X2	X3
UCB		10	10	1
LCB		0	0	0
USB		10	10	1
LSB		0	0	0
Initial State Vector		0	0	0
Final State Vector		0	0	0
Discrete Values		0	0	1
Inter-connexion Matrix	X1	1	0	0
	X2	0	1	0
	X3	0	0	1

V. **Points to Note**: Example 3.16

1. `Method` — This is a standard implementation of version MPa of the original mathematical program. The 0-1 integer variables are easily accommodated by declaring the matrix `discrete_Var` and making the appropriate entries in the dat file as illustrated above. But note that on other type of programs involving discrete variables (e.g. Example 3.4 in this Manual), the type of entries made in the dat file and the indicator matrix `discrete_var` should not be confused: the former is always a 0-1 indicator matrix with a '1' entry for each discrete variable, and a '0' entry otherwise; the later should contain the actual discrete values for each discrete variable, and an entry of '0' for all non-discrete variables. Note that in this particular case, the discrete value just happens to be 1.

2. `solution control` — Note the objective includes terms of the form

$$\frac{(1-x_3)}{0.8[1-\exp(-0.4x_2)]}$$

These can cause the program to break down when the denominator vanishes. The code fragment `max(small,fv1)` ensures that there is no dividion by zero during the execution of the program.

Copyright © 1997-2006: Ike's Research Ltd

Example 3.17: Mixed Integer Non-linear Programming: Process Synthesis III [99]

$$\min_{\mathbf{x},\,\mathbf{y}} J(\mathbf{x},\mathbf{y}) = \mathbf{a}^T\mathbf{x} + \mathbf{b}^T\mathbf{y} + \exp(x_1) + \exp\left(\frac{x_2}{1.2}\right) - 60\ln(x_4 + x_5 + 1) + 140$$

Subject to:

$$-\ln(x_4 + x_5 + 1) \le 0\,; \qquad -x_1 - x_2 - 2x_3 + x_4 + 2x_6 \le 0$$

$$-x_1 - x_2 - 0.75x_3 + x_4 + 2x_6 \le 0\,; \qquad x_3 - x_6 \le 0$$

$$2x_3 + x_4 - 2x_6 \le 0\,; \qquad 0.2x_4 - x_5 \le 0$$

$$\exp(x_1) - 10y_1 - 1 \le 0\,; \qquad \exp\left(\frac{x_2}{1.2}\right) - 10y_2 - 1 \le 0$$

$$1.25x_3 - 10y_3 \le 0\,; \qquad x_5 - 0.5x_4 \le 0$$

$$x_4 + x_5 - 10y_4 \le 0\,; \qquad -2x_3 + 2x_6 - 10y_5 \le 0$$

$$y_1 + y_2 = 1\,; \qquad y_4 + y_5 - 1 \le 0$$

$$\mathbf{a} = (-10, -15, -15, 15, 5, -20)^T\,; \qquad \mathbf{b} = (5, 8, 6, 10, 6)^T$$

$$\mathbf{c} = (2, 2, 2, 2, 2, 3)^T\,; \qquad \mathbf{0} \le \mathbf{x} \le \mathbf{c}\,; \quad \mathbf{y} \in \{0, 1\}^5$$

I. GENO Output

Generation	*Objective*
0	88.945294
50	73.053334
100	73.048097
150	73.048097
200	73.048085
250	73.048085
300	73.048085
350	73.048082
400	73.048081
500	73.048081

Optimal State Vector: $\mathbf{x}$ = (0.000000, 2.000000, 1.100000, 0.625000, 0.312500, 1.100000) ^{T}

Optimal discrete Vector: $\mathbf{y}$ = (0, 1, 1, 1, 1, 0) ^{T}

Objective Function Value: $J(\mathbf{x}, \mathbf{y})$ = 73.048081

II. General Remarks

This problem was originally proposed by Duran and Grossmann (1986) and was subsequently tackled by others using various techniques; the latest effort appears to be that by Goyal and Ierapetritou (2005) who used an algorithm that is based on simplicial approximation. The best known solution is:

Best Objective Function Value: $J(\mathbf{x}, \mathbf{y})$ = 73.035

Optimal Control Sequence: $\mathbf{x}$ = (0.000000, 2.000000, 1.078000, 0.652000, 0.326000, 1.078000) ^{T}

Optimal Control Sequence: $\mathbf{y}$ = (0, 1, 1, 1, 1, 0) ^{T}

As can be seen above, a straight forward implementation of version MPa of the M-program yields a nearly-optimal solution.

[99] **Source**: Duran and Grossmann (1986).

Copyright © 1997-2006: Ike's Research Ltd

III. **The gep File**: Example 3.17

```
// A constrained uni-objective static optimization problem
// Source: Duran and Grossmann (1986).

#definecs p_maxgens  200
#definecs p_popsize  20
#definecs p_agents   1
#definecs p_order    10
#definecs p_plan     1

#include static_gep_defaults.src

let vars[p_agents, p_order]   = 1   1   1   1   1   1   1   1   1   1;
let discrete_var[p_order]     = 0   0   0   0   0   0   1   1   1   1;

adj_mode              = "s";
solution_type         = "e";
maximise              = false;

timer                 = true;
sol_mtx_check         = true;
constraints_check     = true;

//cross-over probabilities
p_s_xover          = 0.55;
p_a_xover          = 0.55;
p_b_xover          = 0.00;
p_h_xover          = 0.55;
p_d_xover          = 0.55;
p_shuffle          = 0.55;

d_factor           = 0.80;
quantum_0          = 0.1;
rand_seed          = 240657;

proc (1) = m_rate(i,d); retp(0.05); endp;

proc (1) = bm_rate(d); retp(0.005); endp;

proc (2) = f(i, d, v_array);
local c,fv,u,x,z;
   u = matinit(order, plan, 0);
   x = matinit(order, horizon, 0);
  {u,x} = assign_sequences(i,d,u,x);

   //evaluate constraints
   z = equations(x,0,horizon);
   c = constraints(z,x,horizon);
   v_array  = evaluate_constraints(c,v_array);
   v_array[2] = v_array[2] + distance_to_set(0,1,z[1]);

   //evaluate objective
   fv = objective(z,x,horizon);
   retp (fv,v_array);
endp;


proc (1) = objective(z,x,k);
local fv;
   fv = 140 + 5*z[1] + 8*x[7,k] + 6*x[8,k] + 10*x[9,k] + 6*x[10,k] - 10*x[1,k] \
      - 15*x[2,k] - 15*x[3,k] + 15*x[4,k] + 5*x[5,k] - 20*x[6,k] \
      - + exp(x[1,k]) + exp(x[2,k]/1.2) - 60*ln(x[4,k] + x[5,k] + 1);
   if (maximise); fv = fv; else; fv = -fv; endif;
   retp(fv);
endp;


proc (1) = equations(x,y,k);
local z; z = zeros(1,1);
   z[1] = 1 - x[7,k];
   retp (z);
endp;
```


Copyright © 1997-2006: Ike's Research Ltd

```
proc (1) = constraints(z,x,k);
local c; c = zeros(14,1);
   c[1] = -ln(x[4,k] + x[5,k] + 1);
   c[2] = x[4,k] + 2*x[6,k] - x[1,k] - x[2,k] - 2*x[3,k];
   c[3] = x[4,k] + 2*x[6,k] - x[1,k] - x[2,k] - 0.75*x[3,k];
   c[4] = x[4,k] - x[6,k];
   c[5] = 2*x[4,k] - x[5,k] - 2*x[6,k];
   c[6] = x[5,k] - 0.5*x[4,k];
   c[7] = 0.2*x[4,k] - x[5,k];
   c[8] = x[9,k] + x[10,k] - 1;
   c[9] = exp(x[1,k]) - 10*z[1] - 1;
   c[10] = exp(x[2,k]/1.2) - 10*x[7,k] - 1;
   c[11] = 1.25*x[3,k] - 10*x[8,k];
   c[12] = x[4,k] + x[5,k] - 10*x[9,k];
   c[13] = 2*x[6,k] - 2*x[4,k] - 10*x[10,k];
   c[14] = x[9,k] + x[10,k] - 1;
   retp (c);
endp;
```

VI. **The dat File**: Example 3.17

Table 3.17: Input Data for Process Synthesis III

VARIABLE NAME		X1	X2	X3	X4	X5	X6	X7	X8	X9	X10
UCB		2	2	2	2	2	3	1	1	1	1
LCB		0	0	0	0	0	0	0	0	0	0
USB		2	2	2	2	2	3	1	1	1	1
LSB		0	0	0	0	0	0	0	0	0	0
Initial State Vector		0	0	0	0	0	0	0	0	0	0
Final State Vector		0	0	0	0	0	0	0	0	0	0
Discrete Values		0	0	0	0	0	0	1	1	1	1
Inter-connexion Matrix	X1	1	0	0	0	0	0	0	0	0	0
	X2	0	1	0	0	0	0	0	0	0	0
	X3	0	0	1	0	0	0	0	0	0	0
	X4	0	0	0	1	0	0	0	0	0	0
	X5	0	0	0	0	1	0	0	0	0	0
	X6	0	0	0	0	0	1	0	0	0	0
	X7	0	0	0	0	0	0	1	0	0	0
	X8	0	0	0	0	0	0	0	1	0	0
	X9	0	0	0	0	0	0	0	0	1	0
	X10	0	0	0	0	0	0	0	0	0	1

V. **Points to Note**: Example 3.17

1. `Method` — This is a standard implementation of version MPa of the original mathematical program. The 0-1 integer variables are easily accommodated by declaring the matrix `discrete_var` and making the appropriate entries in the dat file as illustrated above. But note that on other type of programs involving discrete variables (e.g. Example 3.4 in this Manual), the type of entries made in the dat file and the indicator matrix `discrete_var` should not be confused: the former is always a 0-1 indicator matrix with a '1' entry for each discrete variable, and a '0' entry otherwise; the later should contain the actual discrete values for each discrete variable, and an entry of '0' for all non-discrete variables. Note that in this particular case, the discrete value just happens to be 1.

Copyright © 1997-2006: Ike's Research Ltd

Example 3.18: Generalised Disjunctive Programming: Process Synthesis IV [100]

$$\min_{\mathbf{x}} J(\mathbf{x},\mathbf{y}) = (x_1+3)^2 + (x_2-2)^2 + c$$

Subject to: $Y_1 \veebar Y_2 \veebar Y_3$ [101]

Where: $Y_1 \Rightarrow (x_1^2 + x_2^2 - 1 \le 0) \wedge (c = 2)$ [102]

$Y_2 \Rightarrow ((x_1-4)^2 + (x_2-1)^2 - 1 \le 0) \wedge (c = 1)$

$Y_3 \Rightarrow ((x_1-2)^2 + (x_2-4)^2 - 1 \le 0) \wedge (c = 3)$

$x_i \in [0,8],\ i = 1,2\ ;\ \ Y_i \in \{\text{True}, \text{False}\},\ i = 1,2,3$

I. GENO Output

Generation	*Objective*
0	3.340000
10	1.180000
20	1.174100
30	1.171716
40	1.171716
50	1.171640
60	1.171581
70	1.171575
80	1.171573
90	1.171573
100	1.171573
110	1.171573
120	1.171573
130	1.171573
140	1.171573
150	1.171573
160	1.171573
170	1.171573
180	1.171573
190	1.171573
200	1.171573

Optimal State Vector: $\mathbf{x}_T$ = (3.292887, 1.707100, 1.000000, 0.000000)^T

Objective Function Value: $J(\mathbf{x}, \mathbf{y})$ = 1.171573

II. General Remarks

This is a simple Generalised Disjunctive Program (GDP) whose purpose is to illustrate how GENO may be programmed to solve such problems via the Big-M relaxation method. Techniques for converting a GDP into a nonlinear mixed-integer program may be found in Raman and Grossmann (1991); and Grossmann (2002) reviews deterministic numerical algorithmics that are currently available for solving MINLPs.

