Hybrid Designer

User Manual
User Documentation
System Documentation

Written by Edward van Kuik
Edited by G Seeling-Hochmuth

January 1998
ENERGY AND DEVELOPMENT RESEARCH CENTRE
University of Cape Town

Table of contents

Note

This document is divided into three sections. The *User Manual* describes how to use the program's interface and how to interpret its output. *User Documentation* discusses how the program works and what concepts are involved. *System Documentation* focuses on the implementation of the concepts discussed in the *User Documentation*. This section also discusses confusing or difficult sections of code.

Items written in **bold** refer to menu options, or buttons on dialog boxes in the application.

USER MANUAL

What is *Hybrid Designer*?

Hybrid Designer is a program that will design hybrid energy systems best suited for a particular site. In order to make these decisions, *Hybrid Designer* needs the following information:

- weather information about the site:
  wind speeds, temperature, irradiation
- demand information about the site:
  AC & DC demand requirements
- available components and their characteristics:
  operating costs, lifetimes, installation costs, efficiency information, etc.
- general options:
  title, duration, time intervals, optimisation options, etc.

Based on this information *Hybrid Designer* uses optimisation and simulation techniques to determine and evaluate the best configuration of components as well as produce information on how to operate them. Output from *Hybrid Designer* includes:

- sizing configuration:
  which component to use, and how many of each of them
- operation strategy information:
  how to operate the diesel, at which battery SOC they should switch on, etc.

Hybrid Designer can run quite slowly, so at least a Pentium 150Mhz, with 64M RAM is required.

Structure of *Hybrid Designer*

First, create a scenario. A scenario is a file that contains all the information required by *Hybrid Designer*. Once this file has been created, it can be shared with other users of *Hybrid Designer*. The component characteristics are embedded in the scenario file as well as in the default template.

Then run the scenario. *Hybrid Designer* will determine which component configurations and which operation strategies provide good results. A list of these can be viewed and more detailed information graphed and analysed.

Creating a scenario

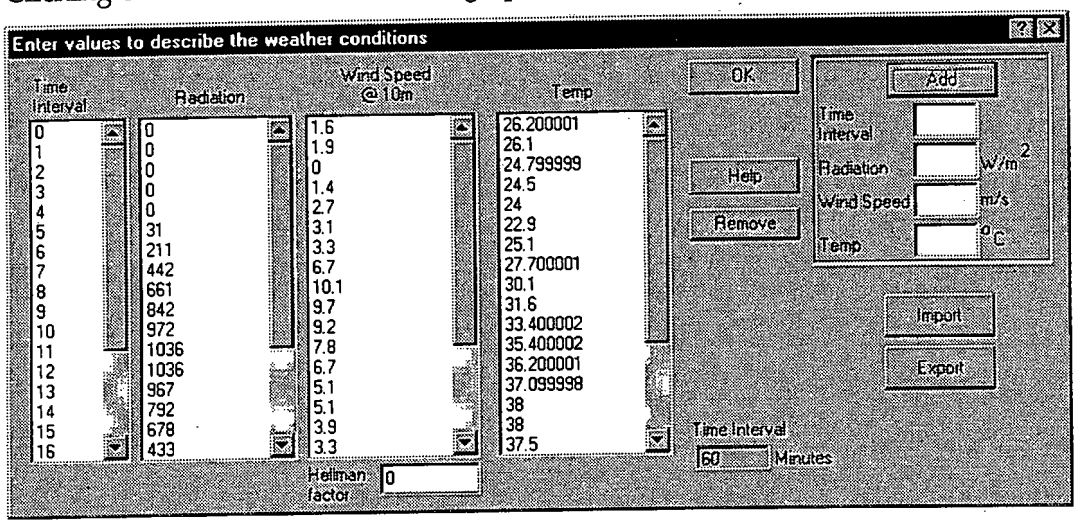
To create a scenario, click the **New** option under the **File** menu. This will make a copy of the *default template scenario*. A scenario contains all the input information required by *Hybrid Designer*.

To change the scenario information, there are several options under the **Scenario** menu option:

- Weather data
- Demand profiles (AC or DC)
- Select components
- Options

Weather data

Clicking on **Weather Data** will bring up a screen like this one:



Hybrid Designer works with *time intervals*. For each of these, the data is considered to be constant (an average) for that period. The usual time interval is 60 minutes.

For each time interval, *Radiation*, *Wind Speed* and *Temperature* are required to describe the weather conditions. The box *Time Interval No.* indicates the time interval to which one is referring. If the time interval is 60 minutes then time interval No. 0 refers to the time from 0 to 59 minutes, while No. 1 refers to the time from 60 to 119 minutes, and so on.

After placing the cursor in the *Time Interval* box, use TAB to move to the next box and Shift-TAB to move back. Once you have placed values in all four boxes, click **Add** to enter these values into the database.

You need not enter data for every time interval. If you are running the simulation for one year, and the time interval is in hours, this would become a considerable task. All values not specified will be calculated from neighbouring values, using linear interpolation. That is, if time interval 12 is 100 and time interval 14 is 120, then time interval 13 will be calculated as 110.

If the highest specified time interval is, say, 23, then all time intervals beyond these values will be repeated. Time interval 24, therefore, will be the same as time interval 0; and time interval 25 will be the same as time interval 1, and so on. Be careful here: if you have data for only one day, then ensure you enter a value for 0, and for 23, but do not go beyond these values. Entering, say, a 24th value would cause information to be out of sync.

Time interval 0 is required, or will be assumed to be zero. For convenience the present time interval step is shown in the corner of this dialog box. This value can be changed in **Options**, under **Scenario**. Use only multiples of hours, and avoid intervals smaller than 1 hour.

Values can also be imported into these fields. Clicking the **Import** button allows the user to specify a text file (.txt) with the following format:

time_interval radiation wind_speed temperature

Example:

0 700 2 21

1 705 2.4 21.1

10 710 2.3 22

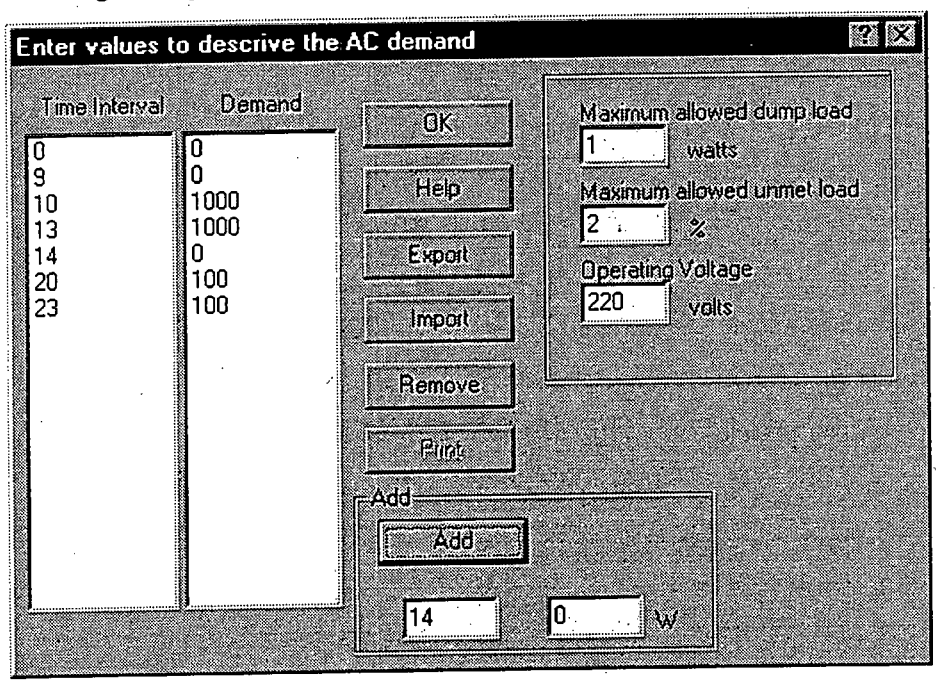
In this example, time interval 1 would have a radiation level of 705 W/m², a wind speed of 2.4 m/s and a temperature of 21.1°C. One can export the information into a file as described above. A text file like this can be created from a spreadsheet program such as Microsoft Excel©.

