SimSEE User Manuals

Volume 6 - OddFace

Optimizador Distribuido De Funciones de Alto Costo de Evaluación.
Distributed Optimizer of High Cost Evaluation Functions..

Ing. Ruben Chaer
April 2019 - Montevideo - Uruguay

Sumario

1. Introduction.

This document presents the computer application for Distributed Optimization of High Cost Evaluation Functions (OddFace). OddFace was developed in the framework of the ANII FSE 18-2009 project "SimSEE platform improvements" which ended in September 2012.

In a nutshell, OddFace is a tool to solve Optimization Problems, consisting of the search for the parameter set that minimizes the value of an Objective Function or Cost Function. The search is limited to the values of the parameters within a possible set called in the mathematical jargon as a Feasible Region or Problem Domain.

OddFace specializes in the search for the minimum in situations where the Cost Function is of high evaluation cost, understanding as such a function that requires considerable calculation time to evaluate its value at each point of the search space (given game value of parameters). To do this, OddFace implements a communication layer between possible Explorer Robots in order to allow the distribution of the calculation between several machines (or calculation nodes) keeping the results in a Problem Database (BDP) to which all the Explorers can access, as shown in the scheme in Fig.1. Each Robot Explorer consults the BDP to know what the others have managed to explore and, based on that information, it is proposed to evaluate a new point (value of the parameter set) of the search space. Since a Genetic Programming algorithm was implemented for the main exploration strategy, a value of the parameter set (or point of the search space) is called *Individual* and the Individuals are classified in the BDP from most to least *successful* in increasing order of the estimate resulting from the cost function. The most successful individuals are those with the lowest estimated value of the cost function. Robots work in an infinite loop until they are ordered to finish their tasks by repeatedly executing the steps shown in Fig.2

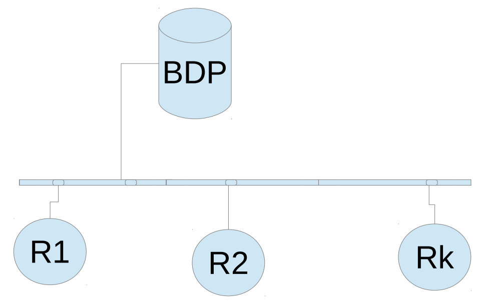
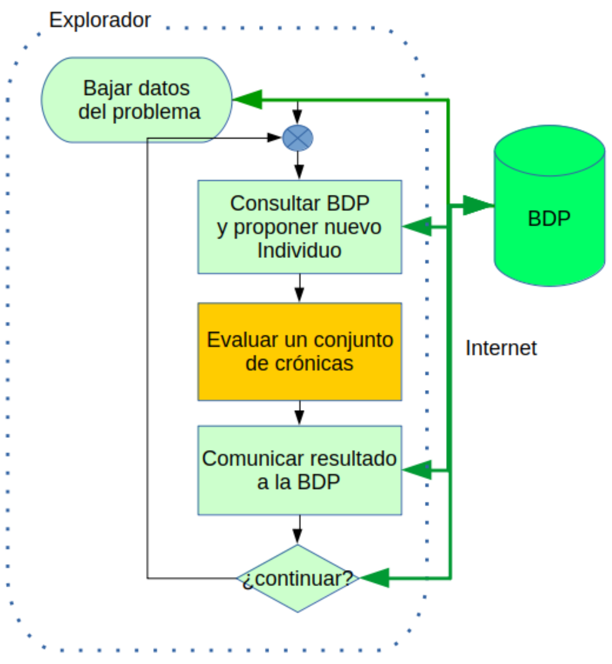


Fig. 1 Servidor de Base de Datos y Robots.

In the application area for which OddFace was designed, the evaluation of the Cost Function generally involves the simulation of a dynamic system (a system where the actions of the present impact on the future) operated with certain operating rules and subject to uncertainty during a time horizon. To fix ideas, if the problem is the optimization of investments in electricity generation, the evaluation of the Cost Function will mean simulations of steps of time (eg weekly) over a horizon of tens of years (eg 30 years) considering different realizations of the stochastic processes involved (for example, the tributary flows to hydroelectric power plants, the production of wind,


Explorador
Bajar datos del problema
Consultar BDP y proponer nuevo Individuo
Evaluar un conjunto de crónicas
Comunicar resultado a la BDP
¿continuar?
BDP
Internet
Fig. 2: Loop of an explorer.

solar, temperature, the price of a barrel of oil, etc.).

In general, the uncertainties are relevant and to obtain significant results, the simulations must be performed on a set of chronicles of the order of 100 for expected value estimates and of the order of 1000 or more when it comes to measuring the probability of specific events. The Cost Function is the expected value of the future cost of operating the system efficiently in the assembly (set) of simulated chronicles. The greater the number of simulated chronicles, the greater the accuracy with which the value of the cost function will be estimated.

OddFace makes use of this evaluation in sets of chronicles to carry out a search based first on a reduced set, that then, on the most successful Individuals, is improved by subjecting them to new sets of chronicles.

As already mentioned, OddFace was created for its application to the optimization of the Investment Plan in Generation, but since it was an efficient tool for the optimization of complex functions, Robots have subsequently been developed to optimize the Annual Maintenance Plan, the Optimization of LNG Shipment Agencies based on developments carried out in projects of the Sectorial Energy Fund of ANII that gave rise to the applications OddFace_PIG, OddFace_PAM and OddFace_OptimA respectively. Another application of OddFace is the one developed in ADME for the calibration of wind farm model parameters and for the training of neural networks for generating generation forecasts from climate forecasts.

The main applications OddFace_PIG, OddFace_PAM and OddFace_OptimA are based on the variation of parameters on a SimSEE Room (a Room is a representation of an electrical generation system to simulate) and the SimSEE simulation of the system (represented by the corresponding Room) allows calculation of the Cost of Supplying Demand (CAD) during a given time horizon (depending on the problem). It is this CAD, which reflects the future cost of the optimal operation of the system given in a set of realization of the represented stochastic processes (or chronicles in the jargon used in the electrical sector) that is used as a cost function to evaluate the Individuals (with the meaning they have depending on whether the problem is PIG, PAM or OptimA).

The SimSEE platform allows simulations of the electricity generation system of a region or country. In the simulation it is possible to represent both the generation plants based on fossil fuels (fuel oil, diesel, etc.) and hydroelectric generation plants (with and without reservoirs), plants based on renewable sources such as wind or solar, battery banks, etc. It is also possible to represent in the simulations the interconnections with other electrical systems (for example Uruguay with Argentina and Brazil). This brief description is made to show clearly that the simulation of a system has associated the representation of a reality full of details and uncertainties. As already mentioned, these simulations are performed by simulating many "chronicles" (or "possible stories" or "realization of the stochastic processes involved", all expressions that mean the same in the context of this document). Normally, as the main result of the simulation, the expected value of the Demand Supply Cost (CAD) is obtained, which is the integral of the costs incurred in the time horizon considered.

An important aspect of the type of optimization problem is that the Cost Function is evaluated by Monte Carlo simulations and, consequently, what is obtained is an estimate of its value, an estimate that improves with the amount of draws used. It is also interesting that each Monte Carlo draw is associated with an evaluation of the function that is independent of the rest and this has two direct implications: a) allow the distributed evaluation of the different draws and b) be able to obtain intermediate estimators that are improving with the amount of evaluations that are carried out.

In formal terms, the cost function to be evaluated is of the type:

$F(X,r)$ where $X \in D$ is the vector of optimization parameters of the problem that can take values in the "Problem Domain" D and r identifies a possible embodiment of the set of stochastic processes. The function $F(X,r)$ is "the cost" of operating the system if the parameters are set to X and the simulation is performed with the chronicle (luck, or realization of the stochastic processes) r .

We are interested in solving the problem of finding "the best" set of parameters X , meaning "the best" one that minimizes a cost function that can be built from the evaluations $F(X,r)$ in the set of r . As an example, (and surely the objective of greater use) is to seek to minimize the expected value of $F(X,r)$ in the assembly (set of possible embodiments) of the r .

In the case of the expected value, the objective function to minimize is:

$$f(X)=\langle F(X,r)\rangle_r$$

In the simulation examples of the electric power generation system, the function $F(X,r)$ is of "high evaluation cost" (power and calculation time) and therefore the results of each evaluation are stored in the BDP to try to make the most of them and try not to repeat evaluations.

In the example, considering the electric power generation system of Uruguay, a simulation of 100 chronicles, of the type carried out for investment planning purposes, with a weekly horizon of 30 years, takes the order of 10 minutes of calculation in a desktop PC. If thousands of possible expansion plans have to be evaluated, the resolution on a single desktop computer becomes impossible.