As can be seen above, a straight forward implementation of version MPa of the M-program yields the optimal solution. Also note that unlike with other (deterministic) solution algorithms, the Big-M parameters needn't be known in advance.

---

[100] **Source**: Lee and Grossmann (2000).

[101] The notation $\veebar$ denotes the 'exclusive-OR' logical operator, and the proposition $Y_1 \veebar Y_2 \veebar Y_3$ asserts that exactly one constraint is operational at any time.

[102] Algebraic expressions may be interpreted as logic propositions (see e.g. Denvir: 1986).

Copyright © 1997-2006: Ike's Research Ltd

III. **The gep File**: Example 3.18

```
// A constrained uni-objective static optimization problem
// Source: Lee and Grossmann (2000).

#definecs p_maxgens  200
#definecs p_popsize  20
#definecs p_agents   1
#definecs p_order    4
#definecs p_plan     1

#include static_gep_defaults.src

let vars[p_agents, p_order]   = 1   1   1   1;
let discrete_var[p_order]     = 0   0   1   1;

adj_mode              = "s";
solution_type         = "e";
maximise              = false;

timer                 = true;
sol_mtx_check         = true;
constraints_check     = true;

//cross-over probabilities
p_s_xover          = 0.55;
p_a_xover          = 0.55;
p_b_xover          = 0.00;
p_h_xover          = 0.55;
p_d_xover          = 0.55;
p_shuffle          = 0.55;

d_factor           = 0.80;
quantum_0          = 0.1;
rand_seed          = 240657;

proc (1) = m_rate(i,d); retp(0.05); endp;

proc (1) = bm_rate(d); retp(0.005); endp;

proc (2) = f(i, d, v_array);
local c,fv,u,x,z;
   u = matinit(order, plan, 0);
   x = matinit(order, horizon, 0);
  {u,x} = assign_sequences(i,d,u,x);

   //evaluate constraints
   z = equations(x,0,horizon);
   c = constraints(z,x,horizon);
   v_array  = evaluate_constraints(c,v_array);
   v_array[2] = v_array[2] + distance_to_set(0,1,z[1]);

   //evaluate objective
   fv = objective(z,x,horizon);
   retp (fv,v_array);
endp;


proc (1) = objective(z,x,k);
local fv;
   fv = (x[1,k] - 3)^2 + (x[2,k] - 2)^2 + 2*z[1] + x[3,k]+ 3*x[4,k];
   if (maximise); fv = fv; else; fv = -fv; endif;
   retp(fv);
endp;


proc (1) = equations(x,y,k);
local z; z = zeros(1,1);
   z[1] = 1 - x[3,k] - x[4,k];
   retp (z);
endp;
```


Copyright © 1997-2006: Ike's Research Ltd

```
proc (1) = constraints(z,x,k);
local c, M; c = zeros(3,1);
   M = large;
   c[1] =  x[1,k]^2 + x[2,k]^2 - 1 - M*(1 - z[1]);
   c[2] = (x[1,k] - 4)^2 + (x[2,k] - 1)^2 - 1 - M*(1 - x[3,k]);
   c[3] = (x[1,k] - 2)^2 + (x[2,k] - 4)^2 - 1 - M*(1 - x[4,k]);
   retp (c);
endp;
```

VI. **The dat File**: Example 3.18

Table 3.18: Input Data for Process Synthesis IV

VARIABLE NAME		X1	X2	X3	X4
UCB		8	8	1	1
LCB		0	0	0	0
USB		8	8	1	1
LSB		0	0	0	0
Initial State Vector		0	0	0	0
Final State Vector		0	0	0	0
Discrete Values		0	0	1	1
Inter-connexion Matrix	X1	1	0	0	0
	X2	0	1	0	0
	X3	0	0	1	0
	X4	0	0	0	1

V. **Points to Note**: Example 3.18

1. `MINLP reformulation` — Logic propositions can always be converted into algebraic inequalities by associating a binary variable with each literal.[103] By this device, the proposition

$$Y_1 \veebar Y_2 \veebar Y_3$$

translates into the linear inequality

$$y_1 + y_2 + y_3 = 1 .$$

The Big-M relaxation of the inequality in logic propositions such as

$$Y_1 \Rightarrow (x_1^2 + x_2^2 - 1 \le 0) \wedge (c = 2)$$

is simply

$$x_1^2 + x_2^2 - 1 \le M(1 - y_1) .$$

Note 1: Y_1 = True implies c = 2 and Y_1 = False implies c = 0, hence the code fragment `2*z[1]` in `proc (1) = objective(z,x,k)`.

Note 2: Y_1 = True implies $y_1 = 1$ which in turn implies that the inequality $x_1^2 + x_2^2 - 1 \le 0$ applies. On the other hand, Y_1 = False implies $y_1 = 0$ which in turn implies that the inequality $x_1^2 + x_2^2 - 1 \le 0$ is irrelevant.

2. `Method` — After reformulating the GDP as an MINLP as outlined above, the problem implementated as version MPa of the original mathematical program. The 0-1 integer variables are accommodated by declaring the matrix `discrete_var` and making the appropriate entries in the dat file as illustrated above.

[103] See Raman and Grossmann (1991) and references cited therein.

Copyright © 1997-2006: Ike's Research Ltd

Example 3.19: Generalised Disjunctive Programming: Process Synthesis V [104]

$$\min_{\mathbf{x},\,\mathbf{y}} J(\mathbf{x},\mathbf{y}) = \mathbf{a}^T\mathbf{x} + \mathbf{b}^T\mathbf{y} + \exp(x_1) + \exp\left(\frac{x_2}{1.2}\right) - 60\ln(x_4 + x_5 + 1) + 140$$

Subject to:

$$-\ln(x_4 + x_5 + 1) \le 0\,; \qquad -x_1 - x_2 - 2x_3 + x_4 + 2x_6 \le 0$$

$$-x_1 - x_2 - 0.75x_3 + x_4 + 2x_6 \le 0\,; \qquad x_3 - x_6 \le 0$$

$$2x_3 + x_4 - 2x_6 \le 0\,; \qquad 0.2x_4 - x_5 \le 0$$

$$x_5 - 0.5x_4 \le 0\,; \qquad Y_1 \vee Y_2 \vee Y_3 \vee Y_4 \vee Y_5$$

$$(Y_1 \vee \neg Y_2) \vee (\neg Y_1 \vee Y_2)\,; \qquad \neg Y_4 \vee \neg Y_5$$

Where:

$$Y_1 \Rightarrow (\exp(x_1) - 11 \le 0) \wedge (c_1 = 5)\,; \qquad \neg Y_1 \Rightarrow (x_1 = 0) \wedge (c_1 = 0)$$

$$Y_2 \Rightarrow (\exp\left(\frac{x_2}{1.2}\right) - 11 \le 0) \wedge (c_2 = 8)\,; \qquad \neg Y_2 \Rightarrow (x_2 = 0) \wedge (c_2 = 0)$$

$$Y_3 \Rightarrow (1.25x_3 - 10 \le 0) \wedge (c_3 = 6)\,; \qquad \neg Y_3 \Rightarrow (x_3 = 0) \wedge (c_3 = 0)$$

$$Y_4 \Rightarrow (x_4 + x_5 - 10 \le 0) \wedge (c_4 = 10)\,; \qquad \neg Y_3 \Rightarrow (x_4 = 0) \wedge (x_5 = 0) \wedge (c_4 = 0)$$

$$Y_5 \Rightarrow (-2x_3 + 2x_6 - 10 \le 0) \wedge (c_5 = 6)\,; \qquad \neg Y_5 \Rightarrow (x_6 - x_3 \le 0) \wedge (c_5 = 0)$$

$$\mathbf{a} = (-10, -15, -15, 15, 5, -20)^T\,; \qquad \mathbf{b} = (5, 8, 6, 10, 6)^T$$

$$\mathbf{c} = (2, 2, 2, 2, 2, 3)^T\,; \qquad \mathbf{0} \le \mathbf{x} \le \mathbf{c}\,; \quad Y_i \in \{\text{True, False}\},\ \forall i$$

I. GENO Output

Generation	*Objective*
0	125.990343
50	73.053334
100	73.048086
150	73.048086
200	73.048085
300	73.048085
400	73.048085
500	73.048085

Optimal State Vector: $\mathbf{x} = (0.427900, 2.000000, 1.100000, 0.625000, 0.312500, 1.100000)^T$

Optimal discrete Vector: $\mathbf{y} = (0, 1, 1, 1, 1, 0)^T$

Objective Function Value: $J(\mathbf{x}, \mathbf{y}) = 73.048085$

II. General Remarks

This is a repeat of Example 3.17 but this time expressed as a Generalised Disjunctive Program. This example also illustrates how disjunctions that involve propositions of the form $\neg Y_i \Rightarrow (x_i = 0) \wedge (c_i = 0)$ are treated.[105] As can be seen above, a straight forward implementation of version MPa of the M-program yields a nearly-optimal solution. Note that unlike other solution algorithms where a judicious choice of the Big-M parameter is imperative (Lee, S. and I. E. Grossmann: 2005), here this parameter needn't be known in advance.

[104] **Source**: Grossmann and Lee (2002).

[105] Such disjunctions are "absorbed" into the *entire* M-program resulting in "binary-modulated" terms such x_3y_1.

Copyright © 1997-2006: Ike's Research Ltd

III. **The gep File**: Example 3.19

```
// A constrained uni-objective static optimization problem
// Source: Grossmann and Lee (2002).

#definecs p_maxgens  200
#definecs p_popsize  30
#definecs p_agents   1
#definecs p_order    10
#definecs p_plan     1

#include static_gep_defaults.src

let vars[p_agents, p_order]   = 1   1   1   1   1   1   1   1   1   1;
let discrete_var[p_order]     = 0   0   0   0   0   0   1   1   1   1;

adj_mode              = "s";
solution_type         = "e";
maximise              = false;

timer                 = true;
sol_mtx_check         = true;
constraints_check     = true;

//cross-over probabilities
p_s_xover          = 0.55;
p_a_xover          = 0.55;
p_b_xover          = 0.00;
p_h_xover          = 0.55;
p_d_xover          = 0.55;
p_shuffle          = 0.55;

d_factor           = 0.250;
quantum_0          = 0.1;
rand_seed          = 2406525;

proc (1) = m_rate(i,d); retp(0.05); endp;

proc (1) = bm_rate(d); retp(0.003); endp;

proc (2) = f(i, d, v_array);
local c,fv,u,x,z;
   u = matinit(order, plan, 0);
   x = matinit(order, horizon, 0);
  {u,x} = assign_sequences(i,d,u,x);

   //evaluate constraints
   z = equations(x,0,horizon);
   c = constraints(z,x,horizon);
   v_array  = evaluate_constraints(c,v_array);
   v_array[2] = v_array[2] + distance_to_set(0,1,z[1]);

   //evaluate objective
   fv = objective(z,x,horizon);
   retp (fv,v_array);
endp;


proc (1) = objective(z,x,k);
local fv;
   fv = 5*z[1] + 8*x[7,k] + 6*x[8,k] + 10*x[9,k] + 6*x[10,k];
   fv = fv + 140 - 10*x[1,k]*z[1] - 15*x[2,k]*x[7,k] - 15*x[3,k]*x[8,k] \
      + 15*x[4,k]*x[9,k]  + 5*x[5,k]*x[9,k] - 20*x[6,k] + exp(x[1,k]*z[1]) \
      + exp((x[2,k]*x[7,k])/1.2) - 60*ln(x[4,k]*x[9,k] + x[5,k]*x[9,k] + 1);
   if (maximise); fv = fv; else; fv = -fv; endif;
   retp(fv);
endp;


proc (1) = equations(x,y,k);
local z; z = zeros(1,1);
   z[1] = 1 - x[7,k];
   retp (z);
endp;
```