You are also asked to enter a Hellman factor. This factor describes the surface area and its effect on wind at various heights above ground. A Hellman factor of zero would describe a very smooth surface, such as ice. 1.33 is an average factor.

Tip: Linear interpolation takes time to compute, so for performance it is best to enter all information correctly.

Demand profiles

Clicking on **Demand Profiles**, and then on **AC** or **DC Demand Profiles** will produce the following dialog box:



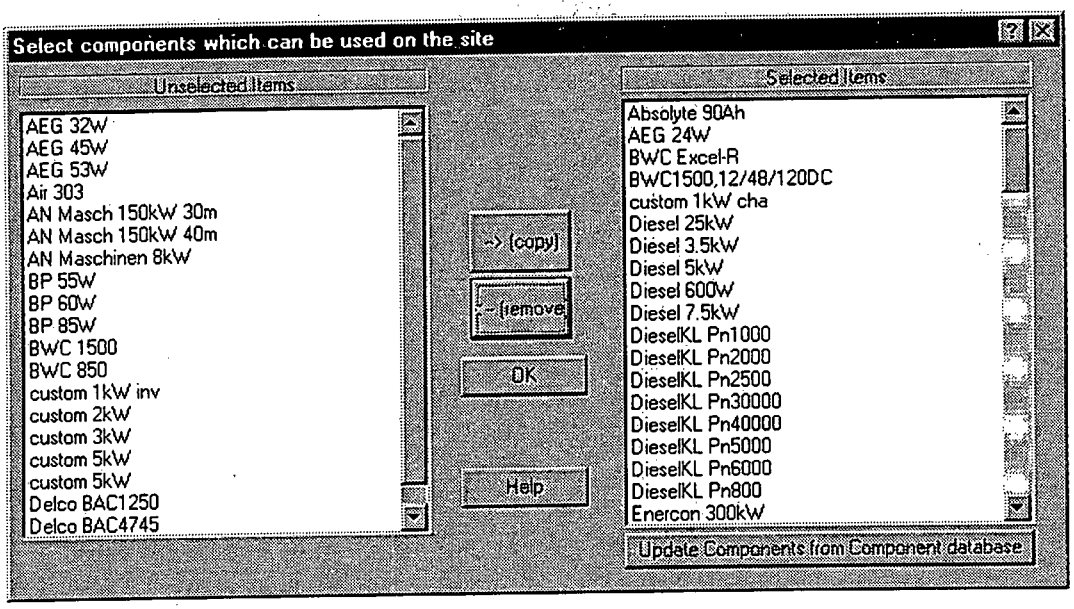
This window also enables changes to the *Maximum allowed dump load* in watts over the simulation interval. If more watts must be dumped, then this configuration is not suitable for the site and will not be placed in the list of best configurations or strategies. *Maximum allowed unmet load* is used when the applications are not critical. No penalties are charged for dumping or unmet load below the specified allowed limits.

Operating voltage is the operating voltage of the AC or DC bus.

Tip: As with weather data, the program will run faster if linear interpolation is not used. For performance, enter all the information for each time interval exactly.

Component selector

One can choose not to use all the components in the component database. Clicking on **Select Components** will bring up this window:



Simply click on the components you would like to select or unselect, and then click on the appropriate arrow.

Only *selected* components will be used in the calculation of best configurations. If you have created a new scenario, then all the components that are marked as **default components** will automatically be selected. All realistic components in the component database are marked to be included in the scenario. Ideal components, and other experimental or incomplete models are not included.

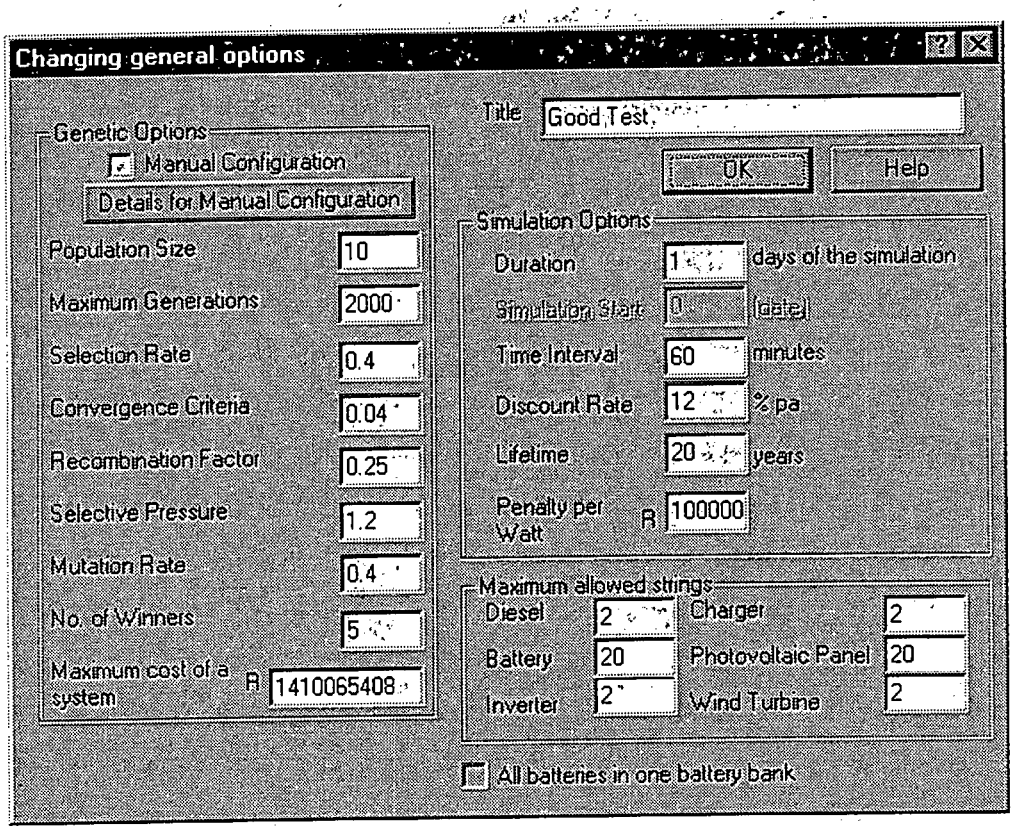
If you would like to add a component which is not available in the component database, or to change the characteristics of existing components, refer to **Component Manager**.

Be careful when choosing strange sets of components. Combinations such as batteries and diesels, with no chargers will cause the diesel to run at a very high capacity in order to try and charge the batteries. Select only chargers and inverters that match bus voltages correctly. Choose either parallel or non-parallel inverters, do not try and use them simultaneously.

Try and select as few components as possible! The fewer solutions available to the computer, the faster it will be able to find the optimal solutions.

General options

The scenario requires you to enter some general options. Clicking on **options** displays this window:



The first block gives all the options pertaining to the genetic algorithms which determine the best component configurations and operating strategies. These options will have defaulted values, but can be changed to customise the genetic algorithms.

Clicking on **Manual Configuration** disables the genetic side of things altogether. You can then specify precisely which components should be simulated. When the program runs, only one configuration will be tested and no optimisation will take place. If, for example, you need ten panels of a certain type, those ten panels should be added to the list of components on this dialog. When adding manually, remember that we are referring to strings of components. So if, say, you're working with a 24V bus, and adding 12V batteries, then there will be 2 batteries in every string.

Population Size refers to the number of configurations that will be tested in each generation.

Maximum Generations sets a limit on how long the program will continue to run. The program will also stop when the convergence criteria are met.

Selection Rate indicates how many individuals will be selected for reproduction in the genetic algorithm. Usual values is about 0.5.

Convergence Criteria defines when the program should stop. The convergence criteria are met when all the possible solutions start to look the same. This is calculated quite simply, by comparing the fitness of each solution. It is assumed that if two solutions have equal fitness, then they are equal too.

Recombination Factor explains how the offspring are created from their parents. A recombination factor of zero would result in offspring being 50% of each parent.