The developed algorithm explores the domain D using different "scanning agents" (Explorers) that can behave differently. All Explorers feed the results to a shared Problem Database (BDP). Different Explorers run on different nodes of a calculation network, thus managing to distribute the exploration among many Explorers working in parallel.

As for the evaluation of a point $X \in D$, the implementation allows to indicate a subset of the assembly. This implies that the same point can be calculated more than once, with different subsets of chronicles. Each new information is thus used to improve the representation of the objective function. If the same point is evaluated in the subsets of realizations $s_{1,} s_{2,} \ldots\ldots, s_k$, the estimation of the objective function will improve in a series that tends to the true value if each new evaluation includes the information of the previous ones, $f_1(X), f_2(X), \ldots, f_k(X) \rightarrow f(X)$

The scan of the domain D is then carried out in a distributed way, by several Explorers, using the information of all the evaluations shared permanently among the Explorers. Each Explorer, using an algorithm for estimating a new point (Individual), generates a proposal for a new evaluation point and performs an evaluation with a set of realizations. This results in an "estimate" of the objective function at that point. If the point had already been calculated, the new information is integrated by improving the existing estimate.

The main components of the algorithm are:

1) Communication layer to allow distributed calculation.
2) Calculation explorers of different types.
3) Evaluator of the function $F\ X, r$. This function evaluator is the part of the algorithm that needs to be defined for each type of problem to be solved.
4) Incremental improvement of estimates.

2. OddFace_prepare – User Manual.

2.1. Introduction.

In order to use OddFace, it must be installed on a server. This section describes the OddFace_prepare application that allows the definition of problems for resolution by OddFace on a calculation server.

At the server level, it is possible to define Users to allow access to the OddFace installation. In turn, users are grouped into Groups that share the definition of Optimization Problems.

In the server installation you can add applications to solve each type of problem such as:

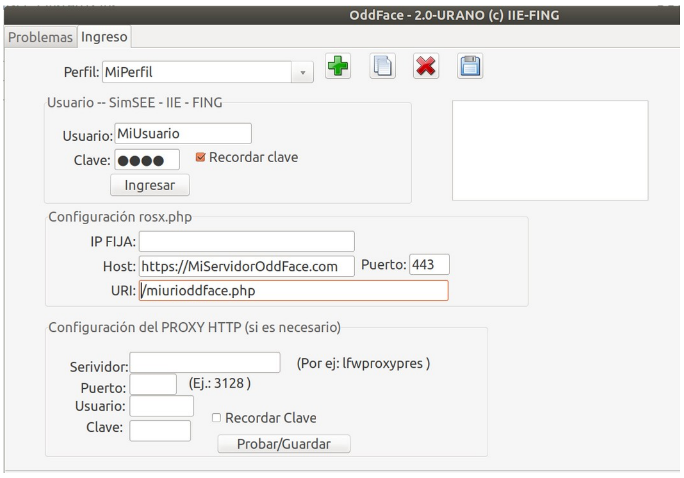
OddFace_PIG = Generation Investment Planning.

OddFace_PAM = Annual Maintenance Plan.

OddFace_OptimA = LNG Shipping Agenda Optimizer.

In configuring the groups on the server, each group must be granted permission on the type of problems it can solve.

The "OddFace_Prepare" application allows to edit Problems for their solution with "OddFace_PIG" and with "OddFace_PAM", etc.



OddFace - 2.0-URANO (c) IIE-FING

Problemas | Ingreso

Perfil: MiPerfil

Usuario -- SimSEE - IIE - FING

Usuario: MiUsuario

Clave: ●●●● ☑ Recordar clave

Ingresar

Configuración rosx.php

IP FIJA:

Host: https://MiServidorOddFace.com Puerto: 443

URI: /miurioddface.php

Configuración del PROXY HTTP (si es necesario)

Servidor: (Por ej: lfwproxypres)

Puerto: (Ej.: 3128)

Usuario:

Clave: ☐ Recordar Clave

Probar/Guardar

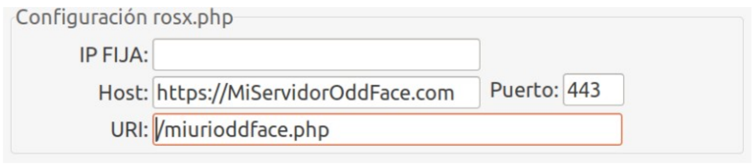
Fig. 3: User Validation Form

2.2. LOGIN.

Fig.5 shows the user validation and User Profiles configuration form. The form has a Profiles selector at the top. The values displayed in the other parts of the form correspond to the selected profile. In the profile selector, you can change the name of the profile in order to save it with a different name.

The Username and Password must be configured to give access to the server. Profile information is saved either by pressing the button (reminiscent of the time we were saving on diskettes) or automatically by pressing the button Ingresar .

The set of buttons:



Configuración rosx.php

IP FIJA:

Host: https://MiServidorOddFace.com Puerto: 443

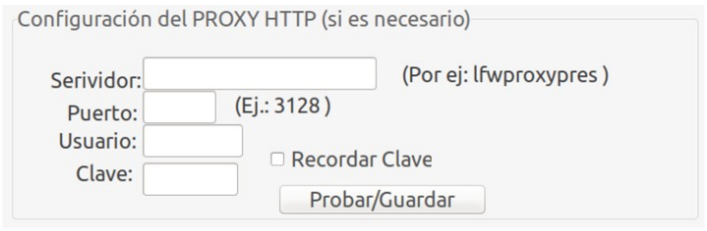
URI: /miurioddface.php

Fig. 4: Server connectivity configuration.

allow you to add a new clean profile, create a new profile from copying the currently selected one, delete the selected profile and save the profile set to disk.

Fig.4 shows the panel that allows configuring connectivity with the server. As can be seen, you must indicate the protocol (http or https) and the host (Host field) and the corresponding port for the protocol (Port, in figure 443 standard port for the https protocol). If it is a server with a FIXED IP and does not have an assigned name, you can use the FIXED IP field instead of the Host field (in which case it should be empty). In the URI field: you must put the location of the php script that receives the OddFace requests. The information in this panel should be provided by who has configured the OddFace server.

Depending on your location, you may need to configure a proxy to access the OddFace server. To do this you must use the panel shown in Fig.Error: no se encontró el origen de la referencia. The proxy information depends on the local network to which it connects. As a general rule, if with your internet browser you can access the OddFace server using the settings you have placed in the panel corresponding to Fig.5, you should be able to configure OddFace_prepare to connect as well. If you cannot connect by leaving the Proxy information blank and do not know the information on it, you can try to check your Internet browser settings to see what the settings are. The Test / Save button allows you to test if you have configured the Proxy correctly.

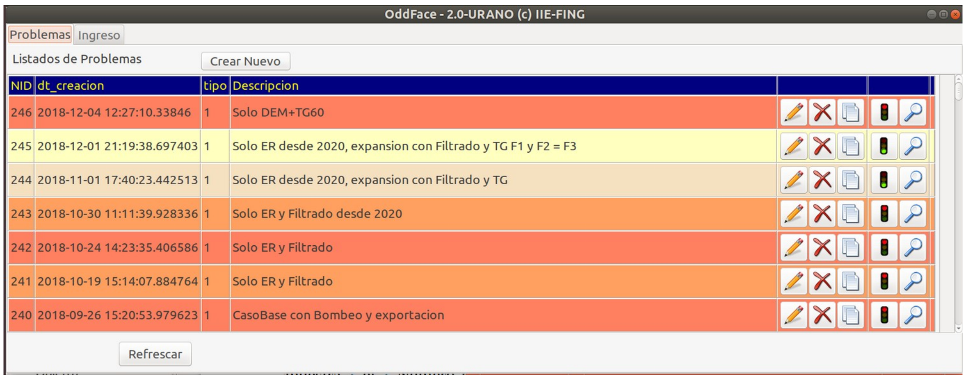

Configuración del PROXY HTTP (si es necesario)

Servidor:		(Por ej: lfwproxypres)
Puerto:		(Ej.: 3128)
Usuario:		
Clave:		☐ Recordar Clave

Fig. 5: Configuration of a proxy.

2.3. Problems List.

After logging in, a list of the defined OddFace problems is displayed. Fig.6 shows this screen. In this case, as you can see, there are several defined problems. The first column of the list shows the Identification Number (NID) of each of the problems. In the example, the problems with NID 245 and 244 have a lighter color than the rest. This lighter color is due to the fact that these problems are "active" while the rest are "inactive". The buttons: allow you to “Edit the problem parameters (the pencil)”, “Eliminate the problem (the cross)”, “Create a new problem by duplicating a problem (the overlapping sheets)”, “Activate or Deactivate (according to the traffic light be in red or green respectively) ” and “Explore the results of the problem (the magnifying glass) ”.