Copyright © 1997-2006: Ike's Research Ltd

```
proc (1) = constraints(z,x,k);
local c, M; c = zeros(15,1);
   M = large;
   c[1] = -ln(x[4,k]*x[9,k] + x[5,k]*x[9,k] + 1);
   c[2] = x[4,k]*x[9,k] + 2*x[6,k] - x[1,k]*z[1] - x[2,k]*x[7,k] - 2*x[3,k]*x[8,k];
   c[3] = x[4,k]*x[9,k] + 2*x[6,k] - x[1,k]*z[1] - x[2,k]*x[7,k] - 0.75*x[3,k]*x[8,k];
   c[4] = x[3,k]*x[8,k] - x[6,k];
   c[5] = 2*x[3,k]*x[8,k] - x[4,k]*x[9,k] - 2*x[6,k];
   c[6] = x[5,k]*x[9,k] - 0.5*x[4,k]*x[9,k];
   c[7] = 0.2*x[4,k]*x[9,k] - x[5,k]*x[9,k];
   c[8] = exp(x[1,k]*z[1]) -  11;
   c[9] = exp((x[2,k]*x[7,k])/1.2) - 11;
   c[10] = 1.25*x[3,k]*x[8,k] - 10;
   c[11] = x[4,k]*x[9,k] + x[5,k]*x[9,k] - 10;
   c[12] = 2*x[6,k] - 2*x[3,k]*x[8,k] - 10 - M*(1 - x[10,k]);
   c[13] = x[6,k] - x[3,k]*x[8,k] - M*x[10,k];
   c[14] = 1 - z[1] - x[7,k] - x[8,k] - x[9,k] - x[10,k];
   c[15] = x[9,k] + x[10,k] - 1;
   retp (c);
endp;
```

VI. **The dat File**: Example 3.19

Table 3.19: Input Data for Process Synthesis V

VARIABLE NAME		X1	X2	X3	X4	X5	X6	X7	X8	X9	X10
UCB		2	2	2	2	2	3	1	1	1	1
LCB		0	0	0	0	0	0	0	0	0	0
USB		2	2	2	2	2	3	1	1	1	1
LSB		0	0	0	0	0	0	0	0	0	0
Initial State Vector		0	0	0	0	0	0	0	0	0	0
Final State Vector		0	0	0	0	0	0	0	0	0	0
Discrete Values		0	0	0	0	0	0	1	1	1	1
Inter-connexion Matrix	X1	1	0	0	0	0	0	0	0	0	0
	X2	0	1	0	0	0	0	0	0	0	0
	X3	0	0	1	0	0	0	0	0	0	0
	X4	0	0	0	1	0	0	0	0	0	0
	X5	0	0	0	0	1	0	0	0	0	0
	X6	0	0	0	0	0	1	0	0	0	0
	X7	0	0	0	0	0	0	1	0	0	0
	X8	0	0	0	0	0	0	0	1	0	0
	X9	0	0	0	0	0	0	0	0	1	0
	X10	0	0	0	0	0	0	0	0	0	1

V. **Points to Note**: Example 3.19

1. `Method` — In the MINLP reformulation, disjunctions of the form such as

$$Y_1 \Rightarrow (\exp(x_1) - 11 \le 0) \wedge (c_1 = 5) \qquad \neg Y_1 \Rightarrow (x_1 = 0) \wedge (c_1 = 0)$$

are eliminated by replacing the continuous variable x_1 and the discrete variable c_1 with the "binary-modulated" terms x_1y_1 and c_1y_1 respectively *throughout* the M-program.

Copyright © 1997-2006: Ike's Research Ltd

Example 3.20: Generalised Disjunctive Programming: Job Shop Scheduling [106]

$$\min_{\mathbf{x}} J(\mathbf{x},\mathbf{y}) = \mathrm{T}$$

Subject to:

$$x_1 + 8 - \mathrm{T} \le 0$$

$$x_2 + 5 - \mathrm{T} \le 0$$

$$x_3 + 6 - \mathrm{T} \le 0$$

$$Y_1 \vee Y_2 \vee Y_3$$

Where:

$$Y_1 \Rightarrow (x_1 - x_3 + 5 \le 0)\,; \quad \neg Y_1 \Rightarrow (x_3 - x_1 + 2 \le 0)$$

$$Y_2 \Rightarrow (x_2 - x_3 + 1 \le 0)\,; \quad \neg Y_2 \Rightarrow (x_3 - x_2 + 6 \le 0)$$

$$Y_3 \Rightarrow (x_1 - x_2 + 5 \le 0)\,; \quad \neg Y_3 \Rightarrow (x_2 - x_1 \le 0)$$

$$\mathrm{T} \in [0,\infty)\,; \quad x_i \in [0,\infty)\,; \quad Y_i \in \{\mathrm{True}, \mathrm{False}\}\,; \quad \forall i$$

I. GENO Output

Generation	*Objective*
0	92193810385651892000.000000
10	6586481546.200001
20	17.270000
30	12.081000
40	11.059000
50	11.012100
60	11.000000
70	11.000000
80	11.000000
90	11.000000
100	11.000000

Optimal State Vector: $\mathbf{x}$ = (3.000000, 0.000000, 1.000000, 0.000000, 1.000000, 0.000000) ^{T}

Objective Function Value: $J(\mathbf{x}, \mathbf{y})$ = 11.000000

II. General Remarks

This is a simple Generalised Disjunctive Program (GDP) whose purpose is to illustrate how GENO may be programmed to solve such problems via the Big-M relaxation method. The techniques for converting a GDP into a nonlinear M-program are outlined in Raman and Grossmann (1991); and Grossmann (2002) reviews the algorithmic techniques that are currently available for solving the resulting MINLP. As can be seen above, a straight forward implementation of version MPa of the M-program yields the optimal solution. Note that unlike other solution algorithms where a judicious choice of the Big-M parameter is imperative because

> "If the value [of M] is too small, then feasible points may be cut off. If [the value] is too large, then the continuous relaxation might be too loose yielding poor lower bounds" (paraphrased from Lee, S. and I. E. Grossmann: 2005)

with GENO, this parameter needn't be known in advance.

[106] **Source**: Lee and Grossmann (2000).

Copyright © 1997-2006: Ike's Research Ltd

III. **The gep File**: Example 3.20

```
// A constrained uni-objective static optimization problem
// Source: Lee and Grossmann (2000)

#definecs p_maxgens  500
#definecs p_popsize  30
#definecs p_agents   1
#definecs p_order    7
#definecs p_plan     1

#include static_gep_defaults.src

let vars[p_agents, p_order]   = 1   1   1   1   1   1   1;
let discrete_var[p_order]     = 0   0   0   1   1   1   0;

adj_mode              = "s";
solution_type         = "e";
maximise              = false;

timer                 = true;
sol_mtx_check         = true;
constraints_check     = true;

//cross-over probabilities
p_s_xover          = 0.55;
p_a_xover          = 0.55;
p_b_xover          = 0.00;
p_h_xover          = 0.55;
p_d_xover          = 0.55;
p_shuffle          = 0.00;

d_factor           = 0.80;
quantum_0          = 0.1;
rand_seed          = 240657;

proc (1) = m_rate(i,d); retp(0.05); endp;

proc (1) = bm_rate(d); retp(0.005); endp;


proc (2) = f(i, d, v_array);
local c,fv,u,x,z;
   u = matinit(order, plan, 0);
   x = matinit(order, horizon, 0);
  {u,x} = assign_sequences(i,d,u,x);

   //evaluate constraints
   c = constraints(0,x,horizon);
   v_array  = evaluate_constraints(c,v_array);

   //evaluate objective
   fv = objective(0,x,horizon);
   retp (fv,v_array);
endp;


proc (1) = objective(z,x,k);
local fv;
   fv = x[7,k];
   if (maximise); fv = fv; else; fv = -fv; endif;
   retp(fv);
endp;
```


Copyright © 1997-2006: Ike's Research Ltd

```
proc (1) = constraints(z,x,k);
local c, M; c = zeros(10,1);
   M = large;
   c[1] = x[1,k] - x[3,k] + 5 - M*(1 - x[4,k]);
   c[2] = x[3,k] - x[1,k] + 2 - M*x[4,k];
   c[3] = x[2,k] - x[3,k] + 1 - M*(1 - x[5,k]);
   c[4] = x[3,k] - x[2,k] + 6 - M*x[5,k];
   c[5] = x[1,k] - x[2,k] + 5 - M*(1 - x[6,k]);
   c[6] = x[2,k] - x[1,k] - M*x[6,k];
   c[7] = x[1,k] + 8 - x[7,k];
   c[8] = x[2,k] + 5 - x[7,k];
   c[9] = x[3,k] + 6 - x[7,k];
   c[10] = 1 - x[4,k] - x[5,k] - x[6,k];
   retp (c);
endp;
```

VI. **The dat File**: Example 3.20

Table 3.19: Input Data for the Job Shop Scheduling Problem

VARIABLE NAME		X1	X2	X3	X4	X5	X6	X7
UCB		120	120	120	1	1	1	1e20
LCB		0	0	0	0	0	0	0
USB		120	120	120	1	1	1	1e20
LSB		0	0	0	0	0	0	0
Initial State Vector		0	0	0	0	0	0	0
Final State Vector		0	0	0	0	0	0	0
Discrete Values		0	0	0	1	1	1	0
Inter-connexion Matrix	X1	1	0	0	0	0	0	0
	X2	0	1	0	0	0	0	0
	X3	0	0	1	0	0	0	0
	X4	0	0	0	1	0	0	0
	X5	0	0	0	0	1	0	0
	X6	0	0	0	0	0	1	0
	X7	0	0	0	0	0	0	1

V. **Points to Note**: Example 3.20

1. `Method` — In the MINLP reformulation, propositions of the form

$$Y_i \Rightarrow g_i(\mathbf{x}) \le 0$$

are replaced by Big-M relaxations of the form

$$g_i(\mathbf{x}) \le M(1 - y_i);$$

and propositions of the form

$$\neg Y_j \Rightarrow g_j(\mathbf{x}) \le 0$$

are replaced by Big-M relaxations of the form

$$g_j(\mathbf{x}) \le My_j.$$

Copyright © 1997-2006: Ike's Research Ltd

4. Summary

This document has outlined the usage of a program called GENO, and has presented some numerical results that attest to their efficacy on a variety of problems. The program is easy to use. Apart from the program file (a listing of which is repeated here as a reminder), the user only needs to supply: (a) a source file with a '.gep' filename extension that contains the GAUSS code for the problem at hand; (b) a data file with a '.dat' filename extension that provides the data required. GENO exhibits impressive numerical performance on a wide range of problems. Some salient programming points are listed below.

Program 4.1: A Typical GENO Program

```
new;
#linesoff
   use geno;
   #include ../source/static_opt/sc_sp1.gep;
   load input_data[] = ../data/sc_sp1.dat;
   create sol_fp = ../output/sc_sp1.out with sc, horizon, 8;
   output file = ../output/sc_sp1.txt reset;
   outwidth 250;
   call geno_main;
   output off;
   fp = close(sol_fp);
end;
```

- `Generic type` — GENO solves dynamic optimisation problems of various types whose generic form is given by program MP1 above; the static optimisation case is easily accommodated by merely making the assignment: `p_plan = 1`.

- `dynamics` — When solving actual dynamic optimisation problems, the dynamic constraints are explicitly coded in **`proc`** `a(i,j,k)`, **`proc`** `b(i,j,k)`, **`proc`** `aa(z,w,j,k)`, **`proc`** `bb(z,w,j,k)`; if the state vector is fixed at the final time, then one need to code **`proc`** `ee(z,w,j,k)` as well. On the other hand, if the problem is in fact static, then there is no need for the explicit coding mentioned above because the implied dynamic process is simply $x_{k+1}^{i} = x_{k}^{i} + u_{k}^{i}$, and this is already coded in `static_gep_defaults.src`.

- `ff_sates` — The procedure **`proc`** `ee(z,w,j,k)` is the state equation expressed in "reverse", i.e. it gives $\bar{\mathbf{x}}_k$ in terms of $\bar{\mathbf{x}}_{k+1}$ and $\bar{\mathbf{u}}_k$. This is procedure is crucial to GENO's capability on two-point boundary value problems; and this perhaps highlights a weakness of the program in its current version in that it may not be easy to "factor out" $\bar{\mathbf{x}}_k$ from the coefficient expressions A and B of MP1 in all cases. On two-point boundary value problems, the values of the fixed states should be provided in the data file.