Selective Pressure exaggerates the difference between good and not-so-good solutions.

Mutation Rate allows each new individual to mutate slightly over every generation. This value greatly effects the performance of the genetic algorithm, experiment with different values here, if you find the genetic algorithm is performing poorly.

No. of Winners refers to how many component configurations should be listed when the program run is completed, that is, the number of *best* results listed.

Title is the scenario's title, for future reference.

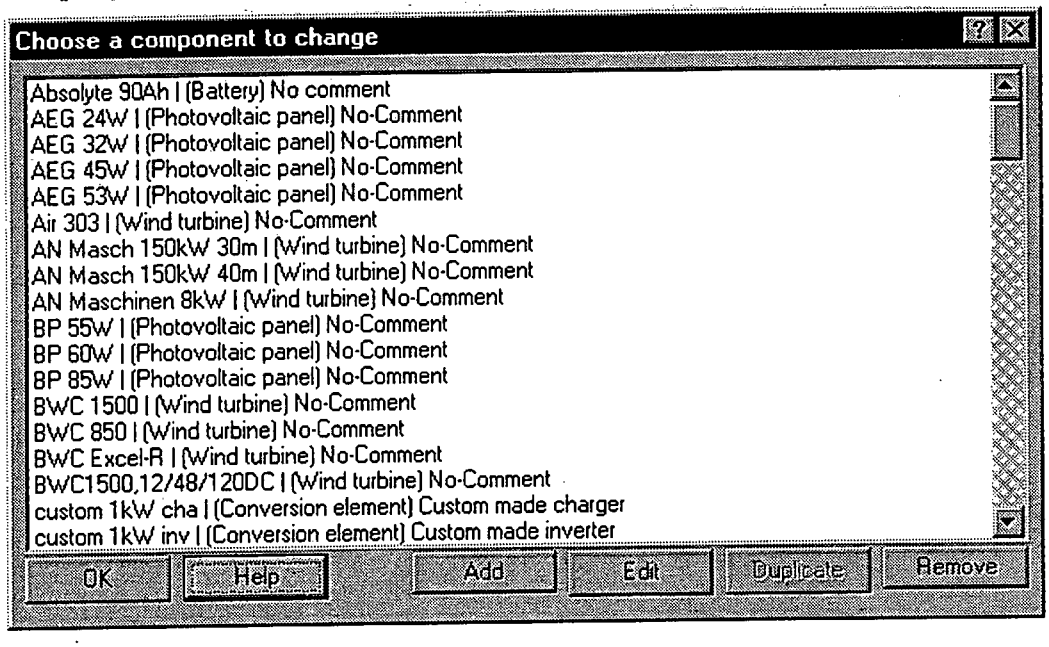
Duration refers to the number of *days* over which the simulation should run.

Lifetime indicates how far the results from the simulation should be extrapolated for evaluating life-cycle costs. The amount is expressed in *years*.

Time Interval is normally set to 60 minutes. Be careful when using alternative time intervals. Use only multiples of hours, i.e. 120 minutes, etc. Program perfomance for smaller intervals is relatively untested.

Component manager

The program includes a component database, but components may be changed or added.



Certain information is required to describe the working and costing of a component when it is part of a hybrid system. To add or change a component, click **Default** and then **Components** to enter the component database.

A list of components currently entered in the system is shown in the *Component Manager* dialog box. To change the characteristics of a particular component, select the component and then click **Edit**. This window allows you to enter a name for the component and determine the category into which it falls.

The category is important because the **component details** are very different for different components.

Efficiency graph data is also required to describe the component, and is also dependent on the type of component that is being edited.

Here follows a list of **component details** that are stored for each component type (excluding **efficiency graph data**):

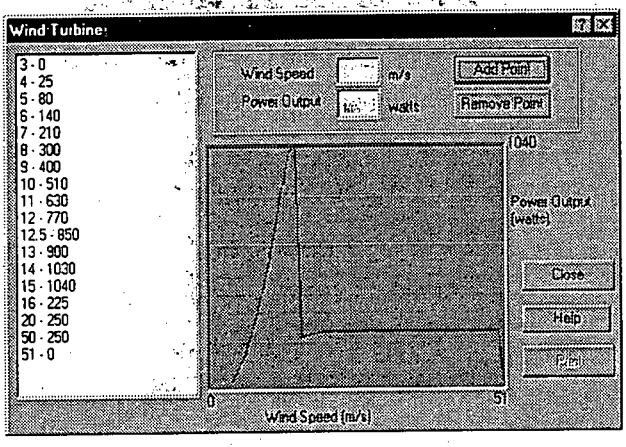
Component type	Units	Comments
	Diesel	
AC operating voltage	Volts	
Nominal power output	Watts	
Component lifetime	Hours runtime	After x runtime hours the component is replaced and costs to the amount of Component Cost are incurred.
Component cost	Default currency	The cost of the component.
Installation cost	Default currency	The cost of the installation of this component.
BOS cost	Default currency	Balance of system costs. Any other costs that are required in the installation of this component.
Fuel price (per litre)	Default currency	
Overhaul cost	Default currency	Whenever the component needs an overhaul, this amount is incurred.
Overhaul period	Hours runtime	The number of runtime hours before Overhaul Cost is incurred.
Maintenance cost	Default currency	Whenever the component needs to be repaired, it costs this much.
Maintenance period	Hours runtime	How often the diesel component is repaired.
Maintenance time	Hours	How long it takes to repair or maintain the diesel component.
Diesel tank cost	Default currency	The cost of the diesel fuel tank.
Diesel tank size	Litres	The size of the fuel tank.
	Battery	
AC operating voltage	Volts	
Nominal capacity	Amp hours	
Maximum SOC	% of nominal capacity	Maximum state of charge.
Minimum SOC	% of nominal capacity	Minimum state of charge.
Initial SOC	% of nominal capacity	Initial state of charge at beginning of simulation.
Component cost	Default currency	Cost of the battery.
Installation cost	Default currency	Installation costs.
BOS cost	Default currency	Balance of system costs.
Component lifetime	Cycles	The lifetime of the battery.
Maintenance cost	Default currency	
Maintenance period	Months	How long it takes to repair or maintain the battery.
Maintenance time	Hours	How often the battery is repaired.
Maximum charging current	Amps	
Maximum discharging current	Amps	
Wood parameters		Consists of two values which describe the efficiency of the battery. If these are set to ZERO,

		then the efficiency graph is used.
Wind turbine		
DC operating voltage	Volts	
Component lifetime	Years	
Nominal output power	Watts	
Maintenance period	Months	How long it takes to repair or maintain the wind turbine.
Maintenance cost	Default currency	
Maintenance time	Hours	How often the wind turbine is repaired.
Component cost	Default currency	
Installation cost	Default currency	
BOS cost	Default currency	
Overhaul cost	Default currency	
Overhaul period	Months	
Height	Meters	How high the wind turbine is from the surface.
Photo-voltaic panel		
DC operating voltage	Volts	
Component lifetime	Years	
Nominal output power	Watts (peak)	
Component cost	Default currency	
Installation cost	Default currency	
BOS cost	Default currency	
Maintenance period	Months	
Maintenance cost	Default currency	
Maintenance time	Hours	
Conversion elements		
DC operating voltage	Volts	
AC operating voltage	Volts	
Component lifetime	Years	
Nominal power AC	Watts	
Nominal power DC	Watts	
Component cost	Default currency	
Installation cost	Default currency	
BOS cost	Default currency	
Maintenance period	Months	
Maintenance cost	Default currency	
Maintenance time	Hours	
Parallel inverter?	Yes/no	
AC -> DC?	Yes/no	Is this a charger?
DC -> AC?	Yes/no	Is this an inverter?
DC -> DC?	Yes/no	Currently not used.

When adding or editing batteries, be sure to give them conservative initial SOC. If, for example, the simulation ends when the battery is at a lower level, then it will charge for very high replacement costs.

Efficiency graph data

The efficiency of a component is entered as a graph. Various points on the graph must be entered, and linear interpolation is used to join the dots. All values are stored as floating point numbers.