OddFace - 2.0-URANO (c) IIE-FING

NID	dt_creacion	tipo	Descripcion
246	2018-12-04 12:27:10.33846	1	Solo DEM+TG60
245	2018-12-01 21:19:38.697403	1	Solo ER desde 2020, expansion con Filtrado y TG F1 y F2 = F3
244	2018-11-01 17:40:23.442513	1	Solo ER desde 2020, expansion con Filtrado y TG
243	2018-10-30 11:11:39.928336	1	Solo ER y Filtrado desde 2020
242	2018-10-24 14:23:35.406586	1	Solo ER y Filtrado
241	2018-10-19 15:14:07.884764	1	Solo ER y Filtrado
240	2018-09-26 15:20:53.979623	1	CasoBase con Bombeo y exportacion

Fig. 6: Problems List.

If the light is green (problems 244 and 245 in Fig.6) the entire row will be lighter in color than if it is red. The green light indicates that the problem is “active” and therefore on the server side it will be distributed among the different calculation nodes. If the traffic light is red the problem is "inactive" and therefore will not be distributed for execution.

When you want to modify any parameter in an existing problem it is advisable to stop the execution and wait for a while (about 20 minutes) to make sure that all the Cluster browsers have stopped. In this way, when the execution starts again, the explorers will begin to evaluate with the new data. If the execution of the problem is not stopped, the new browsers that are triggered will use the new data, but those that are already running will continue with the initial configuration.

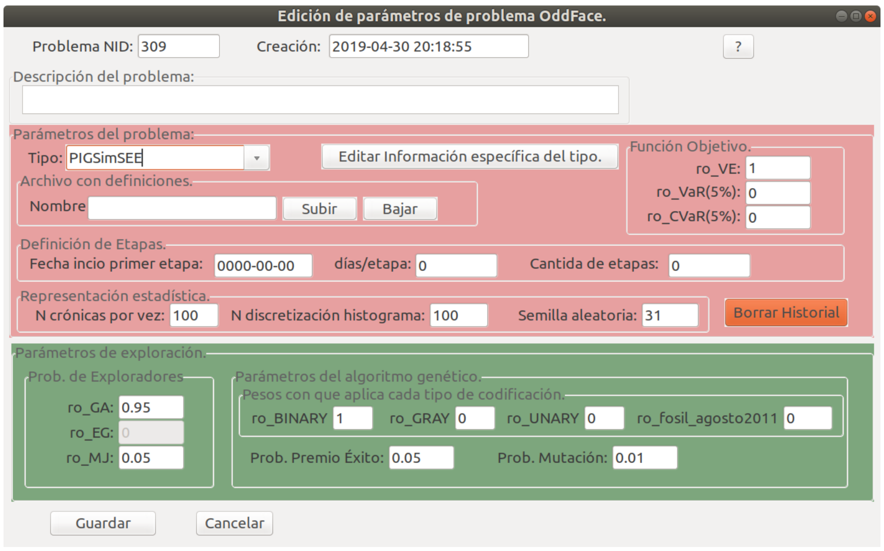
In the first column the problem NID appears, which is an identification number that is automatically assigned when a problem is created.

In the second column the “dt_creación” appears, which is the date and time at which the problem was created.

The third column shows the type of problem, with 1 being an "OddFace_PIG" and 2 being an "OddFace_PAM".

The description of the problem introduced by the user during its editing appears in the fourth column.

2.4. Editing the parameters of an OddFace problem.


Edición de parámetros de problema OddFace.

Problema NID: 309 Creación: 2019-04-30 20:18:55 ?

Descripción del problema:

Parámetros del problema:

Tipo: PIGSimSEE Editar Información específica del tipo.

Archivo con definiciones.
Nombre Subir Bajar

Función Objetivo.
ro_VE: 1
ro_VaR(5%): 0
ro_CVaR(5%): 0

Definición de Etapas.
Fecha incio primer etapa: 0000-00-00 días/etapa: 0 Cantida de etapas: 0

Representación estadística.
N crónicas por vez: 100 N discretización histograma: 100 Semilla aleatoria: 31 Borrar Historial

Parámetros de exploración.

Prob. de Exploradores
ro_GA: 0.95
ro_EG: 0
ro_MJ: 0.05

Parámetros del algoritmo genético.
Pesos con que aplica cada tipo de codificación.
ro_BINARY 1 ro_GRAY 0 ro_UNARY 0 ro_fosil_agosto2011 0
Prob. Premio Éxito: 0.05 Prob. Mutación: 0.01

Guardar Cancelar

Fig. 7: Problem editing form.

By pressing the “Create New” button to create a new problem or the “Pencil” to edit an existing one (see Fig.6) you can access the edition form of an OddFace problem shown in Fig.7.

As you can see, the form is divided into three Panels. In the first one there is the identification of the problem, in the "Problem parameters" panel there are the main parameters and it defines the structure of the problem and in the "Exploration parameters" panel there are the parameters that configure the behavior of the browsers but that do not change the structure of the Problem. It is important to note that during the execution of a Problem, it is possible to change the parameters of the “Scanning Parameters” panel, and the evaluations already made stored in the BDP remain valid. On the contrary, if any parameter of the “Problem Parameters” panel is changed, the structure of the problem will be changed, so you must stop the execution and delete all the evaluations stored in the BDP (use the “Clear History” button available for this in the same Panel).

The “Problem NID:” field shows the Identification Number (NID) of the Problem. The "Creation" field shows the creation date of the problem. These two fields are not editable by the user.

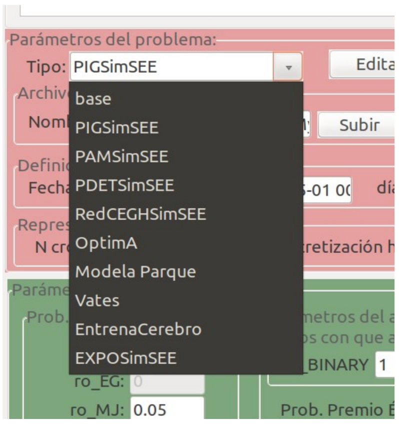
The “Problem description” field allows the user to enter a text with the description of the problem that later appears in the problem list (Fig. 6). It is important that you enter a text that allows you to identify the problem.

a) Type.

This selector will allow the user to select among the types of problems that the group to which the user belongs is authorized to run on the OddFace server.

b) Type information editor.

This button allows you to edit information specific to the selected Problem Type. Depending on the type you will have or not an additional parameter editing form. The detail of the edition of each type should be read in the corresponding manual. Fig.8 shows the selector displayed with the types available to date (May 2019).



- base
- PIGSimSEE
- PAMSimSEE
- PDETSimSEE
- RedCEGHSimSEE
- OptimA
- Modela Parque
- Vates
- EntrenaCerebro
- EXPOSimSEE

Fig. 8: Available types (May 2019)

c) File with definitions.

Pressing the "Upload" button will allow you to select a file from your computer to upload to the OddFace server associated with the Problem you are editing. In PIGSimSEE, PAMSimSEE and OptimA problems, this file is a SimSEE Room. In other types of problems, the file may contain other information. e.g. in the "Model Park" problem, the file contains the time series of the wind station's weather station and the time series of the power generated to calibrate the model parameters.

d) Objective Function.

This panel allows to define how the objective function is constructed based on the histogram of the evaluations $F(X, r)$ by assigning weights to combine the Expected Value, Value at Risk 5% (VaR = Value at Risk) and the Value at Risk Conditioned at 5% (CVaR = Conditional Value at Risk). With these weights, the Cost Function to be minimized is formed as shown in eq.

$$C=\rho_{VE}\langle F(X,r)\rangle_r+\rho_{VaR}VaR(F,5\%)+\rho_{CVaR}CVaR(F,5\%)$$ ec.(1) Cost Function.