- `Discrete values` — Problems with discrete variables are easily accommodated in GENO; one merely defines the matrix `discrete_vars` and provides the actual values of the discrete variables in the data file.

- `Multiple objectives` — Solving multi-objective problems for various types of solution is easily implemented: one merely needs to appropriately define **`proc`** `f(i,d,v_array)` and set the parameters `adj_mode` and `solution_type` to their appropriate values.

- `Tuning` — On most problems, tuning the algorithm for optimum performance may be achieved by "tweaking" only five parameters: `rand_seed`, `d_factor`, the return values of `m_rate`, `bm_rate`, and occasionally `p_popsize`. However, it should always be remembered that "the quest for a magic set of parameter values that will give the best possible results [in every case] is a fool's errand" (Michalewicz and Fogel: 2000, p.277).

Copyright © 1997-2006: Ike's Research Ltd

5. References

AGHASSI, M., D. Bertsimas and G. Perakis (2005). Solving Asymmetric Variational Inequalities via Convex Optimization. (To appear in *Operations Research Letters*.) [Online]. Available from: http://web.mit.edu/maghassi/www/publications/vi_via_convex_opt.pdf [Accessed June 24, 2006].

AMARÑANZAS, R. and J. A. Lozano (2005). A Multi-Objective Approach to the Portfolio Optimization Problem. *IEEE Congress on Evolutionary Computation (CEC'2005)*, pp. 1388-1395, **2**, IEEE Service Center, Edinburgh, Scotland.

ASAADI, J. (1973). A Computational Comparison of Some Nonlinear Programs. *Mathematical Programming*, **4**, pp. 144-154.

AVRIEL, M. and A. C. Williams (1971). An Extension of Geometric Programming with Applications in Engineering Optimization. *Journal of Engineering Mathematics*, **5**, pp. 187-194.

BABU, B. V. and A. R. Angira (2002). A Differential Evolution Approach for Global Optimization of MINLP Problems. *Proceedings of the Fourth Asia Pacific Conference on Simulated Evolution and Learning (SEAL-2002)*, **2**, pp. 880-884. Singapore.

BÄCK, T. (1993). Optimal mutation rates in genetic search. In Forrest, S. (Ed.). *Proceedings of the Fifth International Conference on Genetic Algorithms*, pp. 2–8. San Mateo, CA: Morgan Kaufmann.

BAGLEY, J. D. (1967). *The Behaviour of Adaptive Systems which Employ Genetic and Correlation Algorithms*. Unpublished Doctoral Dissertation, University of Michigan, Ann Arbor, MI, USA.

BAŞAR, T. and G. J. Olsder (1999). *Dynamic Non-cooperative Game Theory*. SIAM Classics in Applied Mathematics, **23**, SIAM, Philadelphia

BEASLEY, D., D. R. Bull and R. R. Martin (1993a). An Overview of Genetic Algorithms. Part 1, Fundamentals. *University Computing*, **15**, pp. 170-181.

BEASLEY, D., D. R. Bull and R. R. Martin (1993b). An Overview of Genetic Algorithms. Part 2, Research Topics. *University Computing*, **15**, pp. 170-181

BECKER, R., S. Hall and B. Rustem (1994). Robust Optimal Decisions with Stochastic Nonlinear Economic Systems. *Journal of Economic Dynamics and Control*, **18**, pp. 125-147.

CAVICCHIO, Jr., D. J. (1970). *Adaptive Search Using Simulated Evolution*. Unpublished Doctoral Dissertation, University of Michigan, Ann Arbor, MI, USA.

CETIN, B. C., J. Barhen and J. W. Burdick (1993). Terminal Repeller Unconstrained Sub-energy Tunneling (TRUST) for Fast Global Optimization. *Journal of Optimization Theory and Applications*, **77**, pp. 97-126.

CHIPPERFIELD, A. (1997). *Genetic Algorithms in Engineering Systems*, IEE Press.

COELLO COELLO, C. A. (2000). Constraint-handling Using an Evolutionary Multi-objective Optimization Technique. *Civil Engineering and Environmental Systems*, **17**, pp. 319-346.

COELLO COELLO, C. A. (2000). Handling Preferences in Evolutionary Multi-Objective Optimization: A Survey, *Congress on Evolutionary Computation*, **1**, pp. 30-37, Piscataway, New Jersey.

COELLO COELLO, C. A. (2001). A Short Tutorial on Evolutionary Multi-objective Optimization. In Zitzler, E., K. Deb, L. Thiele, C. A. Coello Coello, and D. Corne, (Eds.). *First International Conference on Evolutionary Multi-Criterion Optimization*. Lecture Notes in Computer Science, **1993**, Springer-Verlag, Berlin.

COLVILLE, A. R. (1968). A Comparative Study of Non-linear Programming Codes. *IBM Scientific Centre Report 320-2949*, New York.

CURRIE, D. A. and P. Levine (1993). *Rules, Reputation and Macroeconomic Coordination*. Cambridge University Press, Cambridge

DAS, I. and J. Dennis (1997). A Closer Look at Some Drawbacks of Minimising Weighted Sums of Objectives for Pareto Set Generation in Multi-criteria Optimisation Problems. *Structural Optimisation*, **14**, pp. 63-69.

DAVIS, L. (1989). Adapting Operator Probabilities in Genetic Algorithms. In Schaffer, J. D. (Ed.). *Proceedings of the Third International Conference on Genetic Algorithms*, pp. 61–69. San Mateo, CA: Morgan Kaufmann.

DE JONG, K. A. (1975). *An analysis of the behaviour of a class of genetic adaptive systems*. Unpublished PhD Dissertation, Department of Computer and Communicative Science, University of Michigan, Ann Arbor, MI., U.S.A.

DEB, K. (1997). GeneAS: A Robust Optimal Design Technique for Mechanical Component Design. In D. Dasgupta, and Z. Michalewicz (Eds.). *Evolutionary Algorithms in Engineering Applications*. Springer-Verlag, Berlin.

DEB, K., J. Sundar, U.B. Rao and S. Chaudhuri (2006). Reference Point Based Multi-Objective Optimization Using Evolutionary Algorithms. *Internattional Journal of Computational Intelligence Research*, **2**, pp. 273-286.

DENVIR, T. (1986). *Introduction to Discrete Mathematics for Software Engineering*. MacMillan, London.

Copyright © 1997-2006: Ike's Research Ltd

DURAN, M. and I. E. Grossmann (1986). GeneAS: An Outer Approximation Algorithm for a Class of Mixed Integer Nonlinear Programs. *Mathematical Programming*, **36**, pp. 307 - 339.

FACCHENEI, F. and J. Soares (1997). A New Merit Function for Nonlinear Complementary Problems and a Related Algorithm. *SIAM Journal on Optimization*, **7**, pp. 225-247.

FERRIS, M. C. and C. Kanzow (1998). *Complementarity and Related Problems. A Survey* [Online]. Available from: http://citeseer.ist.psu.edu/ [Accessed July 1, 2005].

FLOUDAS, C. A., A. Aggarwal and A. R. Ciric (1989). Global Optimum Search for Non-convex NLP and MINLP Problems. *Computers and Chemical Engineering*, **13**, pp. 1117 - 1132.

FONSECA, C. M. and P. J. Flemming (1995). An Overview of Evolutionary Algorithms in Multi-objective Optimisation. *Evolutionary Computation*, **3**, pp. 1-16.

GOLDBERG, D. E. (1989). *Genetic Algorithms in Search, Optimisation and Machine Learning*, Addison-Wesley.

GOLDBERG, D. E., and Deb, K. (1991). A Comparative Analysis of Selection Schemes Used in Genetic Algorithms. *Foundations of Genetic Algorithms*, **1**, pp. 69–93.

GOLDBERG, D. E., Deb, K. and Clark, J. H. (1992). Genetic Algorithms, Noise and the Sizing of Populations. *Complex Systems*, **6**, pp. 333–362.

GOLDBERG, D. E., Deb, K. and Thierens, D. (1993). Toward a Better Understanding of Mixing in Genetic Algorithms. *Journal of the Society of Instrument and Control Engineers*, **32**, pp. 10–16.

GOYAL, V. and M. G. Ierapetritou (2005). Computational Studies Using a Simplicial-approximation Based Algorithm for MINLP Optimization. To appear in *Computers and Chemical Engineering*, **?**, pp. ?.

GREFENSTETTE, J. J. (1986). Optimisation of Control Parameters for Genetic Algorithms. *IEEE Transactions on Systems, Man and Cybernetics*, **SMC-16**. pp. 122-128

GROSSMANN, I. E. (2002). Review of Nonlinear Mixed Integer and Generalized Convex Disjunctive Programming Techniques. [Online] Available from: http://egon.cheme.cmu.edu/papers.html [Accessed November 20, 2006].

GROSSMANN, I. E. and Lee, S. (2002). Generalized Convex Disjunctive Programming: Convex Hull Relaxation. [Online] Available from: http://egon.cheme.cmu.edu/papers.html [Accessed October 30, 2006].

HANCOCK, P. J. (1994). An Empirical Comparison of Selection Methods in Evolutionary Algorithms. In Terence C. Fogerty (Ed.). *Evolutionary Computing AISB Workshop*, Lecture Notes in Computer Science, **865**, Springer-Verlag, Berlin.

HARIK, G. R., Cant´u-Paz, E., Goldberg, D. E., and Miller, B. L. (1997). The Gambler's Ruin Problem, Genetic Algorithms and the Sizing of Populations. In Bäck, T. (Ed.), *Proceedings of 1997 IEEE International Conference on Evolutionary Computation*, pp7–12, New York: IEEE Press.

HARKER, P. T. (1984). A Variational Inequality Approach for the Determination of Oligopolistic Market Equilibrium. *Mathematical Programming*, **30**, pp. 105-111.

HEDAR, A. and M. Fukushima. (2005). Derivative-Free Filter Simulated Annealing Method for Constrained Continuous Global Optimization. In G. Di Pillo and F. Giannessi (Eds.). *Nonlinear Optimization and Applications 2*. Kluwer, Amsterdam.

HOCK, W. and K. Schittkowski (1981). *Test Examples for Non-linear Programming Codes*, Lecture Notes in Economics and Mathematical Systems, **187**, Spring-Verlag, Berlin.

HOLLAND, J. H. (1975). *Adaptation in Natural and Artificial Systems*. MIT Press, Cambridge, MA.

HOMAIFAR, A. X. Qi and S. H. Lai (1994). Constrained Optimization via Genetic Algorithms. *Simulation*, **62**, pp. 242-254.

HOOKER, J. N. (2003). Logic-Based Methods for Global Optimization. [Online] Available from: http://egon.cheme.cmu.edu/papers.html [Accessed October 30, 2006].

JOINES, J. A. and C. R. Houck (1994). On the Use of Non-stationary Penalty Functions to Solve Nonlinear Constrained Optimization Problems with GAs. *Proceedings of the IEEE ICEC*, pp. 579-584.

JÖRNSTEN, K. O. (1991). A Method to determine Oligopolistic Market Equilibra Based on a Convex Optimisation Formulation. *Optimisation*, **22**, pp. 439-447

JULSTRÖM, B. A. (1995). What Have You Done for Me Lately? Adapting Operator Probabilities in a Steady-state Genetic Algorithm. In Eschelman, L. (Ed.), *Proceedings of the Sixth International Conference on Genetic Algorithms*, pp. 81–87. San Francisco, CA: Morgan Kaufmann.

KANNAN, B. K. and S. N. Kramer (1994). An Augmented Lagrangian Multiplier Based Method for Mixed Integer Discrete Continuous Optimisation and its Applications to Mechanical Design. *Journal of Mechanical Design. Transactions of the ASME*, **116**, pp. 318-320.