The range and domain of this graph depends on the component type:

Component type	*X-axis*	*Y-axis*	*Comment*
Diesel	Fuel (litres per hour)	Capacity factor (%)	
Battery	Efficiency (%)	Soc (%)	Discharge efficiency is 1
Wind	Output power (watts)	Wind speed (m/s)	
PV	Output current (amps)	Irradiation (w/m²)	
Conversion element	Efficiency (%)	Output power (watts)	

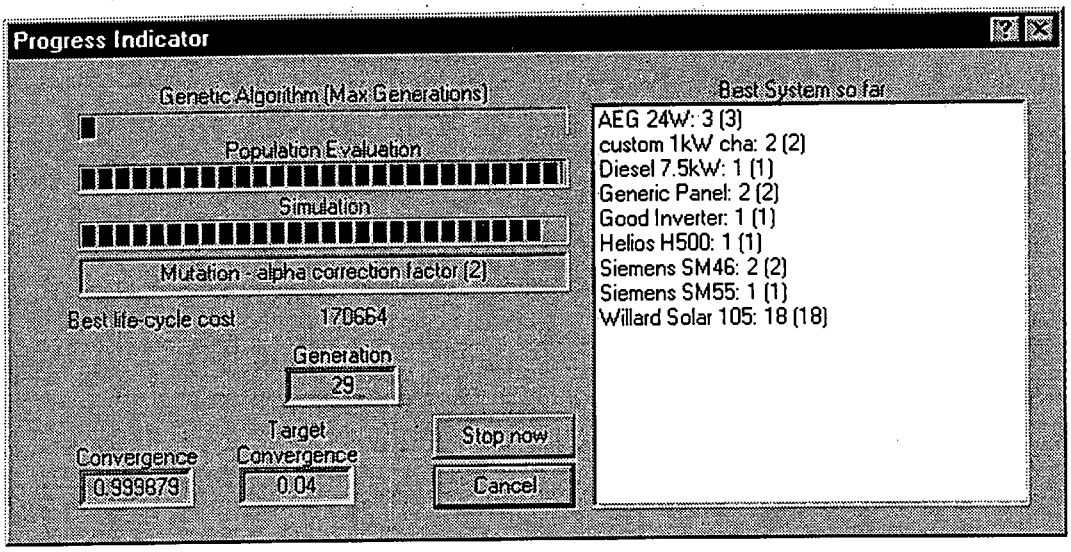
Here is an example of the **efficiency graph data** window. To add points, insert the appropriate values into the boxes provided and click **Add**. To remove values, click on the corresponding value in the box on the left, and click **Remove**. If two or more numbers are provided, then a graph is displayed and (optionally) printed. The efficiency graph should not go through the origin. Zero efficiencies cause problems with many functions in the program.

The wood parameters for batteries describe a efficiency graph for the battery, if these are set to zero, then the user-specified efficiency graph is used instead.

Running the scenario

Once the information has been entered into the scenario, it can be run.

Once the scenario starts to run, the following window will be displayed:



This window shows the progress of the genetic algorithm. When the convergence falls below the target convergence, the genetic algorithm has reached the end and the results are ready.

The output

The main program window shows the output from the run.

Click on any of the results produced, and then click on **Output** in the pull down menus. This gives you three options:

- **Sizing**

This shows you which components were chosen in the optimal system, and how many of each. You will notice that this number is a multiple of the number of components in each string. The number in brackets is the number of strings in the system.

- **Operation strategies**

If diesels were chosen for the optimal system, their operation strategies will be shown in this dialog box.

On the left is a list of components. On the right is a description of how this component should be used in the hybrid system. This information determines when a component should be switched on (as described below). If a component such as a diesel generator is switched on, its running level is calculated optimally elsewhere in the simulation program. Its running level may, however, be calculated as zero. (This would imply that the diesel is idle.)

Only diesel components have an operation strategy. Operations of other components are determined by weather and demand. The operation strategies for diesel generators are described in such a way that it would be possible to design and build a controller for the diesel. It works based on the average battery SOC and the unmet load/dumping. (This is explained in more detail later.)

In an operation strategy, five values describe the component's behaviour with regard to switching on or off.

Battery SOC indicates whether the component should be switched on if battery state of charge levels fall below certain boundaries. If the battery state of charge falls below the **LB** (Lower Bound), the component is switched on. If it rises above the **UB** (Upper Bound), the component is allowed to be switched off. Values are given in percentages of the battery's

state of charge range. 100% means the battery is at its maximum state of charge (as specified in the component characteristics) while 0% indicates that the battery is at its lowest allowed state of charge. If more than one battery is used, then an average battery state of charge is used.

Unmet load/dump operates in a similar fashion. Unmet load can be thought of as a negative dump value. If the unmet load (or dump) reaches the **LB,** the component will be switched on, and if it reaches the **UB,** the component is allowed to be switched off.

The last column, **Minimum Runtime Hours,** shows that the component cannot be switched on for a period of time shorter than this value.

- **Details**

This is where you can graph all details of the system. Some information is graphed against the 'simtime' (duration of simulation) and others against the 'lifetime' (usually 20 years).

- Output watts for each time interval for each component vs simtime
- Battery SOC for each time battery vs simtime
- Fuel costs for each diesel vs simtime
- AC & DC currents for each component vs simtime
- Temperature for each time interval vs simtime
- Wind speeds for each time interval vs simtime
- Radiation for each time interval vs simtime
- AC & DC demand for each time interval vs simtime
- AC & DC Dumping or Unmet load for each time interval vs simtime
- Replacement costs for each year in the project lifetime, for each component vs lifetime
- Maintenance costs for each year in the project lifetime, for each component vs lifetime
- Overhaul costs for each year in the project lifetime, for each component vs lifetime
- Running costs for each year in the project lifetime, for each component vs lifetime

Choose which type of information is required, select the component and click **Plot.**

There is an **Export** option that will export the selected information as a text file. The first column refers to the time interval, and the second to the values of the selected information.

The **Export All** option will export all the data to such a file, suitable to be imported as a comma delimited text file into a application such as Microsoft Excel©.

Changing the default template

Whenever a scenario is created, a copy is made from the default template. One can change the default template directly by clicking the options in the **Default Template** menu option. Notice that instead of the **Component Selector** there is a **Component Manager** to change component characteristics.

Possible future developments

The following developments are projected for future versions of *Hybrid Designer*:

- The ability to generate weather data based on skeleton weather profiles.

- More detailed analysis of output, such as the ability to graph more than one time series at a time.
- Sensitivity analysis, to show which values of the output are more important than other values.
- Increasing the speed it takes to run the program. This would allow the user to use more individuals, and run the program for more generations, giving increased certainty of optimal solutions.

USER DOCUMENTATION

Overview of *Hybrid Designer*

On creation of a new scenario, the information from the default template is copied to a new file, called a *Scenario*. When the user clicks **run**, this file is passed to the genetic algorithm. In this way *Hybrid Designer* tries to find a solution. This consists of a component configuration, (which components to use, and how many of them) and, in the case of each diesel, an operation strategy (when to allow them to be switched on or off).

The genetic algorithm creates a number of random solutions, called the population. Every solution in this random population is then evaluated by means of a *Simulation*. Each individual of the population (that is, each solution) is given a score in order to rank its suitability for a given site. This score is determined from the life-cycle costs of the system and whether it meets the demand requirements. The life-cycle costs of the system are the initial costs plus all future costs for the lifetime of the project (usually 20 years). The system which can produce the lowest life-cycle costs is the optimal solution. If a solution fails to meet the required demand, then a penalty is applied to that system in the form of a fine per unmet watt.

Genetic optimisation

A population of random solutions (within bounds) is created. Each solution in this population is evaluated by a simulation. This population is sorted according to the life-cycle costs.

The system generates offspring (new solutions) according to the **Selection Rate** (specified by the user). If the **Selection Rate** is, say, 0.6 then the number of new solutions created will be 60% of the total number in the **Population** (specified by the user). Each new solution is based on a merger between each pair in the **Population**, starting from the cheapest solution, i.e. individuals 0 and 1, 1 and 2, 2 and 3, and so on.