Where:

- it must be fulfilled that the sum of the weights is equal to 1 $\rho_{VE}+\rho_{VaR}+\rho_{CVaR}=1$
- $VaR(F,5\%)=v/P(F(X,r)>v)=5\%$
- $CVaR(F,5\%)=\langle F(X,r)\rangle_{5\%mayores}$

Fig.9 shows the example of the evaluation of 1000 chronicles of the operation of the Uruguayan generation system for the next 30 years, updating the values at 5% per year. In summary, using the

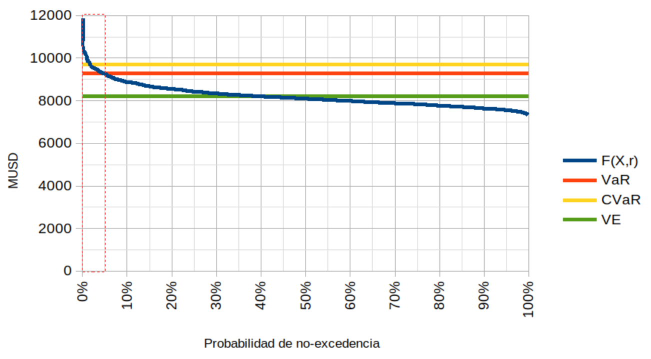


Fig. 9: Permanence of F (X, r) and definitions of VE, VaR and CVaR

parameters ro_VE, ro_VaR and ro_CVaR you can configure the risk aversion you want to consider. If you do not want to use risk aversion simply set ro_VE = 1 and the other two parameters to zero. (This is the most common use).

In some OddFace applications, the results are not stochastic (that is, the evaluation does not depend on a set of simulation chronicles). In those cases, VE = VaR = CVaR, so the risk aversion configuration is meaningless.

e) Stages Definition.

While exploration algorithms could be applied to problems in which time does not intervene, it is very common that optimization involves finding the best set of parameters in a set of time stages. For this reason, and guiding the solution towards its integration with the SimSEE platform, the description of a set of "stages" that should be considered as "decision steps" is included among the parameters. The parameter vector X is composed of a set of sections that describe the possible parameters of each stage. If the OddFace problem that is being solved does not imply temporary stages, it is sufficient to consider that there is only one stage.

- First stage start date: The format of this field is year-month-day (yyyy-mm-dd) using four digits for the year and two digits for the month and day (ISO format). Specify the start date of the first decision stage.
- Days / stage: This field must have the number of days per stage, that is, the number of days between decisions.
- Number of stages: This field indicates the total number of decision stages.

f) Statistical representation.

- N chronicles at a time: It is the amount of chronicles used in each simulation.
- N histogram discretization: The number of points used to represent the histogram of f (x). (In this first version this number must be equal to "N chronicles"). The idea is that it may be different, but for simplicity of this first implementation, they are set equal.
- Random Seed: Used to initialize all random number generators in the specific Robots for evaluating a Problem. The random seed is imposed with a value equal to the value specified in the form + number of evaluations of the individual. In this way, it will happen that the first evaluation has seed specified in the form, the second one has as seed the one specified plus one and so on. This allows all individuals to be subjected in their first evaluation to the same realization of the stochastic processes and therefore allows to have a better estimate of the individual's performance (conditioned by its DNA) regardless of “the luck” that they have to live (they subject them to the same fate).

g) Delete history.

This button removes all the problem assessments registered in the BDP. It is reasonable that if you change any parameter that changes the structure of the problem, eliminate all previous evaluations performed to start over. Before clearing history, it is also reasonable that you have deactivated the execution of the problem (putting a red traffic light on the problem in the List of

Problems Fig.6) and that sufficient time has elapsed so that all the Robots that were working on the problem have tried to communicate with the BDP and be notified that they must stop.

h) "Scanning parameters" Panel

The "Scanning parameters" panel (green color) contains the parameters that determine the behavior of the Explorers trying to solve the problem. The changes you make to these parameters will have an effect on the new browsers. If you want all explorers to change their behavior as quickly as possible, you must deactivate the problem (change it to red traffic light), wait for the time it takes to run the simulations plus a few minutes and then activate the problem again. As mentioned, the parameters of this panel change the behavior of the explorers but do not change the structure of the problem. For this reason, it is not necessary in this case to clear the evaluation history.

- ro_GA, ro_EG and ro_MJ: They are the probabilities that the mechanism of proposal of new exploration point by the genetic algorithm (GA), by gradient estimation (EG) or that the evaluation is repeated on one of the best points already evaluated in order to improve its estimate (MJ). In this first implementation, only the GA and MJ possibilities are available. The implementation of EG is unfinished, so that parameter must remain at 0 (zero). In the example of Fig. 7, with a 95% probability the next point to explore using the genetic algorithm is suggested and with a 5% probability the evaluation of one of the points selected "as best" will be improved. The selection mechanism of the genetic algorithm regulated by the GA_premio_exito parameter described below is used for the "best" selection criteria.
- Type of Coding: OddFace can apply the genetic algorithm with different ways of coding the parameters. In these boxes you can specify what type of coding you want to use, and for this you must enter a value between zero and one in each box in order to choose or combine the options. The sum of all values must be 1. BINARY, GRAY, UNARY and fosil_agosto 2011 are currently available.
    - ✗ *BINARY*: each integer optimization parameter X / (X> = Xmin). (X <= Xmax) as the binary number encoding (X-Xmin). If the parameter is real, it is specified as Y / (Y> = Ymin). (Y <= Ymax) and the amount of bits to be used for value coding (Y-Ymin). In the case of the type of PAM problem, all parameters are integers. The crossing operation of the genetic algorithm involves traversing the binary chains and taking the 50 bit of the parent A with 50% probability and the 50 bit of the parent B with 50% probability. On the string thus obtained, the mutation operation is subsequently applied, which implies altering some bits with a given probability. Two individuals that are "close" from the point of view of the value of the parameters, may be distant in terms of the differences in the binary chains that represent the DNA. For example, the binary string 0001 and 1110 may be representing the value 16 and 15 respectively for a parameter that can take values between 0 and 31. The crossing of 0001 and 1110 can for example result in 0000 or 1111 (zero and 31) , with which it can be seen that crossing two close individuals from the point of view of the parameters leads to distant individuals.
    - ✗ *GRAY*: Two consecutive numbers have only one bit difference in the coding. This code is used to try to reflect the proximity in the binary chains, the GRAY code is used. When GRAY coding is used, before crossing and mutations, the DNA is transformed from its BINARY representation to GRAY code. The crossing and the mutation are then performed (thus generating individuals that are related to their parents) and then the result is converted from GRAY code to simple BINARY.
    - ✗ *UNARY*: it would be the equivalent of assuming that the value of the parameter is

represented by the amount of 1s (ones) in a set of bits. For example, to represent numbers from 0 to 31 it could be assumed that 310-bit sets are used (it is just an example). Thus, the amount of 1s (ones) in the 310 bits, divided by 10 is the value of the parameter. With this type of coding, when performing the crossing operation an average value is obtained between the two sets of 1s (ones) with dispersion around that value. The implementation carried out in OddFace is not exactly a representation by a set of Zeros and Ones, but it was decided to operate directly on the parameters. Given two individuals with parameters X1 and X2, it is generated by crossing, the individual $X=\frac{(X_1+X_2)}{2}+z(X_2-X_1)$ with $z\in U(-1,1)$ and limiting X to the range Xmin, Xmax that corresponds.

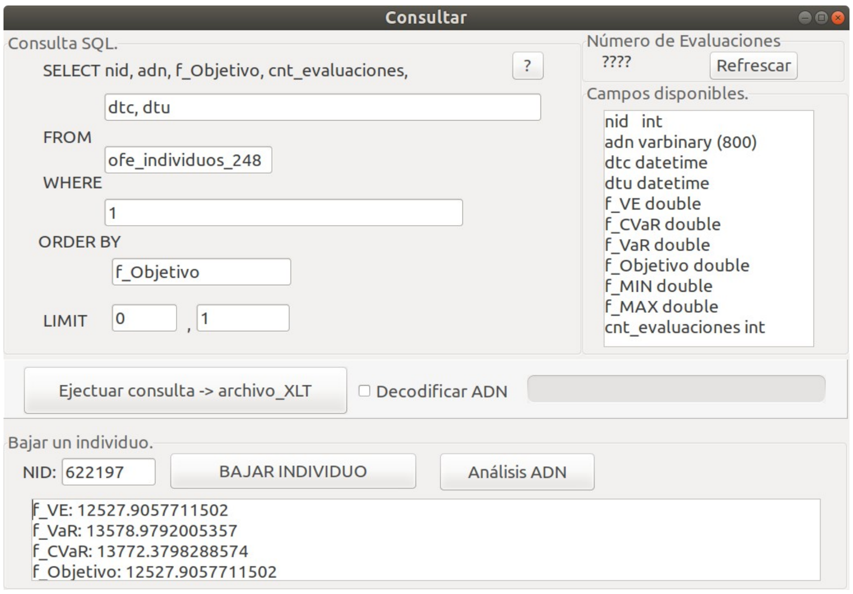
✗ *ro_fosil_agosto 2011*: In this version of OddFace it is not implemented. It should be left at zero until this type of coding is implemented.