KEARFOTT, R. B. (1996). Interval Computations. Introduction, Uses and Resources. *Euromath Bulletin*, **2**, pp. 95-122.

Copyright © 1997-2006: Ike's Research Ltd

KHOMPATRAPORN, C., Z. Zabinsky and J. D. Pinter (2001). *Comparative Assessment of Algorithms and Software* [Online]. Available from: http://citeseer.ist.psu.edu/ [Accessed August 15, 2005].

KOCIS, G. R. and I. E. Grossmann (1987). Relaxation Strategy for the Structural Optimization of Process Flow Sheets. *Industrial and Engineering Chemistry Research* , **26**, pp. 1869-1880

KOCIS, G. R. and I. E. Grossmann (1988). Global Optimization of a Non-convex Mixed-integer Non-linear Programming (MINLP) Problems in Process Synthesis. *Industrial and Engineering Chemistry Research*, **27**, pp. 1407-1421

KOCIS, G. R. and I. E. Grossmann (1989). A Modelling and Decomposition Strategy for the MINLP Optimization of Process Flow Sheets. *Computers and Chemical Engineering* , **13**, pp. 797-819

KOLSTAD, C. O. and L. Mathiesen (1991). Computing Cournot-Nash Equilibra. *Operations Research*, **39**, pp. 739-748

KOON, G. H. and A. V. Sebald (1995). Some Interesting Test Functions for Evaluating Evolutionary Programming Strategies. In J. R. McDonnel, R. G. Reynolds and D. B. Fogel (Eds.). *Evolutionary Programming IV*, Proceedings of the Fourth International Conference on Evolutionary Programming, MIT Press, Cambridge

KUO, T. and S. Hwang (1996). A Genetic Algorithm with Disruptive Selection. *IEEE Transactions on Systems, Man and Cybernetics*, **26**. pp. 299-307.

LARSON, R. E. and J. L. Casti (1978). *Principles of Dynamic Programming*. Marcel Dekker, New York.

LEE, S. and I. E. Grossmann (2000). New Algorithms for Generalized Disjunctive Programming. *Computers and Chemical Engineering*, **24**, pp. 2125-2141.

LEE, S. and I. E. Grossmann (2005). Logic-based Modelling and Solution of Non-linear Discrete/Continuous Optimization Problems. *Annals of Operations Research*, **139**, pp. 267-288.

LEVY, H. and H. M. Markowitz (1979). Approximating Expected Utility by a Function of Mean and Variance. *American Economic Review*, **69**, pp. 308-317.

LEYFFER, S. (1993). *Deterministic Methods for Mixed Integer Nonlinear Programming*. Unpublished PhD Dissertation, Department of Mathematics and Computer Science, University of Dundee, Dundee, Scotland.

LOBO, F. G. (2000). *The parameter-less Genetic Algorithm: Rationale and Automated Parameter Selection for Simplified Genetic Algorithm Operation*. Unpublished PhD Dissertation, Universidade Nova de Lisboa, Lisboa, Portugal.

LUENBERGER, D. G. (1998). *Investment Science*. Oxford University Press, New York

MARCO, N., J.A., Désidéri, and S. Lanteri (1999). Multi-Objective Optimization in CFD by Genetic Algorithms. *Rapport de Recherche*, **3686**, Institut National de Rechereche en Informatique et en Automatique, Sophia Antipoli, Cédex-France.

MARKOWITZ, H. M. (1952). Portfolio Selection. *Journal of Finance*, **7**, pp. 77-91.

MARKOWITZ, H. M. (1991). Foundations of Portfolio Theory. *Journal of Finance*, **XLVI**, pp. 469-477.

MICHALEWICZ, Z. (1994). *Genetic Algorithms + Data Structures = Evolutionary Programs*. Springer-Verlag, Berlin

MICHALEWICZ, Z. (1995). Genetic Algorithms, Numerical Optimization and Constraints. *Proceedings of the 6th International Conference on Genetic Algorithms*, Pittsburg, July 15-19, pp. 151-158.

MICHALEWICZ, Z. and D. B. Fogel (2000). *How to Solve It. Modern Heuristics*. Springer-Verlag, Berlin

MICHALEWICZ, Z. and N. Attia (1994). Evolutionary Optimisation of Constrained Problems. In Sebald, A. V. and L. J. Fogel (Eds.). *Proceedings of the 3rd Annual Conference on Evolutionary Programming*. World Scientific, River Edge, NJ.

MITCHELL, M. (1996). *Introduction to Genetic Algorithms*, Bradford Books.

MÜHLENBEIN, H. (1992). How Genetic Algorithms Really Work: I. Mutation and Hill-climbing. In Männer, R. and Manderick, B. (Eds.). *Parallel Problem Solving from Nature*, **2**, pp. 15–25. Amsterdam, The Netherlands: Elsevier Science.

MURPHY, F. H., H. D. Sherali and A. L. Soyster (1982). A Mathematical Programming Approach for Determining Oligopolistic Market Equilibrium. *Mathematical Programming*, **26**, pp. 40-47.

NASH, J. (1950). The Bargaining Game. *Econometrica*, **18**, pp. 155-162.

NASH, J. (1951). Non co-operative Games. *Annals of Mathematics*, **54**, pp. 287-295.

NASH, J. (1953). Two-person Co-operative Games. *Econometrica*, **21**, pp. 128-140.

NAYLOR, T. H. and F. Tapon (1982). The Capital Asset Pricing Model: An Evaluation of its Potential as a Strategic Planning Tool. *Management Science*, **28**, pp. 1166-1173.

PARASKEVOPOULOS, D., E. Karakitsos and B. Rustem (1991). Robust Capacity Planning Under Uncertain. *Management Science*, **37**, pp. 787-800.

RADCLIFFE, N. J. (1991). Equivalence Analysis of Genetic Algorithms. *Complex Systems*, **5**. pp. 183-205.

Copyright © 1997-2006: Ike's Research Ltd

RAGSDELL, K. M. and D. S. Phillips (1976). Optimal Design of a Class of Welded Structures using Genetic Algorithms. *ASME Journal of Engineering for Industries*, Series B, **98**, pp. 1021-1025.

RAMAN, R. and I. E. Grossmann (1991). Relation Between MINLP Modelling and Logical Inference for Chemical Process Synthesis. *Computers and Chemical Engineering*, **15**, pp. 73-84.

ROSENMÜLLER, J. and W. Trockel (2001). Game Theory. Working Paper No. 321, *Institute of Mathematical Economics*, University of Beilefeld, Germany.

RUSTEM, B. (1987). Optimal, Time-consistent Robust Feedback Rules Under Parameter, Forecast and Behavioural Uncertainty. In N. M. Christodoulakis (Ed.). *Dynamic Modelling and Control of National Economies*, 8th IFAC Symposium, Edinburgh, June 27-29, 1989.

SANDGREN, E. (1988). Nonlinear Integer and Discrete Programming in Mechanical Design. *Proceedings of ASME Design Technology Conference*, Kissimine, Florida, pp. 95-105.

SCHAFFER, J. D. (1984). *Some Experiments in Machine Learning Using Vector Evaluated Genetic Algorithms*. Unpublished PhD Dissertation, Vanderbilt University, Nashville, U.S.A.

SCHAFFER, J. D., Caruana, R. A., Eshelman, L. J., and Das, R. (1989). A study of Control Parameters Affecting online Performance of Genetic Algorithms for Function Optimization. In Schaffer, J. D. (Ed.). *Proceedings of the Third International Conference on Genetic Algorithms*, pp. 51–60. San Mateo, CA: Morgan Kaufmann.

SCHOENBERG, R. (1993). Markowitz Portfolio Analysis for the Individual Investor. *RSJ Software Company*, Seattle: WA

SHARPE, W. F. (1991). Capital Asset Prices With and Without Negative holdings. *Journal of Finance*, **XLVI**, pp. 469-477.

SIDDALL, J. N. (1972). *Analytical Design Making in Engineering Design*. Prentice-Hall, New Jersey.

SIWALE, I. (2002). GENO: A General Evolutionary Numerical Optimiser. *Technical Report No. RD-1-2002*, Ike's Research Ltd, London

SIWALE, I. (2005). GENO™ 1.0: Supplement to User Manual. *Technical Report No. RD-4-2005*, Ike's Research Ltd, London

SMITH, R. E., and Smuda, E. (1995). Adaptively Re-sizing Populations: Algorithm, Analysis and First Results. *Complex Systems*, **9**, pp. 47-72.

STREICHERT, F., H. Ulmer, and A. Zell (2003). Evolutionary Algorithms and the Cardinality Constrained Portfolio Optimization Problem. In Ahr, D., R. Fahrion, M. Oswald and G. Reinelt (Eds.). *Selected Papers of the International Conference on Operations Research (OR 2003)*, pp. 253-260, Springer-Verlag, Berlin.

SUN, D. and L. Qi (2000). On NCP Functions. *Computational Optimization and Applications*, **1**, pp. 1-21.

THIERENS, D., and Goldberg, D. E. (1993). Mixing in Genetic Algorithms. In Forrest, S. (Ed.). *Proceedings of the Fifth International Conference on Genetic Algorithms*, pp. 38–45. San Mateo, CA: Morgan Kaufmann.

TÖRN, A. and A. Žilinskas (1989). *Global Optimisation*. Springer-Verlag, Berlin.

TROCKEL, W. (1999). Unique Nash Implementation for a Class of Bargaining Solution. *International Game Theory Review*, **1**, pp. 267-272.

WEINBERG, R. (1970). Computer Simulation of a Living Cell. *Dissertations Abstracts International*, **31**, 5312B.

WHITE, M. S. and S. J. Flockton (1995). Modelling the Behaviour of Genetic Algorithms. *Genetic Algorithms in Engineering Systems. Innovations and Applications*, pp. 349-356, IEE Conference Publications No. 414.

WHITLEY, D. (1994). A Genetic Algorithm Tutorial. *Statistics and Computing*, **4**. pp. 65-85.

WHITTLE, P. (1982). *Optimization over Time, Dynamic Programming and Stochastic Control*. Volume 1, Wiley, New York.

WOLPERT, D. H. and W. G. Macready (1997). No Free Lunch Theorems for Optimization. *IEEE Transactions on Evolutionary Computation*, **1**, pp. 67-82.

YU, P. L. (1973). A Class of Solutions for Group Decision Problems. *Management Science*. **19**. pp. 936-946

ZELENY, M. (1976). The Theory of the Displaced Ideal. In Zeleny, M. (Ed.). *Multiple Criteria Decision Making. Kyoto 1975*, pp. 153-206, Springer-Verlag, New York

ZHANG, H. (2001). *Improving Constrained Nonlinear Search Algorithms Through Constraint Relaxation*. Unpublished MSc Dissertation, University of Illinois, Urbana-Champaign, U.S.A.

ZHANG, Y. (2005). *A general Robust-Optimization for Nonlinear Programming*. Unpublished Technical Report TR04-13, Department of Computational and Applied Mathematics, Rice University, Houston, Texas, U.S.A.

Copyright © 1997-2006: Ike's Research Ltd

APPENDICES

I — An Introduction to Genetic algorithms

II — Example GENO Programs: A General Classification

II — Example GENO Programs: Reference to Further Information

III — A Classification of Constrained Nonlinear Programming Problems

III — A Comparative Capability Map of Some Existing NLP Solvers

Copyright © 1997-2006: Ike's Research Ltd

Appendix I

An Introduction to Genetic Algorithms[107]

I. Introduction

The Genetic Algorithm (or GA for short) is a recent development in the arena of numerical search methods. GAs belong to a class of techniques called *Evolutionary Algorithms*, including Evolutionary Strategies, Evolutionary Programming and Genetic Programming. One description of GAs is that they are stochastic search procedures that operate a population of entities; these entities are suitably coded candidate solutions to a specific problem; and the GA solves the problem by artificially mimicking evolutionary processes observed in natural genetics.