Each offspring (new solution) is mutated slightly according to the **Mutation Factor** (specified by the user). These offspring are evaluated, by means of a *Simulation*, to calculate their life-cycle costs. The individuals are then inserted back into the original population. Thereafter a selection process is used to reduce the population to its original size, as specified by the user.

The entire process repeats itself until all the individuals in the population are similar within the **Convergence Criteria** set by the user.

So, in summary, the events for the genetic algorithm are as follows:

- generate initial population (randomly)
- evaluate initial population (run the simulation on each individual)
- sort population (according to life-cycle costs)
- rank population (converts life-cycle costs into fitness factor)
- create new offspring (selection)
- recombine these offspring with existing pairs (according to a recombination factor)
- mutate these offspring slightly (according to mutation factor)
- evaluate offspring (by means of simulation)
- reinsert these offspring into original population

Simulation

The simulation takes a solution (which includes the sizing configuration, and operation strategies for any diesels that might be in the sizing configuration) and simulates that system according to the weather and demand requirements.

The simulation runs in two states, *quick* and *in-depth*. In *quick* mode, it records only the life-cycle costs. This is the only information the genetic algorithm requires to continue optimising the solutions. On the final pass, the genetic algorithm will run the simulation in *in-depth* mode, which will record all the intermediate data and costing data for the life-cycle to the solution file (ending in .sol). This data file contains data that is viewed and analysed by the interface.

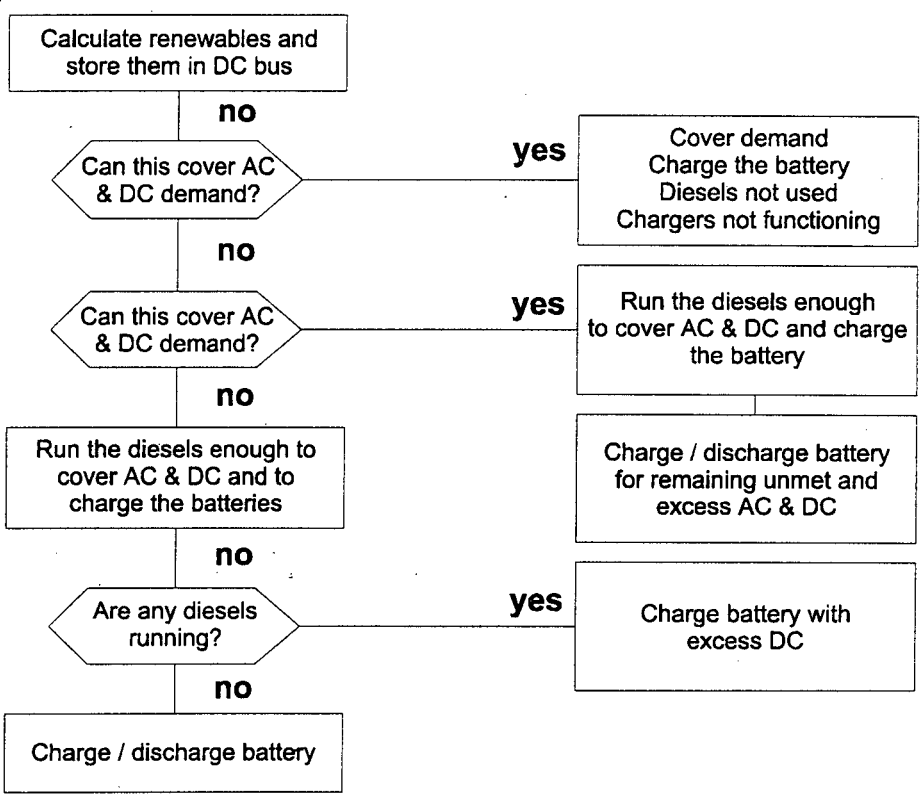
Simulation also outputs a status progress indicator to inform the user of progress. The system configuration shown in the progress window is the best (cheapest) system configuration found so far.

The simulation works as follows:

Initialisation

- Initialise all batteries with their specified initial state of charge.
- Calculate how many components in each string.

For every hour



A diesel component is not necessarily active. According to the operation strategy defined by the genetic algorithm, a diesel may have been declared inactive. However, this activation is based on the battery SOC, the unmet load/dump, and the remaining runtime of the diesel. If the battery, or unmet load/dump falls below a certain point, then the diesel is activated, and remains active for a certain number of runtime hours. However, the battery SOC and unmet load/dump may have changed during the time interval, which could mean that some diesels should have been active after all. In this case the entire time interval is re-calculated, with the appropriate diesels active (or inactive as the case may be).

The table below explains how the elements are used. All batteries are treated as one battery bank, and only one type of battery is allowed by the simulation for each scenario. Since any number of different types of diesel generators, inverters and chargers are allowed, the system automatically chooses the best combination of these based on those components' nominal values which most closely match the desired output. Diesels are usually run as closely as possible to their maximum capacity unless the batteries are full or charging at maximum capacity. The operation strategy allows for the diesels to be turned off, and not all the diesels, therefore, are necessarily active. After the simulation is complete, there is enough information to calculate information such as the running costs and life-cycle costs.

The end of the simulation

On completion of the simulation, running and life-cycle costs are calculated. Running costs are based on how much fuel was used by the diesel, while life-cycle costs are calculated in different ways. If the simulation is not run in *in-depth* mode, then *quick* costing is used. *Quick* costing quickly determines the approximate life-cycle costs of a particular solution, while *in-depth* costing gives a more accurate in-depth calculation and stores all intermediate information for scrutiny by the user. *In-depth* costing is only performed on the final set of solutions.

Quick costing

The following values are added together to determine the life-cycle costs.

- **Diesel generators**

$(D_{cost} / D_{lc}) * (D_{runtime} - D_{lt})$

D_{cost} = Cost of diesel generator

D_{lt} = Lifetime of the diesel generator (in runtime hours)

$D_{runtime}$ = No. of hours the diesel ran

$(D_{maint} / D_{mt}) * (D_{runtime})$

D_{maint} = Maintenance costs for diesel generator

D_{mt} = Time between maintenance

$D_{runtime}$ = No. of hours the diesel ran

$(D_{overhaul} / D_{ot}) * (D_{runtime})$

$D_{overhaul}$ = Overhaul costs for diesel generator

D_{mt} = Time between overhauls

$D_{runtime}$ = No. of hours the diesel ran

- **Batteries**

$B_{cost} / B_l * (B_{cycles}-B_l)$

B_{cost} = Cost of the battery

B_l = Lifetime of the battery in cycles

B_{cycles} = Total cycles of the battery in (20 years)

- **Conversion elements (chargers & inverters)**

$CE_{cost}/CE_l * LC_{duration}$

CE_{cost} = Cost of conversion element

CE_l = Lifetime of the conversation element

$LC_{duration}$ = Duration of life-cycle (20 years)

$CE_{maint}/CE_{mt} * LC_{duration}$

CE_{maint} = Maintenance cost of conversion element

CE_{mt} = Time between maintenance

$LC_{duration}$ = Duration of life-cycle (20 years)

- **PVs**

$PV_{cost}/PV_{l} * (LC_{duration} - PV_{l})$

PV_{cost} = Cost of photovoltaic panel

PV_{l} = Lifetime of the photovoltaic panel

$LC_{duration}$ = Duration of life-cycle (20 years)

$PV_{maint}/PV_{mt} * LC_{duration}$

PV_{maint} = Maintenance costs of photovoltaic panel

PV_{mt} = Time between maintenance

$LC_{duration}$ = Duration of life-cycle (20 years)

- **Wind turbines**

$W_{cost}/W_{l} * (LC_{duration} - W_{l})$

W_{cost} = Cost of wind turbine

W_{l} = Lifetime of the wind turbine

$LC_{duration}$ = Duration of the life-cycle (20 years)

$W_{maint}/W_{mt} * LC_{duration}$

W_{maint} = Maintenance costs of wind turbine

W_{mt} = Time between maintenance

$W_{runtime}$ = Runtime hours of the wind turbine

$W_{overhaul}/W_{ot} * LC_{duration}$

$W_{overhaul}$ = Overhaul costs of the wind turbine

W_{mt} = Time between overhauls

$LC_{duration}$ = Duration of the life-cycle (20 years)

All the costs added together so far must be discounted according to the discount factor.