- GA_premio_exito: This parameter regulates the selection behavior of individuals of the genetic algorithm. At each step, the explorer selects two individuals as the parents to cross them and propose a new individual. For the selection of the parents it starts of an ordered list of the individuals evaluated (registered in the BDP), sorted in descending order of merit (increasing the value of "f_Objective"). With probability "GA_premio_exito", the first individual is selected, and with (1-GA_premio_exito) no. If the first one is not selected, the same procedure is repeated with the list, removing the first one and so on. This leads to the selection of the best ones with the highest probability. The probability of selecting the most successful is $GA_{prob_{exito}}$, that of the following $(GA_{prob_{exito}}-1)GA_{prob_{exito}}$ and by induction, the probability of selecting the $k-ésimo$ individual in the list sorted by increasing order of success is: $(GA_{prob_{exito}}-1)^{k-1}GA_{prob_{exito}}$.
- GA_prob_mutación: After the parents have been selected and crossed by a simple random combination of their genes, the "mutation" mechanism that inverts bits of the binary chain that represents the individual's DNA is applied. "Prob. Mutation "is the probability of bit mutation of the DNA chain. As a problem has a longer DNA chain, the probability of a mutation occurring is greater for the same parameter "Prob Mutation". In the implementation, given the dimension N of the DNA bit string, a random source is constructed that determines the amount m of mutation sequences to be applied as $p_N(m)=C^N_m P^m_{Mut}(1-P_{Mut})^{(N-m)}$. Given a DNA string (bit sequence) this source is used to determine the amount m of mutation sequences to impose. Each mutation sequence is applied by selecting a bit with uniform distribution in the bit string and inverting it. This process is repeated m times. It may be that a bit previously inverted is inverted again.

2.5. Consultations on the history of evaluations.

The evaluation history is stored in a table in the BDP shared by all calculation nodes.

To consult the history you can use the button 🔍 of the list of problems (Fig.6).



SELECT nid, adn, f_Objetivo, cnt_evaluaciones,
dtc, dtu
FROM
ofe_individuos_248
WHERE
1
ORDER BY
f_Objetivo
LIMIT 0 , 1

nid int
adn varbinary (800)
dtc datetime
dtu datetime
f_VE double
f_CVaR double
f_VaR double
f_Objetivo double
f_MIN double
f_MAX double
cnt_evaluaciones int

NID: 622197

```
f_VE: 12527.9057711502
f_VaR: 13578.9792005357
f_CVaR: 13772.3798288574
f_Objetivo: 12527.9057711502
```

Fig. 10: Results inquiry form.

Pressing that button displays the query form shown in Fig.10. The "SQL Query" panel allows the editing of the parameters of an SQL query to be performed on the problem evaluation history table. As you can see, the fields "nid", "adn", "f_Objective", and "cnt_evaluations" are selected by default. In the edit box below, you can add additional fields. The fields in the table are listed in the box to the right of the same panel.

The meanings of the fields are as follows:

- "nid", unique identifier number of the evaluated point.
- “adn”, is the individual's DNA (binary number resulting from crossings and mutations) and is shown in hexadecimal code.
- "f_Objetivo", value of the objective function at the point evaluated.
- "dtc", date and time of the first evaluation of the point.
- "dtu", date and time of the last evaluation of the point.
- "cnt_evaluaciones", number of evaluations made of the point.
- “f_VE ”,“ f_VaR ”,“ f_CVaR ”correspond to the expected value, the Value at Risk 5% and the Value at Risk conditioned to 5% of the set of chronicles in which the individual was evaluated.
- "F_Objective" is the value of the objective cost function of minimization.
- “F_MIN” and “f_MAX” are the minimum and maximum cost values corresponding to the set of chronicles evaluated.

The FROM box has the name of the problem history table and cannot be edited.

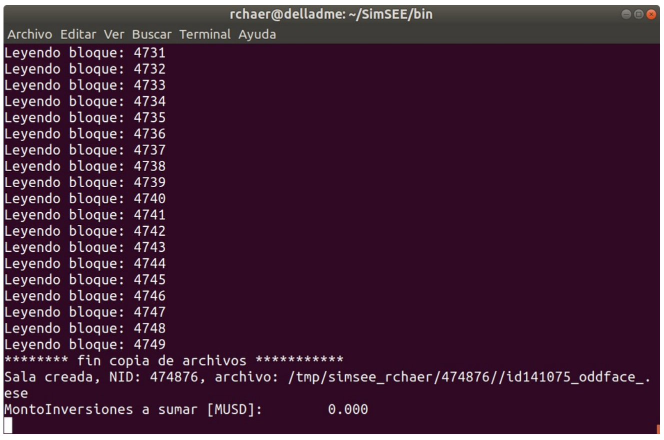
The WHERE box has the "filter" and by default it has 1, which means that all records will be selected. For example, if "cnt_evaluations"> 4 were placed in the WHERE box, only those points that have at least 5 evaluations will be selected.

The ORDER BY box indicates the order in which the results will be sorted. In the example, they are ordered in descending order of the unique identifier number (nid DESC), which means that they will appear in reverse order of creation, that is, at the beginning the last ones created.

The boxes to the right of LIMIT set the offset from the first record and the number of records to download. In the example in the figure, the values are 0 (zero) displacement and 10,000 (ten thousand) records.

Once the parameters that allow you to form the SQL query have been established, pressing the "Execute query -> XLT_file" button sends the query to the server and the selected records saved in an .xlt file will be received (text file with numbers with a "." as a decimal separator and separated by tabs).

The "Download individual" panel allows you to specify the "NID" of the individual you wish to download. By pressing the "Download Individual" button, the DNA of the individual identified by the specified NID is downloaded and the Evaluation Robot is run locally to decode the DNA and create the corresponding representation. For example, in


rchaer@delladme: ~/SimSEE/bin
Archivo Editar Ver Buscar Terminal Ayuda

```
Leyendo bloque: 4731
Leyendo bloque: 4732
Leyendo bloque: 4733
Leyendo bloque: 4734
Leyendo bloque: 4735
Leyendo bloque: 4736
Leyendo bloque: 4737
Leyendo bloque: 4738
Leyendo bloque: 4739
Leyendo bloque: 4740
Leyendo bloque: 4741
Leyendo bloque: 4742
Leyendo bloque: 4743
Leyendo bloque: 4744
Leyendo bloque: 4745
Leyendo bloque: 4746
Leyendo bloque: 4747
Leyendo bloque: 4748
Leyendo bloque: 4749
******** fin copia de archivos ***********
Sala creada, NID: 474876, archivo: /tmp/simsee_rchaer/474876//id141075_oddface_.
ese
MontoInversiones a sumar [MUSD]:        0.000
```

Fig. 11: Example of writing in the text output when downloading an Individual.

the problems of the PIGSimSEE type, the “oddface_pig” application is executed locally to create the SimSEE room corresponding to the downloaded individual.

To do this, a version of the application corresponding to the type of problem being solved must be installed on the computer on which "OddFace_prepare" is running so that it can be run on the downloaded DNA and create the corresponding representation. When this query is made, the messages on the Oddface console screen should be verified, as the application that is running can write relevant information or report a failure. In the case of PIGSimSEE problems, the place where the Room that represents the individual was created is written on the console and the total amount of investments updated that is not represented in the Room. So that you can read the outputs of the console, after finishing it, it remains waiting for the user to press "Enter", after which you can return to the query form.