Naturally, terminology from the field of natural genetics is used to describe genetic algorithms. In a biological organism, the structure that encodes how the organism is to be constructed is called the *chromosome*. One or more chromosomes may be required to completely specify the organism; the complete set of chromosomes is called a *genotype*, and the resulting organism is called a *phenotype*. Each chromosome comprises a set of *genes*, each with a specific position or *locus* on the chromosome. The loci in conjunction with the values of the genes (which are called the *alleles*) determine what characteristics are expressed by the organism.

In the GA analogy, a problem would first be coded and in this regard, genetic algorithms have traditionally used a binary representation in which each candidate solution is coded as a string of 0's and 1's. The GA's "chromosomes" are therefore the strings of 0's and 1's, each representing a different point in the space of solutions. Normally each candidate solution (or "organism") would have only one chromosome, and so the terms *organism*, *chromosome* and *genotype* are often used synonymously in GA literature. At each position on a chromosome is a *gene* that can take on the *alleles* 0 or 1; the phenotype is the *decoded* value of the chromosome. The population of candidate solutions represent a sample of different points of the search space, and the algorithm's genetic processes (see below) are such that the chromosomes evolve and become better and better approximations of the problem's solution over time.

Before a GA can be run, a *fitness function* is required: this assigns a figure of relative merit to each potential solution. Before fitness values can be assigned, each coded solution has to be decoded and evaluated, and the module designed to do this is generally called the *evaluation function*. But whereas the evaluation function is a problem-specific mapping used to provide a measure of how individuals have performed in the problem domain, the fitness function, on the other hand, is a problem-independent mapping that transforms evaluation values into an allocation of *reproductive* opportunities. For any given problem therefore, different fitness functions may be defined. At each generation, the chromosomes in the current population are rated for their effectiveness as candidate solutions, and a process that emulates nature's survival-of-fittest principle is applied to generate a new population which is then "evolved" using some genetic operators defined on the population. This process is repeated a sufficient number of times until a good-enough solution emerges. The three primary genetic operators focused on in practice are *selection*, *crossover* and *mutation*.

II. Selection

This operator is sometimes called *Reproduction*. The reproduction operation is in fact comprised of two phases: the selection mechanism and the sampling algorithm. The selection mechanism assigns to each individual x, a *positive* real number, called the *target sampling rate* (or simply: *fitness*), which indicates the *expected* number of offspring reproduced by x at generation t. In the commonly used *fitness proportionate selection* method, an individual is assigned a target sampling rate equal to the ratio of the individual's evaluation to the average evaluation of all chromosomes in the population. This simple scheme however suffers from the so-called *scaling problem* where a mere shift of the underlying function can result in significantly different fitness values.[108] A technique that has been suggested to overcome this is to assign target sampling rates according to some form of population ranking scheme. Here, individuals are first assigned a rank based on their performance as determined by the evaluation function; thereafter, the sampling rates are computed as some linear or non-linear function of the ranks.

---

[107] Only a brief description of genetic algorithms is provided here. Readers requiring a more detailed account of the subject should consult standard texts such as Goldberg (1989), Michalewicz (1994) or Mitchell (1996); or survey papers such as Beasley, et al. (1993), Whitley (1994) or Chipperfield (1997). A large part of this exposition closely follows Michalewicz (1994: Chapters 3, 4).

[108] For example, given two chromosomes with evaluation values 2 and 1 say, the first will get twice as many offspring as the second. If the evaluation function is changed by simply adding 1 to all values, the two strings will score 3 and 2, a ratio of only 1.5

Copyright © 1997-2006: Ike's Research Ltd

After fitness values have been assigned, the sampling algorithm then reproduces copies of individuals to form an intermediate mating population. The most common method of sampling the population is by the *roulette wheel* method in which each individual is assigned a slice of a virtual roulette wheel which is proportional to the individual's fitness. To reproduce a population of size *P*, the wheel is spun *P* times. On each spin, the individual under the wheel's maker is selected to be in the mating pool of parents who are to undergo further genetic action. An alternative approach and one which minimises *spread*[109] is Stochastic Universal Sampling (SUS). Here, rather than spin the wheel *P* times to select *P* parents, SUS spins the wheel once but with *P* equally spaced pointers which are used to select the *P* parents simultaneously. Reproduction may also be done by a tournament selection. A typical implementation is as follows. Two individuals are chosen at random from the population and a random number *r* is chosen between 0 and 1. If $r < k$ (where *k* is a tuning parameter, say 0.75), the fitter of the two individuals is selected to go into the mating pool; otherwise the less fit individual is chosen. The two are then returned to the original population and can be selected again.

Reproductive processes may be implemented in *generational* or *steady-state* mode. Generational reproduction replaces the entire population with a new population, and the GA is said to have a generation gap of 1. Steady-state reproduction on the other hand replaces only a few individuals at a time. *Elitism* is an addition to selection that forces the GA to retain some number of the best individuals at each generation. Such individuals can be lost if they are not selected to reproduce or if they are operated on by the genetic operators.

The selection operator is the driving force in GAs, and the *selection pressure*[110] is a critical parameter. Too much selection pressure may cause the GA to converge prematurely; too little pressure makes the GA's progress towards the solution unnecessarily slow.

III. Crossover

The crossover operation is also called *recombination*. It is generally considered to be the main exploratory device of genetic algorithms. This operator manipulates a pair of individuals (called *parents*) to produce two new individuals (called *offspring* or *children*) by exchanging corresponding segments from the parents' coding. The simplest form of this operator is the single-point crossover, and this is as illustrated below where the crossover point is the position marked by the symbol, |.

$$\begin{array}{l} \text{Parent1}:(\text{a b c}\,|\,\text{d e}) \\ \text{Parent2}:(1\,2\,3\,|\,4\,5) \end{array} \xrightarrow{\text{Crossover at position 3}} \begin{array}{l} \text{Child1}:(\text{a b c}\,|\,4\,5) \\ \text{Child2}:(1\,2\,3\,|\,\text{d e}) \end{array}$$

Other binary-coded crossover operators which are variations of the above scheme have since been defined, e.g., two-point crossover, uniform crossover and shuffle crossover. For real-coded GAs, recombination is usually defined in a slightly different way. We mention three crossover operators that are employed by GENO:

- ARITHMETIC CROSSOVER. This operator produces two offspring that are convex combinations of the parents. If the chromosomes $\mathbf{c}_k^v = \left(v_k^1, v_k^2, \ldots, v_k^N\right)$ and $\mathbf{c}_k^w = \left(v_k^1, v_k^2, \ldots, v_k^N\right)$ are selected for crossover, the offspring are defined as:

$$\mathbf{c}_k^1 = \alpha \cdot \mathbf{c}_k^v + (1-\alpha) \cdot \mathbf{c}_k^w \quad \text{and} \quad \mathbf{c}_k^2 = \alpha \cdot \mathbf{c}_k^w + (1-\alpha) \cdot \mathbf{c}_k^v, \quad \text{where} \quad \alpha \sim \mathrm{UD}[0,1].$$

- HEURISTIC CROSSOVER. This operator combines two chromosomes and produces one offspring as follows: if $\mathbf{c}_k^v$ and $\mathbf{c}_k^w$ are two parent chromosomes such that the fitness of $\mathbf{c}_k^v$ is not worse than that of $\mathbf{c}_k^w$, then the offspring is:

$$\mathbf{c}_k^x = \mathbf{c}_k^v + \alpha \cdot (\mathbf{c}_k^v - \mathbf{c}_k^w), \quad \text{where} \quad \alpha \sim \mathrm{UD}[0,1].$$

  Here, the idea is to use the "quasi-gradient" of the evaluation function as a means of directing the search process.

- DIFFERENTIAL CROSSOVER. This operator uses three parents: one parent is taken as the "base", and the other two are used to generate the search direction. Thus, if $\overline{\mathbf{u}}_T^B$, $\overline{\mathbf{u}}_T^v$ and $\overline{\mathbf{u}}_T^w$, are the parent chromosomes with $\overline{\mathbf{u}}_T^B$ as the "base", then the offspring are:

$$\overline{\mathbf{u}}_T^1 = \overline{\mathbf{u}}_T^B + \alpha \cdot (\overline{\mathbf{u}}_T^w - \overline{\mathbf{u}}_T^v) \quad \text{and} \quad \overline{\mathbf{u}}_T^2 = \overline{\mathbf{u}}_T^B + \alpha \cdot (\overline{\mathbf{u}}_T^v - \overline{\mathbf{u}}_T^w), \quad \text{where} \quad \alpha \in [0,1].$$

  In GENO, the factor α is pre-selected from the unit interval, although random variable may also be used.

---

[109]For large populations, a good sampling algorithm should statistically result in the expected number of offspring for each individual. However, with the relatively small populations typically used in GAs, the *actual* number of offspring allocated to an individual is often far from its expected value. The range of possible actual values, given an expected value is called the spread.

[110]A definition of *selection pressure* is that it is the target sampling rate assigned to the best member of the population (Hancock: 1994).

Copyright © 1997-2006: Ike's Research Ltd

IV. Mutation [111]

By modifying one or more of the gene values of an existing individual, mutation creates new individuals and thus increases the variability of the population. The mutation operator ensures that the probability of reaching any point in the search space is never zero. The mutation operator is applied to each gene of the chromosome depending on whether a random deviate drawn from a certain probability distribution is above a certain threshold. Usually the uniform or normal distribution is assumed. Again, depending on the representation adopted, variations of the basic mutation operator may be defined.

V. Why Genetic Algorithms Work

Currently, there are several competing theories that attempt to explain the macroscopic behaviour of GAs. The original description of GAs as *schema processing* algorithms by John Holland (1975) has underpinned most of the theoretical results derived to date. However, other descriptive models based on Equivalence Relations, Walsh functions, Markov Chains and Statistical Mechanics have since been developed. A survey of these models is beyond the scope of this introductory exposition.[112] Instead, we provide a sketch of the logic leading up to one of the main explanatory models, namely *The Building Block Hypothesis*. We begin by stating some definitions.

- DEFINITION A1. [Schema; Schemata] A *schema* is a template that defines the similarities among chromosomes which is built by introducing the *don't care* symbol (*) in the alphabet of genes. It represents all chromosomes which match it at every locus other than the '*' positions. For example, the schema (1 0 * * 1) represents of four chromosomes, i.e.: (1 0 1 1 1), (1 0 1 0 1), (1 0 0 1 1) and (1 0 0 0 1). A collection of schema is a *schemata*.
- DEFINITION A2. [Order] The *order* of a schema *S* (denoted by o(*S*)) is the number of non-*don't care* positions in the schema. For example, the schemata S_1 = (1 0 * * 1), S_2= (* 0 * * 1), S_3= (* * 1 * *) are of orders 3, 2 and 1, respectively.
- DEFINITION A3. [Defining Length] The *defining length* of a schema *S* (denoted by $\delta(S)$) is the *positional distance* between the first and last *fixed* positions (i.e., the non-*don't care* sites) in the schema. It defines the compactness of the information contained in the schema. For example, defining lengths of the three schemata S_1 = (1 0 * * 1), S_2= (* 0 * * 1), S_3= (* * 1 * *) are $\delta(S_1) = 4$, $\delta(S_2) = 3$ and $\delta(S_3) = 0$, respectively.
- DEFINITION A4. [Schema Fitness] The *schema fitness* is the average of the fitness of all chromosomes matched by the schema in a given population. That is, given the evaluation function *eval*(.) defined on a population of chromosomes $\mathbf{x}_j$ of size *P*, the fitness of schema *S* at generation *t* is:

$$\mathrm{eval}(S,t) = \sum_{j=1}^{P} \mathrm{eval}(\mathbf{x}_j)/P \tag{A1}$$

The evolutionary process of GAs consists of four basic steps which are consecutively repeated, namely:

$t \leftarrow t+1$

select $P(t)$ from $P(t-1)$;

recombine and mutate $P(t)$;

evaluate $P(t)$.