The initial costs are then added to the total:

- Component cost (incl. tank cost, if necessary)
- Installation cost
- Balance of system costs

Finally we add the penalty costs. Penalty costs are defined by the user in the **Simulation Options** section. A simple method imposes a fine for every unmet watt in the system. Note that a certain amount of unmet demand is allowed, as defined by the demand section. This allowed unmet demand is not considered part of the penalty.

This quick costing will not be the same as the in-depth costing, although it should be very similar. The difference comes in when a component is, say, replaced once every 10 years.

This would mean two replacements over 20 years, one in 10 years time and another in 20 years time. The quick costing would simply distribute the costs of two replacements evenly over 20 years. Other differences can occur when one method replaces a component, say twice, and the quick costing will replace the component 2.3 times, say.

In-depth costing

In-depth costing differs in that all data is saved to the output file and more accurate formulas are used.

Tables are used for each lifetime interval, and costs are incurred according to need, in contrast to the *average* replacement or maintenance costs given in quick costing. If, for example, maintenance amounts to R200 every 10 months, then in-depth costing gives a cost of R200 every 10 months in contrast to quick costing which would average out to about 3c per hour (if the Time Interval is hourly).

All the costing information can be viewed from the interface's **Details** section.

SYSTEM DOCUMENTATION

Introduction

This document is written for future programmers and coders who may want to change or review areas of code in Hybrid Designer. This document is divided into three sections:

- An overview of how the program works
- An indepth discussion of each module of the program, where difficult or confusing areas are also explained. (This section should be read with the code at hand)

Hybrid Designer is created with Borland C++ 5.01 using the OWL and STL libraries. There is a known problem with using STL and OWL together as they both have objects called "string". To overcome this we have renamed string to OgadString (Ogad because that is what the program used to be called). Have a closer look at stl.h for more comments on this.

We initially tried to separate the interface part from the rest of the program to facilitate porting the program to other platforms - or possibly to allow it to be used remotely over the internet through CGI, Java or something like that.

Although we have tried to be object-oriented, but many areas of code are basically procedural, for speed purposes.

As containers, we have almost always used deques (double-ended queues).

Description of data types

Data types are where information is stored. Here follows all the data types used in the program (excluding temporary ones used in the interface) In brackets is the name of the file that holds the source code to run the modules. For example, OPS means that the files are OPS.H (the header file) and OPS.CPP (the actual code)

Component configuration and operation strategy (ops, compscen)

Originally we had the operation strategies and component configurations as completely separate entities, but now, we have compscen as a base class for ops. This may seem odd, but that's because it is, and was the easiest solution at the time. The individual consists of all information to be optimised. It consists of a component scenario which is a list of all the components and how many of each of them should be used in the site configuration. An example would be:

Component Type	*How many strings of each of them we shall use*
Phillipso Diesel 500W	2
Bluebell Diesel 100W	1
Silly Farm Chargers	5
Samsungy Photovoltaic Panel 50W	18
Funny Farm PV Model 135A	0
Simba Battery 1Ah	17
Happy Farm Inverters Model 567A	1

Also, as part of the individual, is the operation strategy for each diesel component in the individual. In the above example, there are only three diesels. We have two diesel *types*, and one type has two *instances*. (not referring to OO instances)

Each diesel holds information on when it is allowed to operate. This is based on two factors, the average SOC of the system, and the Unmet Load or Dumping of the system. If the average SOC falls below a certain point then the diesel will be activated. When this SOC rises about another point the diesel will no longer be allowed to operate.

These two values, relating to the SOC, are called the Upper and Lower SOC values. (Usoc, Lsoc) For these calculations, SOCs are stored as a percentage of the maximum available Ah from all the batteries.

Similarly, when the unmet load or dumping (called the external value, positive or negative) falls below a certain value then the diesels ALSO (logical OR) allowed to operate, etc. These two values are called the Upper and Lower External values. (Uext, Lext) These values are expressed as a percentage of the total demand requirements.

Additionally, another value is needed to describe the operation for the diesel, and this is the minimum runtime values (Mrt) – that is, if a diesel is switched on, it stays on for a certain number of hours.

So all the values for the different diesels are stored in a table. Another example:

Diesel Number	*Instance Number*	*Usoc*	*Lsoc*	*Uext*	*Lext*	*Mrt*
0	0	70.43	100	0.02	-10.3	5
0	1	67.43	95.34	-5.32	1.2	3
1	0	70.12	94.32	0.02	0	0

In this example, the first diesel (diesel 0) has two instances (instance 0 and instance 1). This diesel will be active when the average SOC falls below 70.43% of the battery capacity and switch off again only when it rises to 100%. It will ALSO (logical OR) be on if the unmet load is more than 0.02% of demand and switch off again if the dumping is more than 10.3% of demand. It will remain on until the remaining runtime hours have expired. (logical AND)

The components (compon)

The components fall into different fixed catagories. Each one has separate information. Consult the User Manual for a list of information stored for each component.

The COMPON modules also contains certain functions to help the simulation find out information about the components these functions include:

- Max possible (dis)charging for battery:
  This function looks at the battery and calculates how many amps it can accept or produce depending on its maximum (as defined by the user) and the possible (depending on its SOC)
- Self discharging for the battery at 16.6% in 100 days:
  This function is used by the *New Battery SOC* function when the new battery SOC is calculated for the beginning of the next time interval.
- Calculation of strings to match up voltages
  This function will calculate how many components should be in a string. i.e. two 12V batteries on a 24V bus. Some components cannot be in series, and so will be automatically set to 1.
- PV output according to efficiency graph data
  The efficiency graph of PV's is for ambient temperature. This function will return the true amps from the PV panel according to the weather data.
- Wind output according to efficiency graph data
  The efficiency graph for wind turbines gives the output in watts, and this function will convert to the appropriate amount of amps in the system. Output is correct for different wind height according to some Hellman formula.
- Power Converters output according to efficiency graph data
  For inverters and chargers this function, will determine (according to the efficiency graph) how many DC amps will produce how many AC amps, and the other way round.
- Diesel fuel costs according to efficiency graph data
  This function looks at how much fuel is being used to run the diesel at a certain capacity factor, and calculates the cost thereof.
- New battery SOC after charging or discharging
  Calculated the new battery SOC according to how many amps are arriving or leaving to or from the battery. It also reduces the battery SOC according to a self discharge formula (16.6% every 100 days)

Efficiency graph data (effgraph)

All values are stored as floating point numbers. All numbers should be positive.

Component type	*X-axis*	*Y-axis*	*Comment*
Diesel	Fuel (litres per hour)	Capacity factor (%)	
Battery	Efficiency (%)	SOC (%)	Discharge efficiency is 1
Wind	Output power (watts)	Wind speed (m/s)	
PV	Output current (amps)	Irradiation (W/m²)	
Conversion element	Efficiency (%)	Output power (watts)	

The efficiency graph will automatically interpolate data that is not provided. Linear interpolation is used. Also "inverse" functions are supported. By "inverse", I mean to query the component in the reverse direction. When, for example, the inverter is queried as to how much DC it needs for a specific amount of AC, the certain efficiency is required. A binary search is done on the graph to find this value.

Weather (weather)

For each Time Interval, **Radiation, Wind Speed** and **Temperature** is required to describe the weather conditions. If information is not specified for any time interval then the values will be interpolated.

Demand (demandd)

The demand stores data in watts for each time interval. Missing information is interpolated. Also stored in this module are the operating voltage, allowed unmet load and dumping values. Stored in voltage, percentage and watts a respectively.