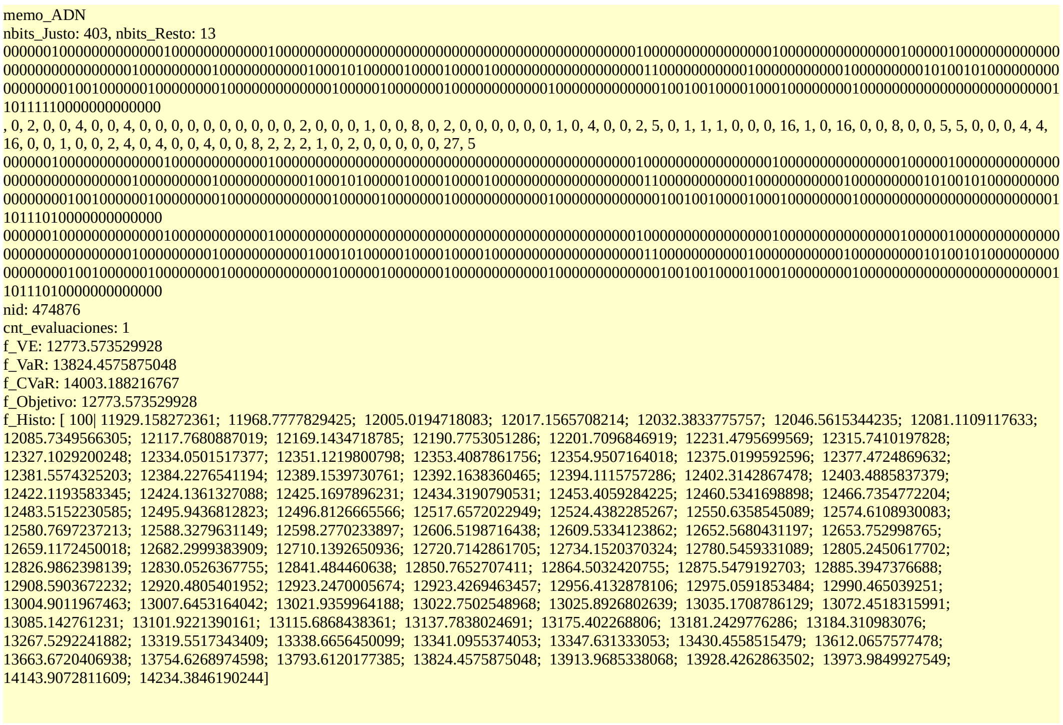
The room is downloaded to a folder in the temporary directory (its location depends on whether it is Linux or Windows), with the original name of the room concatenated with the text: "_oddface_". Fig. 9 shows an example of individual discharge in the case of a PIGSimSEE problem.

Each type of problem must implement where the decoded Individual is stored and how he notifies it in the text terminal.

The "DNA Analysis" button allows you to see the binary chain of an individual. For this, you must enter the NID of the individual you want to see and then press that button. It may be necessary to click on the text box to display the results. Fig. 12 shows an example of the result of pressing that button. This functionality was implemented mostly for purposes of code debugging and research on how to code problems. In the current use of OddFace you hardly need to use it.

First, the string is shown as it is in the database, then the decoded values are shown and then the re-encoded DNA, but "cleaning the redundant bits".

A main representative of the set of DNAs that are decoded with the same result is obtained. The three strands of DNA shown should be the same if the redundant bits are not operated at any time and that is why this functionality is only for debugging the implementations.



```
memo_ADN
nbits_Justo: 403, nbits_Resto: 13
000000100000000000001000000000000100000000000000000000000000000000000000000000000000000000100000000000000001000000000000000100000100000000000000
000000000000000010000000001000000000001000101000001000010000100000000000000000000011000000000001000000000001000000000101001010000000000
000000001001000000010000000001000000000001000001000000010000000000001000000000000010010010000100010000000010000000000000000000000001
1011111000000000000
, 0, 2, 0, 0, 4, 0, 0, 4, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 0, 0, 0, 1, 0, 0, 8, 0, 2, 0, 0, 0, 0, 0, 0, 1, 0, 4, 0, 0, 2, 5, 0, 1, 1, 1, 0, 0, 0, 16, 1, 0, 16, 0, 0, 8, 0, 0, 5, 5, 0, 0, 0, 4, 4,
16, 0, 0, 1, 0, 0, 2, 4, 0, 4, 0, 0, 4, 0, 0, 8, 2, 2, 2, 1, 0, 2, 0, 0, 0, 0, 0, 27, 5
000000100000000000001000000000000100000000000000000000000000000000000000000000000000000000100000000000000001000000000000000100000100000000000000
000000000000000010000000001000000000001000101000001000010000100000000000000000000011000000000001000000000001000000000101001010000000000
000000001001000000010000000001000000000001000001000000010000000000001000000000000010010010000100010000000010000000000000000000000001
1011101000000000000
000000100000000000001000000000000100000000000000000000000000000000000000000000000000000000100000000000000001000000000000000100000100000000000000
000000000000000010000000001000000000001000101000001000010000100000000000000000000011000000000001000000000001000000000101001010000000000
000000001001000000010000000001000000000001000001000000010000000000001000000000000010010010000100010000000010000000000000000000000001
1011101000000000000
nid: 474876
cnt_evaluaciones: 1
f_VE: 12773.573529928
f_VaR: 13824.4575875048
f_CVaR: 14003.188216767
f_Objetivo: 12773.573529928
f_Histo: [ 100| 11929.158272361;  11968.7777829425;  12005.0194718083;  12017.1565708214;  12032.3833775757;  12046.5615344235;  12081.1109117633;
12085.7349566305;  12117.7680887019;  12169.1434718785;  12190.7753051286;  12201.7096846919;  12231.4795699569;  12315.7410197828;
12327.1029200248;  12334.0501517377;  12351.1219800798;  12353.4087861756;  12354.9507164018;  12375.0199592596;  12377.4724869632;
12381.5574325203;  12384.2276541194;  12389.1539730761;  12392.1638360465;  12394.1115757286;  12402.3142867478;  12403.4885837379;
12422.1193583345;  12424.1361327088;  12425.1697896231;  12434.3190790531;  12453.4059284225;  12460.5341698898;  12466.7354772204;
12483.5152230585;  12495.9436812823;  12496.8126665566;  12517.6572022949;  12524.4382285267;  12550.6358545089;  12574.6108930083;
12580.7697237213;  12588.3279631149;  12598.2770233897;  12606.5198716438;  12609.5334123862;  12652.5680431197;  12653.752998765;
12659.1172450018;  12682.2999383909;  12710.1392650936;  12720.7142861705;  12734.1520370324;  12780.5459331089;  12805.2450617702;
12826.9862398139;  12830.0526367755;  12841.484460638;  12850.7652707411;  12864.5032420755;  12875.5479192703;  12885.3947376688;
12908.5903672232;  12920.4805401952;  12923.2470005674;  12923.4269463457;  12956.4132878106;  12975.0591853484;  12990.465039251;
13004.9011967463;  13007.6453164042;  13021.9359964188;  13022.7502548968;  13025.8926802639;  13035.1708786129;  13072.4518315991;
13085.142761231;  13101.9221390161;  13115.6868438361;  13137.7838024691;  13175.402268806;  13181.2429776286;  13184.310983076;
13267.5292241882;  13319.5517343409;  13338.6656450099;  13341.0955374053;  13347.631333053;  13430.4558515479;  13612.0657577478;
13663.6720406938;  13754.6268974598;  13793.6120177385;  13824.4575875048;  13913.9685338068;  13928.4262863502;  13973.9849927549;
14143.9072811609;  14234.3846190244]
```

Fig. 12: DNA analysis

A case is shown in Fig. 12 in which nbits_Justo: 403, nbits_Resto: 13. In this example, the minimum amount of bits needed to encode an individual is 403 bits and the remaining 13 bits are the leftovers to encode DNA strands with 16-bit word sequences.

3. Details of the implementation of the Genetic Algorithms in OddFace.

This section describes the implementation of the Genetic Algorithms in OddFace at the level sufficient to be able to understand and make modifications to the code.

The implementation is divided into two Units (Pascal Modules) that are “utipos_ga” in which the basic types of data are defined for the representation of DNA chains and Genotypes and “uoddface” in which the base classes are defined for the description of OddFace optimization problems.

3.1. Basic definitions for handling DNA strands.

The Pascal unit "utipos_ga" contains the basic definitions for the management of Genetic Algorithms. In this section, ***“Bold italics”*** will be used to write textual definitions of the Pascal source code.

The first thing to highlight is that in the implementation the DNA chains are represented by vectors of BITs organized in 16-bit words (Pascal WORD type). That is, every DNA chain is stored as 16-bit word vectors.

In the Interface, the type is defined:

TCadenaADN = packed array of word;

Thus defined, an instance of TCadenaDNA will be a vector of length to define of data of the *Word* type (16 Pascal bits). The word “packed” tells the compiler to locate all the bytes “contiguously”, providing that we can access them directly. Likewise, the implementation always sought access through the structure of the vector, which should not be relevant if the bytes are contiguous in memory.

With this definition of DNA, if you want to access the bit *k* of a string, you have to access the data (*k div 16*) of the vector and within that data the bit (*k mod 16*). To access the bit (*k mod 16*) you have to build a bit mask that has 0 in all the bits except in the position (*k mod 16*) to achieve, by means of an AND binary operation, isolate the bit you want to access .

These constants are defined:

- ***BIT_MAS_SIGNIFICATIVO = $8000;*** This constant serves as a mask to isolate the most significant bit in a 16-bit data.
- ***BIT_MENOS_SIGNIFICATIVO = $0001;*** This constant serves as a mask to isolate the least significant bit in a 16-bit data.