The main evolutionary process takes place in the *select*, *recombine* and *mutate* phases. After the selection step, we can expect to have $\xi(S,t+1)$ chromosomes matched by the schema *S* in the mating pool. For an average chromosome matched by the schema *S*, the probability of its selection is equal to $\mathrm{eval}(S,t)/F(t)$ where $F(t)$ is the total fitness for the whole population. Since the number of chromosomes matched by schema *S* before selection is $\xi(S,t)$, and the number of single chromosome selections is *P*, it follows that:

$$\xi(S,t+1) = \xi(S,t) \cdot P \cdot \mathrm{eval}(S,t)/F(t) \tag{A2}$$

or, in terms of the average fitness of the population, $\overline{F}(t) = F(t)/P$:

$$\xi(S,t+1) = \xi(S,t) \cdot \mathrm{eval}(S,t)/\overline{F}(t) \tag{A3}$$

---

[111] Advocates of *Evolutionary Strategies* consider this operator to be more fundamental than cross-over. Their argument is that crossover can essentially be viewed as a series of mutations applied to adjacent genes with probability one.

[112] Readers interested in this aspect may wish to consult Radcliffe (1991); White and Flockton (1995); Mitchell (1996).

Copyright © 1997-2006: Ike's Research Ltd

In other words, the number of chromosomes grows as the ratio of the fitness of the schema to the average fitness of the population. This means that an above-average schema receives an increasing number of matching chromosomes in the next generation, a below-average schema receives a decreasing number of chromosomes, and an average schema remains the same.

The schema growth equation (A3) however has to be modified to take into account the destructive effects of recombination and mutation. For chromosomes of length *m*, the crossover site is selected uniformly from among (*m* - 1) possible sites. A schema *S* would be destroyed if the crossover site is located within its defining length. The probability of this happening is $p_d(S) = \delta(S)/(m-1)$ and, hence, the probability of a schema surviving the crossover operation is,

$$p_s(S) = 1 - \left(\delta(S)/(m-1)\right). \tag{A4}$$

The crossover operator is however only applied selectively according to some probability (p_c, say). Furthermore, even when the crossover site is within the defining length there is always a finite chance that the schema may survive. These considerations dictate the modification of (A4) to,

$$p_s(S) \geq 1 - p_c \cdot \left(\delta(S)/(m-1)\right). \tag{A5}$$

Thus, the combined effects of *selection* and *recombination* are summarised by:

$$\xi(S,t+1) \geq \xi(S,t) \cdot \left(\mathrm{eval}(S,t)/\overline{F}(t)\right) \cdot \left(1 - p_c \cdot \left(\delta(S)/m-1\right)\right). \tag{A6}$$

The mutation operator changes a single gene with probability p_m. It is clear that *all* the fixed positions of a schema must remain intact if the schema is to survive mutation. Since each mutation is independent of all others, the probability of a schema *S* surviving mutation is therefore:

$$p_s(S) = \left(1 - p_m\right)^{o(S)}. \tag{A7}$$

And for $p_m << 1$, $p_s(S)$ may be approximated by the first two terms of its binomial expansion, i.e.:

$$p_s(S) \approx \left\{1 - p_m \cdot o(S)\right\}. \tag{A8}$$

Therefore, ignoring higher-order terms involving products of p_c and p_m, the combined effects of selection, crossover and mutation is summarised by the following reproductive schema growth inequality:

$$\xi(S,t+1) \geq \xi(S,t) \cdot \left(\mathrm{eval}(S,t)/\overline{F}(t)\right) \cdot \left(1 - p_m \cdot o(S) - p_c \cdot \left(\delta(S)/m-1\right)\right). \tag{A9}$$

Clearly, the disruptive effects of mutation and crossover are greater, the higher the order, and the longer the defining length of the schema. One can therefore expect that later generations of chromosomes would increasingly be comprised of short, low-order schemata of above-average fitness. This observation is captured by the *Schema Theorem* which states:

- THEOREM A1. [Schema Theorem] Short, low-order, above-average schemata receive exponentially increasing trials in subsequent generations of a genetic algorithm.

An immediate result of this theorem is the assertion that genetic algorithms explore the search space by short, low-order schemata which are used for information exchange during recombination. This observation is expressed by the Building Block Hypothesis which states:

- HYPOTHESIS A1. [Building Block Hypothesis] A genetic algorithm seeks near-optimal performance through the juxtaposition of short, low-order, high-performance schemata called building blocks.

Over the years, many GA applications which support the building block hypothesis have been developed in many different problem domains. However, despite this apparent explanatory power, the hypothesis is not universally valid. In particular, it is easily violated in the so-called deceptive problems.[113]

[113]A problem is referred to as *deceptive* if the average fitness of schemata that are not contained in the global optimum is greater than the average fitness of those that are. A problem is said to be *fully deceptive* if all low-order schemata containing a sub-optimal solution are better than other competing schemata (Beasley, et al.: 1993, p.173).

Copyright © 1997-2006: Ike's Research Ltd

VI. Setting GA Parameters [114]

Before one can use a GA, one needs to specify some parameter values namely the selection pressure, the population size, and the crossover and mutation rates. Both theoretical and empirical studies show that "optimal" values for these parameters depend on how difficult the problem at hand is. And since prior determination of the degree of difficulty a particular problem poses is hard, there are no generally accepted recipes for choosing effective parameter values in every case. However, many researchers have developed good heuristics for these choices on a variety of problems, and this section outlines some their recommendations.

Experimental Studies

De Jong (1975)

Kenneth A. De Jong tested various combinations of GA parameters on five functions with various characteristics including continuous and discontinuous, convex and non-convex, uni-modal and multi-modal, deterministic and noisy for his PhD Thesis. His function suite has since been adopted by many researchers as *the* standard test bed for assessing GA designs.[115]

De Jong used a simple GA with roulette wheel selection, one-point cross-over and simple mutation to investigate the effects of four parameters namely: population size, crossover rate, mutation rate and the generation gap. His main conclusions were that:

- Increasing the population size resulted in better long-term performance, but smaller population sizes responded faster and therefore exhibited better initial performance
- Mutation is necessary to restore lost alleles but this should be kept low at a low rate for otherwise the GA degenerates into a random search
- A cross-over probability of around 0.6 worked best. But increasing the number of cross-over points degraded performance
- A non-overlapping population model worked better in general
- In summary, he concluded that the following set of parameters were efficient (at least for the functions that he studied): population size — 50 - 100; crossover probability — 0.6; mutation probability — 0.001; generation gap — 1

De Jong's work was very important in that it provided practical guidelines for subsequent applications of GA's. His recommendations for the various parameters have been so widely adopted that they are sometimes referred to as "the standard settings". But subsequence research revealed that applying De Jong's parameter values cases can be a serious mistake in some cases.

Schaffer, Caruana, Eshelman and Das (1989)

Recognising that parameter values can have a significant impact on the performance of a GA and that a more thorough investigation was needed, Schaffer et al. (1989) expanded De Jong's experimental setup. In addition to the five functions that he had studied, they introduced five more and performed a more exhaustive assessment of the direct and cross effects of the various parameters on a GA's performance. A notable observation they made was that good GA performance results from an inverse relationship between population size and the mutation rate, i.e. high mutation rates were better for small population sizes and low mutation rates were good for large populations. Whilst recognizing that their results may not generalise beyond the 10 functions in their test suite, they recommend the following parameter values:

- Population size: 20 - 30
- Mutation rate: 0.005 – 0.1
- Cross-over rate: 0.75 – 0.95

De Jong's work was very important in that it provided practical guidelines for subsequent applications of GA's. His recommendations for the various parameters have been so widely adopted that they are sometimes referred to as the "standard settings".

---

[114] This section is largely sourced from Lobo (2000: Chapter 3).

[115] As regards multi-objective optimisation, Schaffer (1984) suggested some simple bi-objection problems which he used to evaluate his multi-objective genetic algorithm called VEGA. His simple unconstrained two-objective functions comprised *the* test suite for validating most of the evolutionary multi-objective optimisation techniques developed in the ensuing years (Coello Coello, 2001, p.4).

Copyright © 1997-2006: Ike's Research Ltd

- Theoretical Studies

Several researchers have theoretically analysed the dynamics of GAs. In his survey, Lobo (2000: chapter 3) reports the most notable of these as being the work on selection by Goldberg and Deb (1991); the work on mutation by Mühlenbein (1992) and Bäck (1993); the research on population sizing by Goldberg, Deb and Clark (1992) and Harik et al. (1997); and the work on control maps by Goldberg, Deb and Thierens (1993) and Thierens and Goldberg (1993). The insights and practical implications afforded by these studies are summarised below.

- On Selection. In the absence of all other operators, repeated use of the selection operator would eventually result in a population comprised of the single chromosome with the highest fitness. Goldberg and Deb define the *takeover time* as the time it takes (as measured by the number of generations elapsed) for this event to occur. They derived takeover time formulae for various selection schemes and validated these using computer simulations. For fitness proportionate selection schemes, the takeover time depends on the fitness function distribution; for order-based selection, the takeover time is independent of the fitness function and is of the order O (log P), where P is the population size. Obviously the takeover time is increases in the presence of cross-over and mutation, but one must be careful not to exert too much selection pressure to cancel the diversifying effects of these operators.

- On Mutation. Independently of each other Mühlenbein (1992) and Bäck (1993) analysed the effects of mutation on a simple (1 + 1) evolutionary algorithm.[116] They both concluded that for a chromosome of length L, the optimal fixed mutation rate is L^{-1}. Intuitively, it is easy to see why there should be this inverse relationship between chromosome length and mutation rate. Besides exploring the search space, the mutation (and to so extent, cross-over operation) can disrupt building blocks during the course of a GA run. And obviously, this is more likely to occur for long chromosomes than short ones since the operator is applied (with probability) to each gene. So in order to minimise building block disruption, one should *decrease* the mutation rate for relatively *longer* chromosomes.

- On Population Size. Studies on population size attempt to formulate schema growth equations similar to A3 that have population size as a variable. Unfortunately, population sizing equations are difficult to use in practice. Lobo (2000) notes:

  > "In order to apply [the equation] the user has to know or estimate the selective advantage that a building block has over its closest competitors; he has to estimate the building block's fitness variance, he has to estimate the maximum level of deception in the problem at hand; and of course he has to hope that the building blocks are going to mix well, which may not occur in practice" (paraphrased from p.34)

  It is difficult to see how these population sizing studies can be used to further inform the choice of parameter values suggested by the empirical work of De Jong (1975) or Schaffer, et al. (1989).

- On Selection. Increasing the population size resulted in better long-term performance, but smaller population sizes responded faster and therefore exhibited better initial performance

- Parameter Adaptation

We mention, in passing, parameter adaptation techniques. These methods change GA parameters as the search progresses. There are three main approaches: (1) centralised methods change parameter values based on a central learning rule;[117] (2) decentralised methods encode the parameters values into the chromosomes themselves;[118] (3) meta-GA's attempt to optimise parameter values by evolving these values for the actual GA that is run at a lower level using the parameters identified by the meta-level GA.[119] The main advantage of a GA so designed is that the user is no longer required to specify parameter values prior to executing the search.

VII. Concluding Remarks

Genetic Algorithms are simple and yet powerful search and optimisation procedures that are widely applicable. Unfortunately, our current knowledge is such that one cannot rigorously predict whether a GA is going to efficient in any given instance due to the difficult in choosing the parameters of the algorithm. Nevertheless, the parameters recommended for GENO are efficient, at least on the examples reported. These parameters were arrived at after extensive trials guided by the empirical and theoretical work outlined above: they are summarised in Table 1A below.

---

[116] In the evolutionary strategies lexicon, the notation (1 + 1) denotes an algorithm that has one parent and creates one child through mutation; the better of the two becomes the next parent in the next generation.

[117] Cavicchio (1970), Davis (1989), Julström (1995), Smith and Smuda (1995), Lobo and Golberg (1997)

[118] Bagley (1967), Bäck (1992)

[119] Weinberg (1970), Mercer and Sampson (1978), Greffenstette (1986).