Simulation options (soptions)

This simple module just stores some general simulation options:

- Duration of the simulation (in days, e.g. 365 days)
- The start of the simulation (for seasonal changes, .e.g. 1 Jan), unused
- Time Interval (in minutes, e.g. 60 minutes), only really tested for 60 mins
- Title of the simulation (for record purposes)
- Discount Factor (or interest rate, in %)
- Lifetime (for life cycle costs, e.g. 20 years)

The genetic options (goptions)

This just stores some general genetic options:

- Population size
- Maximum number of generations
- Convergence factor
- Selection rate
- Selective pressure
- Recombination factor
- Mutation rate
- Maximum number of components allowed for Diesels, Inverters, Chargers, Batteries, Wind Turbines and PV panels.

Trace data (ttrace)

For every time interval:

- Average battery SOC
- Running costs (fuel costs)
- External (unmet load or dumping) for AC and DC.
- For every component
- AC or DC (or both) currents
- Watts
- Remaining runtime
- Battery SOC (if component is a battery)

- EfficiencyLifetime section:
- For every lifetime interval (constant at the moment)
- Replacement cost
- Maintenance cost
- Overhaul cost

Scenario file (scenfile)

This file consists of all the input data required. This file can be saved to a file, and loaded, etc. It always has the extention .SCN. It has the following structure:

- List of components, and their characteristics
- Weather Data (see WEATHER)
- Demand Data for AC (see DEMANDD)
- Demand Data for DC (see DEMANDD)
- Global Options (see GOPTIONS)
- Simulation Options (see SOPTIONS)
- Filename to use (just a string)
- Duration of the simulation (in days)
- Start of simulation (for seasonal changes)
- Time Interval (in minutes)
- Discount Factor (%)

Output file (outfile)

All the output information is ultimately stored here. At the end of the RUN then the OUTFILE is stored to a file on disc, which has the extention .SOL. The output file contains the following:

- The entire final population (OPS and COMPSCEN)
- Trace data for the final population (see TTRACE)

Interface

Main (main)

The main functions of the main module are:

- Initialise the window system

Ogad (ogad)

This module controls the general information window that loads when the Hybrid Designer loads. This operates:

- The pull-down menus and on-screen buttons.
- Opening, Saving of files
- Creating new files.
- Loading the winning population into the window, if there exists output data associated with the currently loaded scenario.

Maintenance of component database (maintain)
Maintains the list of components. Allows creation of new components, and deletion, etc.

Maintenance of components (maincomp)
Intermediate window that calls either the Component Specific window (COMSPECW) or the Efficiency graph window (EFFWIN), for the appropriate component.

Efficiency graph window (effwin)
Manages efficiency graph data, enters information into EFFGRAPH, and plots this information using PLOTTER.

Plotting tool (plotter)
Draws the graph onto the screen and when the window is moved, redraws the graph.

Progress indicator (progress, PROGWIN)
Draws a box on the screen while the genetic, and simulation code is operating to show the user how advanced the calculations are. Also tries to extimate how much time is needed to complete the operation. Also supports the cancel and quit functions, which cancels everything, or outputs the most recently aquired data. Quit will complete the cycle at the end of the current generation, allowing the user to view the data, while cancel will cancel the operation and destroy the data.

Scenario window (scenwin)
Allows the user to enter data about the scenario into the global and simulation options modules (GOPTIONS and SOPTIONS).

Component selection window (selewin)
Allows the user to choose which component should be used in the scenario and which should not. Component names should be unique.

Weather window (weatherw)
Allows the user to enter data about the weather into the weather module. (WEATHER)

Demand window (DEMANDW)
Allows the user to enter data about the demand into the demand module. (DEMAND)

Component specific window (comspecw)
Loads the window appropriate for the current component the use is editing. It then allows the user to enter data into the COMPON module.

Strategy window (stratwin)
Outputs the operation strategy (OPS) to the window. Offers possibilities to print it too. (QPRINT)

Sizing window (sizing)
Outputs the component scenario (COMPSCEN) to the window. Offers possibilities to print it too. (QPRINT)

Calls PLOTTER, QPRINT, TTRACE, SCENFILE, OUTFILE.

Genetic code

The genetic code consists of the following areas:

- Initialisation
- For each generation: (or until QUIT, see PROGRESS)
  - Sorting
  - Ranking
  - Selection
  - Recombination.
  - Mutation
  - Evaluation by simulation (SIMULATE).
  - Reinsertion
  - Output to TTRACE and OUTFILE

Initialisation

Currently each component type has a random number of instances not bigger than the "max components" setting. In optimisation this will be changed to allow the restriction of different type of component seperately.

For every diesel in the component types an operation strategy (OPS) is created in the following way:

- soc = Random(100)
- Usoc = Random(100)
- Uext = Random(200) - 100
- Lext = Random(200) - 100
- Mrt = Random (100)

Each individual is now evaluated.

Loop

The loop is continued for every generation.

Sorting

Sorting is done in a very simple way at the moment. Using the bubble sorting technique. This is very simple, but since sorting not an issue, speedwise, we left it up to this robust little friend.

```
NotOk = true;
while (NotOk) {
        NotOk = false;
        for every individual X (except the last one) {
                for every individual Y (between 1 and X) {
                if (X < Y) then swap X with Y;
                NotOk = true;
```

```
            }
        }
}
```

Ranking

For every individual in the population the fitness is, where SP is selective pressure (user option), POS is the number of the individual, Nind is the number of individuals.

Selection

The top 10% of individuals are selected and thereafter Roulette Wheel Selection is used on all the individuals, including the top 10%.

NMat = Round(Number of Individuals * Selection Rate)

sumfit = sum of all the finesses of all the individuals.

for every individual {

it's probability = it's fitness / sumfit

it's frequency = accumulated probability (from the individuals so far)

}

Add top 10% of individuals to the selected population.

Set X to be first individual in the list.

for every number from 0 to NMat {

while(the frequency of X < a random number AND X is still in the list) {

X is the next individual in the list.

}

Add X to the selected population.

}

Recombination

First the component scenario is recombined.

Then the operation strategy is adjusted in size

Then the operation strategy is recombined.

All even numbered individuals mate with odd numbered individuals

p_even = Random number between (-RecFac,1+2RecFac) //check code

p_odd = Random(-RecFac,1+2RecFac)

new_individual_even = P1+p_even(P2-P1)

old_inidividual_off = P2+p_odd(P1-P2)

where p_even, p_odd are temporary variables.

RecFac is the recombination factor

P1 is any value (locus) in even numbered individual

P2 is any value (locus) in odd numbered individual

The operation strategy has variable size, so only the first X entries are recombined, where X is the size of the smaller individual. The remaining entries in the larger individual will remain unchanged.

Mutation

When the genetic code section begins *Delta Recombination* is calculated. It consists of 20 numbers where number *p* is = .

For every offspring *p* the *Alpha* is calculated. It is a random selection the *Delta Recombination* numbers.

For each number *p* in the *Delta Recombination* numbers

Value += tmp * *Alpha*

where tmp is randomly either -1 or 1.

All the values in the ops are mutated this way.

Ranges are checked again.

Simulation

The simulation keeps tables of information on each component regarding:

- AC current contribution (can be positive or negative)
- DC current contribution (can be positive or negative)
- Fuel costs (if the component is a diesel)
- Remaining runtime (only used for diesels)
- Battery SOC (only used for batteries)

For each time interval, each component is considered and information entered into the AC and DC current contribution vectors.

Keep in mind then we may not be dealing with components alone, components are grouped together according to their voltages. i.e. Two 12V PVs could be used as one 24V component on a 24V DC bus.

Voltages remain constant at all times!

The best way to explain how this works is to illustrate with some C++ code. It's not too difficult. The functions are explained below.

// Firstly we get the current contribution for renewable components, which are all DC anyway. DoRenewables() considers all Wind Turbines (DC) and PV panels (DC) and sets their current contribution in the tables.

Diesels could or could not be active, according to their operation strategy. If they are active, then they can be set to any running level, including zero, which does not mean they are switched off, simply that they could be idle. I.e. they would still consume small amounts of fuel.

If any component is found that it should have been active for the last time interval, then it is activated and the time interval is recalculated.

Then the information is then stored in the trace data (TTRACE) for future use.

Then the costing is calculated, see *Costing* below.