Then, given a vector of DNA data to read the "k" bit (assuming k: 0 .. NBits-1) we would have to do:

mascara_de_bit = BIT_MENOS_SIGNIFICATIVO shl (k mod 16);

jw:= k div 16;

Bitk:= ADN[jw] and mascara_de_bit;

Where: the "*shl*" operator is Pascal's standard (read Shift Left) and has the effect of making a binary shift to the left of the first parameter (*k mod 16*) positions.

The "*div*" operator is Pascal's standard and calculates the integer part of the division.

The "*mod*" operator is Pascal's standard and calculates the remainder of the division.

To set the Bit k of the DNA chain to ONE, the operation would be:

ADN[jw]:= ADN[jw] or mascara_de_bit;

To zero the bit k of the DNA string, the operation would be:

ADN[jw]:= ADN[jw] and NOT(mascara_de_bit);

To reverse bit k of the DNA string the operation would be:

ADN[jw]:= ADN[jw] xor mascara_de_bit ;

a) DNA-> Genotype and Phenotype.

All the information contained in the chromosomes is known as genotype, however this information may or may not be manifested in the individual. The phenotype refers to the expression of the genotype plus the influence of the environment.

The DNA Chain is a representation of the Genotype and totally characterizes the individual from the genetic point of view. The subsequent experience of each individual in their environment may lead to differentiate individuals who are identical from the genetic point of view. For example, two twins with the same genetic information are then differentiable by external features.

The terms "genotype" and "phenotype" were created by Wilhelm Johannsen in 1911. The genotype is the complete hereditary information of an organism, even if it is not expressed. The phenotype is a property observed in the organism, such as morphology, development, or behavior.

By observing the DNA chain, the genotype can be known, observing the appearance and performance of the organism in its environment, the phenotype can be known.

The physical properties of an organism are those that directly determine its chances of survival and reproduction, while inheritance of physical properties only occurs as a secondary consequence of gene inheritance.

Mapping a set of genotypes to a series of phenotypes is sometimes called a genotype-phenotype map. In the case of application in OddFace on the SimSEE platform, the genotype of an individual determines everything necessary to be able to simulate and evaluate the performance of that individual, that is, it sets all the parameters of a SimSEE Room to do the simulation and obtain the Future cost of operating the system as a performance index. When performing a simulation, a set of chronicles (Monte Carlo samples) will be selected and so two evaluations of the same individual with two different sets of chronicles can lead to two different evaluations of the performance of the same individual. Thus, in the case of application, it involves a simulation platform that simulates "the environment" in which the "individual" (characterized by his genotype) has the opportunity to perform and thus show his phenotype.

This representation, adjusted to the reality that the same genotype can run with different luck during the evaluation, can be considered as a “robust” behavior of nature, in which an “absolute leader" is not defined but that there is a set of individuals that they are "the best", but depending on the life of each one, there are some genetics that are better, but in another circumstance the condition of being better may be reversed. This somehow prevents a UNIQUE genetics as a winner, and is important so that the adaptive capacity of organisms is not lost. But thinking of the objective of finding the optimal individual as the solution to a specific problem that has a clear specification (not changing external conditions), this influence of the "luck" of the individual during the evaluation is rather a negative aspect of the AG. In this first implementation of OddFace v1.0 it has been implemented that "there is no control" over the individual's fate and therefore, the same genotype, evaluated twice can have differentiation. In the analysis of the test cases performed for this implementation, this topic is deepened and an alternative implementation is proposed, which although it is not adjusted to reality, is considered better for the resolution of the type of test problems performed.

b) Genotype Descriptors.

In the Unit "utipos_ga" the class is defined:

```
TDescriptorGenotipo = class
  nombre: string; // identificador del parámetro
  nbits: integer;
  constructor Create( nombre_: string; nbits_: integer );
  procedure codificar_ADN( var adn: TCadenaADN; var offset: integer; var mask: word; var Genotipo ); virtual;
  function decodificar_ADN( var Genotipo; var adn: TCadenaADN; var offset: integer; var mask: word ): boolean; virtual;
end;
```

That class is used to describe a Genotype, or parameter of the individual. For example, in the case of PAM, given a Maintenance Order for a unit (For example: "Remove the 6th Central Batlle Unit for 15 days for maintenance between 1/1/2012 and 1/3/2013) the genotype could represent for that order the start date of maintenance.

In the Genotype descriptor:

- The "name" property allows us to clearly identify which parameter it refers to (for example, it could be the maintenance order number).
- The "nbits" property indicates the number of bits needed to encode the possible range of parameter variation.
- The constructor allows us to create an instance of the descriptor.
- The "codify_DNA" procedure receives as a parameter a DNA chain and the offset to the locker in which the Genotype coding begins. The "mask" parameter is a 16-bit binary mask, all zero, except in the position of the first bit of the Genotype in the DNA position [offset]. When coding the genotype, the binary representation of the Genotype parameter is copied, from the bit determined by "mask" within DNA [offset] and the "offset" and "mask" parameters are returned so that they are left pointing to the bit following the last used, within the chain to encode the Genotype. Initially setting offset = 0, mask = 1, and calling the procedure codificar_DNA on the vector of all Genotypes, the total coding of the individual is obtained in DNA.

- The "decode_DNA" function allows the genotype to be read from a DNA chain. The meanings of the “offset” and “mask” parameters are the same as those explained in the previous paragraph and allow reading from the bit string, those corresponding to the particular Genotype and modify the parameters to prepare them so that the next genotype descriptor can do his job. The result of the function is TRUE if the binary decoding did not need to be adjusted to cover the range specified for the parameter and FALSE if an adjustment had to be made. To better understand the meaning of TRUE or FALSE as a result, read the description of the handling of Integer Genotypes below.

As you can see, in this generic class, nothing is said about the type of data of the “Genotype” parameter, it is only assumed that in the memory location pointed by that parameter, nbits can be read or written.

Refined classes of TDescriptorGenotype are defined, to facilitate the handling of Boolean, Real and Integer parameters.

TDescriptorGenotipoBooleano. In this case, simply uses the generic class by setting nbits = 1.

TdescriptorGenotipoEntero. In this class the parameters “k_min and k_max” are added that must be passed in the constructor and set the range allowed for the parameter. The amount of bits is calculated in the constructor and set so that nbits is the smallest integer such that (k_max-kmin) <= 2 ^ nbits. Suppose it is an integer parameter called "Path" that can take the values 1, 2 or 3. Then, when creating the instance of the genotype descriptor we would call the constructor like this: path: = TdescripotrGenotipoEntero.Create ('Path', 1 , 3).
As a consequence of this call, a descriptor will be created, in which nbits = 2. This leads to the existence of 2 bits in the DNA strands that are to represent this genotype. As 2 bits can represent 4 values and only 3 are needed in this case, there is more than one binary encoding that will end up giving the same genotype. This is where the result of the “decode_genotype” function is involved. In the implementation it was decided to directly perform the binary decoding of the "bit segment" of the DNA to an integer obtaining a number between 2 ^ nbits-1. The value thus obtained is added to k_min to obtain the genotype value, if the result is greater than k_max, it is adjusted to k_max and FALSE is returned to indicate that an adjustment had to be made. If the result is TRUE, it was not necessary to make any range adjustment.

TDescriptorGenotipoReal. This refined class of Tdescriptor Genotype is useful for handling Real parameters (floating point). The constructor allows to pass as parameters, the genotype name, the minimum and maximum value of the parameter and a “nbits” parameter that determines the fineness with which the range (x_max - x_min) will be discretized in the representation. The range (x_max - x_min) will be represented by 2 ^ nbits points being therefore the distance between the points of discretization dx = (x_max - x_min) / (2 ^ nbits -1). As you can see, the real parameter is represented by a countable discretization, so it is essentially treated as if it were an integer parameter.

3.2. Definition of basic classes to describe the OddFace problem.

The “uoddface” Pascal unit contains the definitions of the Pascal Classes useful for the definition of OddFace problems.