Copyright © 1997-2006: Ike's Research Ltd

Table 1A: GENO Parameters: [120] Default Values

PARAMETER	TYPE	DEFAULT VALUE	REMARKS
adj_mode	character	none	This parameter is problem-dependent. It should be set to "s" for uni-objective optimisation problems, or if one seeks a Nash equilibrium solution of a multi-objective problem; it should be to "g" in all other cases.
bm_rate	real matrix	0.005	This parameter is the probability of boundary mutation and is returned from a simple procedure.[121] It is problem-dependent but the default value is usually efficient.
constraints_check	Boolean	false	This parameter it allows one to choose whether or not to display values of the constraints at the end of the program run.
d_factor	real matrix	0.15 – 0.8	This is a weighting factor on the direction component of the differential cross-over operator.[122] The parameter is problem-dependent but the default range is usually efficient.
m_rate	real matrix	0.05	This parameter is the probability of ordinary mutation and is returned by a simple procedure.[123] It is problem-dependent but the default value is usually efficient.
maximise	Boolean	true	This parameter is problem-specific: set this to false if the problem is about minimisation.
p_a_xover	real matrix	0.55	This parameter is problem-dependent: it specifies the probability threshold for the arithmetic cross-over operator. The default value is usually efficient.
p_agents	integer	none	This parameter is problem-specific: it declares the number of sub-objective in a multi-objective problem
p_b_xover	real matrix	0.005	This parameter is problem-dependent: it specifies the probability threshold for the boundary cross-over operator. The default value is usually efficient.
p_d_xover	real matrix	0.55	This parameter is problem-dependent: it specifies the probability threshold for the differential cross-over operator. The default value is usually efficient.
p_eqms	integer	1	A constant equal to 1.[124]
p_h_xover	real matrix	0.55	This parameter is problem-dependent: it specifies the probability threshold for the heuristic cross-over operator. The default value is usually efficient.

[120] The Table is arranged in alphabetical order for easy of reference. More information on some of these parameters may be found on the list entitled **Points to Note**: The gep Source File on pages 6 and 7 of this Manual.

[121] This is in anticipation of future program development.

[122] Infra, page 81.

[123] This is in anticipation of future program development.

[124] This parameter is a legacy of the original GENO design. It is located in the source file `static_gep_defaults.src` (or its dynamic equivalent `dynamic_gep_defaults.src)` in order to minimise clutter in the gep file.

Copyright © 1997-2006: Ike's Research Ltd

Table 1A: Continued

p_maxgens	integer	none	This parameter specifies the maximum number of generations that the algorithm will execute. The most efficient value is dependent on the problem and the population size but it would be safe to assume that GENO solves most problems within 500 generations
p_mingens	integer	2	This parameter should always be 2.
p_order	integer	none	This parameter is problem-specific: it specifies the total number of variables in the problem.
p_plan	integer	none	This parameter is problem-specific: it specifies the length of the control vector along the time dimension
p_popsize	integer	10 - 30	This parameter specifies the population size. The most efficient value is problem-dependent but most likely to be within the range shown
p_s_xover	real matrix	0.55	This parameter is problem-dependent: it specifies the probability threshold for the simple cross-over operator. The default value is usually efficient.
p_shuffle	real matrix	0.55	This parameter is problem-dependent: it specifies the probability threshold for shuffling the population. The default value is usually efficient.
p_u_xover	real matrix	0.55	This parameter is problem-dependent: it specifies the probability threshold for the uniform cross-over operator. The default value is usually efficient.
pos_orth	Boolean	true	This parameter is problem-specific: set this to false if the static constraints are of the 'less than' type.
quantum_0	real matrix	none	This parameter is problem-dependent: is specifies the initial size of quanta. In setting this parameter, the object should be to ensure that the initial population is sufficiently diverse on all dimensions. In this regard, a choice of the smaller between 0.1 and 10% of the smallest variable range is normally efficient. But if one seeks an integer solution, then this parameter should be set to 1.
rand_seed	real matrix	none	This is a seed value for the random number generator.
solution_type	character	none	This parameter is problem-dependent: it defines the type of solution sought.
timer	Boolean	false	This parameter declares whether to display GENO's loop time at the end of the program run
vars	binary matrix	none	This parameter is problem-specific: it is an 'incidence matrix' that shows what variables are in each sub-problem of the multi-agent optimisation problem.
vdu_output	Boolean	false	This parameter declares whether to display progress of the best chromosome or its fitness.
view_vars	Boolean	false	In conjunction with vdu_output, this parameter allows one to choose between viewing the variables in the best chromosome or its fitness.

Copyright © 1997-2006: Ike's Research Ltd

Appendix II

Table 2A: Example GENO programs: A General Classification

PROBLEM TYPE	STATIC OPTIMISATION	DYNAMIC OPTIMISATION
SINGLE OBJECTIVE	su_sp1 su_sp2 Su_sp3 sc_sp1 [*versions*: **a**, **b** & **d**] sc_sp2 [*versions*: **a**, **b** & **c**] sc_sp3 [*versions*: **a**, **aa**, **b** & **d**] sc_sp4 [*versions*: **a**, **aa**, & **b**] sc_sp5 [*versions*: **a**, **b** & **d**] sc_sp6 [*versions*: **a** & **b**] sc_sp7 [*versions*: **a**, **b** & **c**] sc_sp8 [*versions*: **a** & **b**] sc_sp9 [*versions*: **a** & **b**] sc_sp10 [*versions*: **a**, **aa**, & **b**] sc_sp11 [*versions*: **a**, **aa**, & **b**] sc_sp12 [*versions*: **a**, **aa**, & **b**] sc_sp13 [*versions*: **a**, **aa**, & **b**] sc_sp14 [*versions*: **a** & **b**]	sc_dp1 sc_dp2a sc_dp2b sc_dp3 sc_dp4 sc_dp5
MULTIPLE OBJECTIVES	mu_sp1 mu_sp2 mu_sp3 mu_sp4	mc_dp1a mc_dp1b mc_dp2

Legend:

sc_sp — ***S***ingle-objective ***C***onstrained ***S***tatic Optimisation ***P***roblem

su_sp — ***S***ingle-objective ***U***nconstrained ***S***tatic Optimisation ***P***roblem

sc_dp — ***S***ingle-objective ***C***onstrained ***D***ynamic Optimisation ***P***roblem

su_dp — ***S***ingle-objective ***U***nconstrained ***D***ynamic Optimisation ***P***roblem

mc_sp — ***M***ultiple-objective ***C***onstrained ***S***tatic Optimisation ***P***roblem

mu_sp — ***S***ingle-objective ***U***nconstrained ***S***tatic Optimisation ***P***roblem

mc_dp — ***M***ultiple-objective ***C***onstrained ***D***ynamic Optimisation ***P***roblem

mu_dp — ***S***ingle-objective ***U***nconstrained ***D***ynamic Optimisation ***P***roblem

Copyright © 1997-2006: Ike's Research Ltd

Table 2B: Example GENO Programs: Reference to Further Information

PROBLEM	**Location in**: 'Supplement to User Manual'
su_sp1	Example 31, p.42
su_sp2	Example 27, p.38
su_sp3	Example 40, p.51
sc_sp1	Example 7, p.9
sc_sp2	Example 12, p.15
sc_sp3	Example 14, p.17
sc_sp4	Example 6, p.8
sc_sp5	Example 8, p.10
sc_sp6	Example 5, p.7
sc_sp7	Example 13, p.16
sc_sp8	Example 11, p.13
sc_sp9	Example 22, p.32
sc_sp10	Example 14, p.17
sc_sp11	Example 41, p.52
sc_sp12	Example 42, p.53
sc_sp13	Example 43, p.54
sc_sp14	Example 44, p.55
sc_dp1	Example 35, p.46
sc_dp2	Example 15, p.18
sc_dp3	Example 21, p.28
sc_dp4	Example 16, p.19
sc_dp5	Example 36, p.47
mu_sp1	Example 19, p.23
mu_sp2	Example 1, p.2
mu_sp3	Example 2, p.4
mu_sp4	Example 3, p.5
mc_dp1a	Example 20, p.24
mc_dp1b	Example 20, p.24
mc_dp2	-

Copyright © 1997-2006: Ike's Research Ltd

Appendix III

Table 3A: A Classification of Constrained Non-linear Programming Problems [125]

CONSTRAINED NLP CLASS	VARIABLE TYPE	FUNCTION TYPE
C1	Continuous	Continuous and Differentiable
C2	Continuous	Continuous and Non-differentiable
C3	Continuous	Discontinuous
C4	Discrete	Continuous and Differentiable
C5	Discrete	Continuous and Non-differentiable
C6	Discrete	Discontinuous
C7	Mixed-integer	Continuous and Differentiable
C8	Mixed-integer	Continuous and Non-differentiable
C9	Mixed-integer	Discontinuous
C10	Continuous	Stochastic Objective and Constraints
C11	Discrete	Stochastic Objective and Constraints
C12	Mixed-integer	Stochastic Objective and Constraints
C13	Continuous	Vector Objective Function
C14	Discrete	Vector Objective Function
C15	Mixed-integer	Vector Objective Function
C16	Continuous	Dynamic Constraints
C17	Discrete	Dynamic Constraints
C18	Mixed-integer	Dynamic Constraints

[125] Global optimisation problems are very heterogeneous: they encompass the usual categories of mathematical programming models, specifically including linear models and the broad nonlinear category. Unsurprisingly, there is no universally accepted way of categorizing nonlinear programs. This classification is essentially that from Zhang (2001) to which the classes C13 through C18 have been added. Classes C16-18 were included in order to emphasize the fact that GENO is equally applicable to dynamic optimisation problems, despite the fact that the discrete-time dynamic programming problem can in principle be converted to a static program by the technique of variable stacking.

Copyright © 1997-2006: Ike's Research Ltd

Table 3B: A Comparative Capability Map of Some Existing NLP Solvers

Constrained NLP Solver	Applicability [126]																		Convexity Required	Principal Methods	Input Formats	Special Requirements
	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	C11	C12	C13	C14	C15	C16	C17	C18				
DONLOP2	✓																		No	Sequential Quadratic Programming	Fortran, AMPL	
LANCELOT	✓																		No	Sequential Quadratic Programming	SIF, AMPL	
LOQO	✓																		Yes	Infeasible primal-dual interior-point	AMPL, Matlab	
MINOS	✓																		No	Sequential linearly constrained algo	GAMS, AMPL	Modest Nonlinearity
KNITRO	✓																		No	primal-dual interior-point	AMPL	Modest Free Variables
SNOPT	✓																		No	Sequential Quadratic Programming	Fortran, GAMS, AMPL	
CONOPT	✓																		No	Feasible Path Method	GAMS	
FSQP	✓																		No	Sequential Quadratic Programming	AMPL	
HQP/OMUSES	✓																		No	Newton-type SQP, integer-point	SIF, C++	
MOSEK	✓																		Yes	Best Interior-point method	AMPL	
GENECOP	✓	✓	✓																No	Genetic Algorithm	C	
COBYLA2	✓	✓	✓																No	SLP with Gradient estimates	Fortran	Inequality Constraints Only
BARON	✓			✓			✓												No	Branch and Reduce	Baron Model	Functions are Factorable
BNB	✓			✓			✓												No	Branch and Bound	Matlab	
MINLP_BB	✓			✓			✓												No	Branch and Bound, SQP	AMPL	
SBB	✓			✓			✓												No	Branch and Bound	GAMS	
MITTLP	✓			✓			✓												Yes	Extended Cutting Plane	C	
ALPHAECP	✓			✓			✓												No	Extended Cutting Plane	Fortran, LP	Pseudo-convex functions
AUGMENTED										✓									No	Interior-point, augmented system	Standard SLP	Linear Programs Only
MSLIP										✓									No	Nested Benders Decomposition	Standard SLP	Linear Programs Only
GENO	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	No	Multi-objective Genetic Algorithm	GAUSS, C++, Mathematica	

[126] See Table 3A above for an explanation of the NLP classes C1, C2, . . . , C18.

Copyright © 1997-2006: Ike's Research Ltd