Notes on various functions

Activation of component strings

This section only refers to the diesels.

For each diesel, if they average battery SOC is smaller or equal than the Lsoc OR the external is smaller than the Lext of the operation strategy of the diesel, then the diesel, if ON, remains on, but cannot switch off.

Else if the diesel is not on, then it is switched on and the remaining runtime is set the operation strategy's runtime. Since something has now changes, the REDO flag is set to ON, and the time interval will be repeated.

If the SOC is smaller or equal the Usoc OR the external is greater than the Uext, then if the diesel is ON, keep it on, and if it is off, keep it off.

Else simply decrease the remaining runtime, by one time interval.

Here is a table showing which areas of the battery SOC affect the the diesels. There is another identical table for externals.

Renewables (dorenewables)

Renewables are simple. They are set to the value as defined by the weather, and their efficiency graphs.

Inverters

(getinverterinv, getinverter, doinverter)

Inverter inverses (Getinverterinv)

This function takes a AC current value, and tries to calculate how much DC is required to produce the desired AC current value. All inverters are considered, and every possible combination of inverters is considered. This may take some time, if there are many inverters in the component scenario. The combination that produces the most efficient conversion is selected, and this combination is stored in temporary variables to be used later. It is important to remember that this function does not affect the current tables, it only queries the inverters.

This function calls the GetInverterInv() of the EFFGRAPH module.

Inverters (getinverter)

This function takes a DC current value, and calculates how much AC can be produces from this. All inverters are considered, every combination of inverters are considered. The best combination is the combination that converts the current most efficiently. The combination is stored in the same temporary variables to be used later again. This function does not affect the current tables, it only queries the inverters.

DOinverters

If no data is sent to this function, i.e. DoInverter(), then it simply uses the temporary data that was last calculated in either GetInverterInv or GetInverter. It puts this information into the current tables.

Else if you call DoInverter(X), then GetInverter(X) is called, before DoInverter().

Chargers
(GetChargerinv, getcharger, docharger)

Chargers work in exactly the same way as inverters. Reread the inverters section, but read AC for DC, and DC for AC.

Batteries
(CHARGE, discharge, maxposcharge, maxposdischarge, setnewbatsoc)

Charge
We assume that the batteries are stored in ascending order according to their nominal capacity. The first battery is charged at it's maximum according to its characteristics set by the user. Then if there is any current left, the second battery is charged, as so on, until no more current is left over, or all batteries are charging at maximum capacity.

Discharge
Similar to CHARGE, the batteries are assumed to be in ascending order, so the first battery is discharged, as much as possible according to its characteristics. After that, the second battery and so on.

Maxposcharge
This calculates how much current can go to the batteries. It uses the component characteristics to calculate this. See COMPON.

Maxposdischarge
Calculates how much current can come from the batteries. It uses the component characteristics to calculate this. See COMPON.

Setnewbatsoc
Simply calculates the new battery SOC after the amount of charging or discharging has been established. See COMPON.

Diesels (getdiesel, dodiesel)

Getdiesel
This function just calculates the maximum amount of current the diesels can produce. (Only active diesels of course)

DOdiesel
This function takes two values; and upper and lower limit. The diesels are allowed to operate at any level between these two levels. The DoDiesel function will try to use the most efficient level, which will probably be the maximum value. (but anyway)

DoDiesel goes through every possible combination of diesels to find which diesels, when running at maximum capacity will produce the closest amount above the lower limit. It then assumes that diesels are in ascending order, and distributes the currents required in order. These values are then added to the AC current vectors.

Quick costing
Quick costing is separate from in-depth costing, as just a quick estimate of costs is calculated to evaluate the individual. After the genetic code has completed in-depth costing is done on the final generation to calculate more accurate costing information.

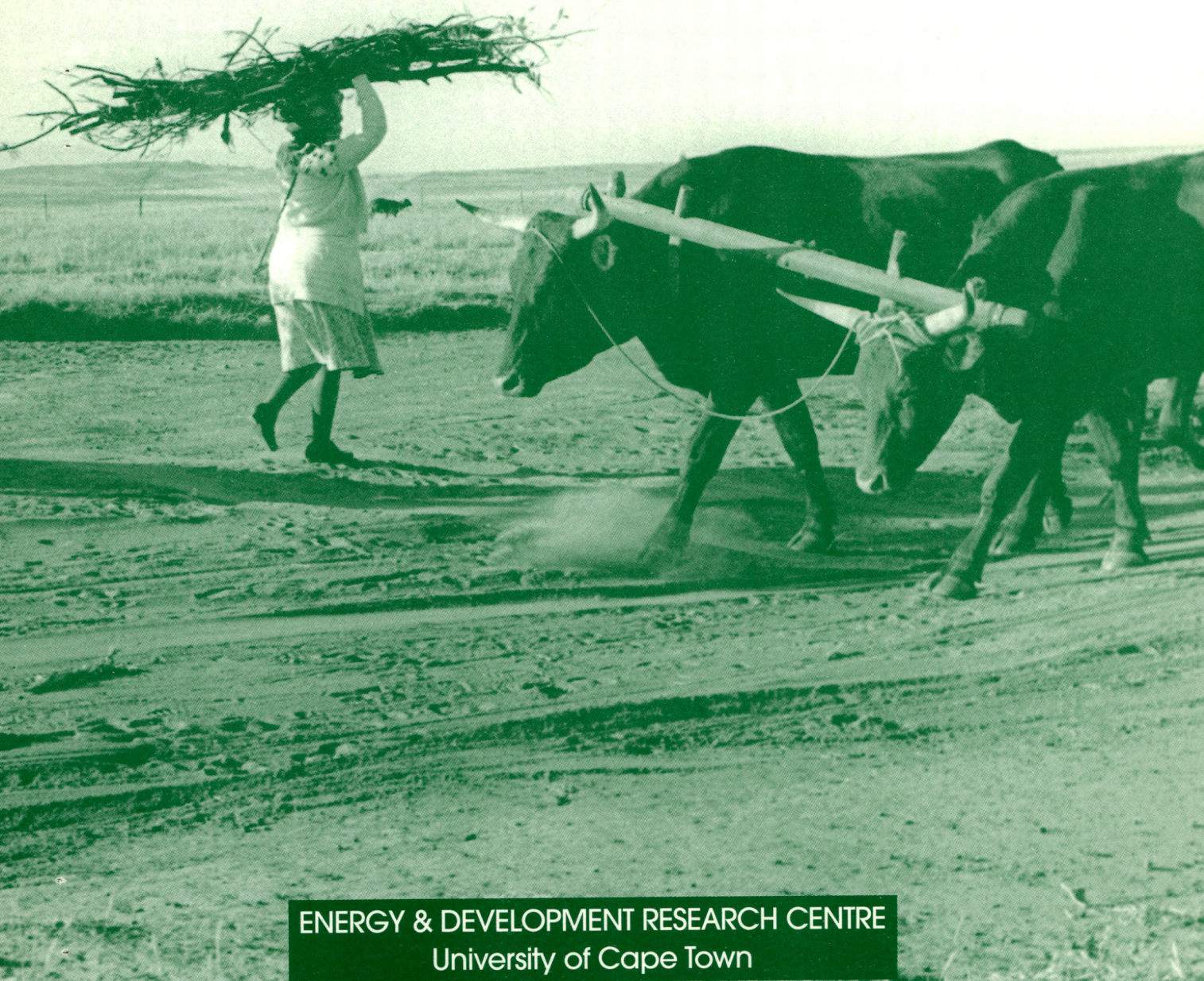
During the actual simulation runtime, actually only fuel costs, runtime hours of diesel, wind and cycles of battery usage are calculated. Fuel costs are calculated directly from the efficiency graph data entered by the user in COMPON.

Runtime costs, such as fuel costs, are carried into the costing section and extrapolated for the entire lifetime.

EDRC REPORT SERIES

Hybrid Designer: User manual, User documentation, System Documentation

Written by Edward van Kuik
Edited by G Seeling- Hochmuth



ENERGY & DEVELOPMENT RESEARCH CENTRE
University of Cape Town