The basic classes are:

- TIndividuo. This Class defines the basis from which to derive classes that can describe individuals from each specific problem. An individual differs from another in essence by its Genotype, represented in its DNA chain. The Individual is associated with a Problem and it is within the framework of that problem that it will be evaluated and depending on its performance it will be classified as more or less apt.
- TProblema. This Class defines the basis from which to derive classes for specific problems. The Problem contains the description that allows to evaluate the performance of the individuals associated with the problem. For example, in a PIG (Generation Investment Planning) problem, the Problem contains the information to generate the SimSEE Room from a given investment plan (ie an individual) and to evaluate the cost resulting from that investment plan.
- TExplorador. This Class generalizes the search mechanism for individuals. Its main method is to "propose a new individual". From this basic class it is possible to derive refined explorers with different strategies to propose new individuals.
- TExploradorGenetico. This class is a TExplorador refinement. The mechanism to propose a new individual is in this class the one of the Crossing of two individuals selected from the group of individuals already evaluated of the problem according to a performance index to obtain the DNA of the new individual. The DNA resulting from the crossing may in turn undergo changes due to the Mutation operation.
- TExploradorMejorador. This class is a TExplorador refinement. The mechanism to propose a new individual is to select one of the best ones already evaluated and simply repeat the evaluation with "other luck" to improve the evaluation of the individual.

The solution of a problem is to activate explorers that are looking for the best individuals (those that minimize the objective function of optimization).

a) TIndividuo.

The TIndividuo class has the properties:

- *Problema: TProblema;* This variable stores a reference to the problem to which the individual belongs.
- *tipo_COD: integer;* This property determines the type of encoding in which the binary representation is stored for the Cross and Mutation operations. The values can be: 0, 1 or 2 depending on whether the encoding is: BINARY, GRAY or UNARY.
- *nid: integer;* That is a unique identifier of the Individual in the table of evaluated individuals associated with the problem to which the individual belongs. If the value is -1 (minus one) it means that a unique identifier has not yet been assigned to the individual. Unique identifiers must be requested from the database server.

- *XR: TVectR;* Real genotypes vector. It contains the set of real values that represent the real parameters of the individual.
- *XE: TVectE;* Integer genotypes vector. It contains the set of integer values that represent the integer parameters of the individual.
- *ADN: TCadenaADN;* Genotype representation (*XR* and *XE*) by a bit string.
- *f_VE, f_VaR, f_CVaR, f_MIN, f_MAX, f_objetivo: NReal;* These variables store the estimates made based on sets of chronicles (Monte Carlo samples) of the expected value of the cost function, which is exceeded with probability 5%, of the expected value of the highest 5% set of costs, the minimum value of the sampled set, the maximum value of the sampled set and the value of the objective function to be minimized respectively. The target value is composed as a linear combination of *f_VE*, *f_VaR* and *f_CVaR* according to the weights specified in the problem definition. These estimators are obtained for each evaluation (performed with a number of Monte Carlo samples) and if the individual is evaluated more than once the averages of the estimators obtained in each evaluation are stored in the variables *f_VE*, *f_VaR*, *f_CVaR*, *f_objective*, and in the variables *f_MIN*, *f_MAX* the minimum and maximum values estimated in each evaluation are stored.
- *cnt_evaluaciones: integer;* Number of times this individual was evaluated. Each evaluation consists of performing a number of samples of the cost function defined in the problem specification. The same individual may have been evaluated more than once and the *cnt_evaluaciones* value realizes that.
- *f_histo: TVectR;* This property is of the real vector type and directly stores the value of the cost function obtained by each Monte Carlo simulation performed during an evaluation. For example, if in the specification of the problem it was established that each evaluation is performed on 100 chronicles (samples) then the *f_histo* vector will have the 100 values obtained during the simulation. In the case where an individual has been evaluated more than once, the *f_histo* vector stores the average of the *f_histo* values that each evaluation would have independently if it were the only one.

```
// crea un nuevo individio "limpio".
constructor CreateNew(problema_: TProblema);


// crea un individuo con un ADN dado poniendo a cero todos los demás parámetros.
// es útil para decodificar la cadena.
constructor CreateFromADN_HexStr( problema_: TProblema; ADN_HexStr:string );



// crea un individo desde un record de la DB
constructor CreateFromRec(problema_: TProblema; r: TDataRecord);


// comunica el resultado al la DB
procedure ComunicarResultado;
```

```
   // convierte el ADN a codificacion GRAY
   procedure toGray;

   // convierte el ADN a codificación BINARY
   procedure toBinary;



   function ADN_AsBinaryStr: string;
   procedure Free; virtual;
 end;
```

4. Additional considerations.

Two important aspects to consider when developing a new type of Problem Explorer in OddFace are described below.

4.1. Monte Carlo Sampling.

The first observation is that, since it is the search for a minimum of a function that is evaluated by a set of Monte Carlo draws by simulation, and since the search for the minimum is definitely the comparison between the evaluation at different points of the search space, the techniques of variance reduction by synchronized sampling are valid for this case. In other words, it would be desirable to compare the values for the same set of embodiments of the stochastic processes, or at least that, if possible, that is, in those aspects that the stochastic processes involved do not depend on the evaluation point, be maintained for both evaluations.

As an example, in the case of PAM, the accidental breakage of the machines should remain unchanged as well as other processes such as wind speeds or flow rates for hydroelectric power plants, so that they do not add differences to the comparison.

To achieve this improvement, it would be sufficient to introduce an order in the random seeds with which the points are evaluated.

If the problem were solved by admitting a single evaluation of NChronicles for each individual, it is clear that the best solution would be to use the same random seed for all evaluations. As the generation of random numbers in SimSEE is implemented, from version 2.63 "Anarchy" (the random seed) determines the generation of random numbers of each component of the simulation independently, with which, when evaluating two individuals , which ultimately means optimization and simulation of two SimSEE Rooms, if the same random seed is used, the generation of randomness in each component that remains unchanged in both Rooms will be identical.

In the management of the realizations that are used during the simulation, it must be ensured that, for the same number of evaluations, the Individuals were tested on the same sets of realizations in what can be maintained, even if the individuals are different. In the implementations of OddFace_PIG, OddFace_PAM and OddFace_OptimA est it is achieved by imposing that simulation seeds are calculated from a Seed_Mother (equal for all) plus the evaluation number. Thus generated, a seed dependent on the evaluation number, to maintain internal coherence, is responsible for

SimSEE.

4.2. Areas of Indifference in the DNA chain.

When selecting the possible ranges of the parameters and how they affect the problem under evaluation, care must be taken not to leave Indifference Zones.By areas of indifference we refer to regions of the Domain that when translated into the Problem (eg when creating the SimSEE Room in the case of OddFace_PIG) are Individuals, for all intents and purposes, equal. Indifference Zones slow down the convergence of the Genetic Exploration algorithm. At the level of the DNA chains the individuals are different, but at the level of the problem to be solved they are identical and it ends up creating crossings about Individuals that do not generate really different explorations.

4.3. Unfeasible individuals and bit residue in the DNA chain.

In the implementation of OddFace, when proposing a new individual, it could happen that it is unfeasible, indicating that it is not possible to perform the corresponding simulation to assess the cost of the individual. It should be implemented in the Explorer itself, a function that, given an Individual, perform the necessary checks and determine whether or not it is feasible. The fact that there is the possibility of generating Unfeasible Individuals generally indicates that the way of coding the parameters of the Problem could be improved, thus avoiding the generation of Unfeasible Individuals. Just as an example, suppose a problem with two parameters $x_1 \in [0,1]$ and $x_2 \in [0,1]$ but for the individual to be feasible $x_1 + x_2 < 1$ must be met. If it is thought that individuals will be allowed to vary freely x_1 and x_2 50% of the region to be explored will correspond to unfeasible individuals. In this example it would be preferable to make a change of variable and instead of x_2 considering $r \in [0,1]$ as a parameter and putting the calculation into the problem coding $x_2 = x_1 r$, thus avoiding regions with infeasible individuals.

5. Annex. Probability of Mutation.

In the problem editing form (see Fig.7) it is possible to define the probability of mutation. This parameter is used to build a random number generator with the distribution

$$p_N(m) = C_m^N P_{Mut}^m (1 - P_{Mut})^{(N-m)} \qquad \text{eq.(2).}$$

where N is the length of the problem's DNA chain (number of bits) and P_{Mut} is the probability of mutation specified in the form. With the probability density of eq.2 the number of mutations m to be performed is generated. These mutations are performed randomly on the bit string sequentially, so a bit inverted in one of the mutations can be reversed again in the following.

Given a sequence of N bits, if in step 1 any of the bits (with equal probability) is selected and reversed, the probability of inverting a given bit is $p_1 = \frac{1}{N}$. If any of the bits is selected again in step 2 and reversed, the probability of having inverted a given bit in the sequence of steps is $p_2 = p_1 (1 - \frac{1}{N}) + (1 - p_1) \frac{1}{N}$

Similarly, you can write the probability that at the end of the step $k+1$ a given bit will be inverted in the sequence of steps such as:

$$p_{k+1}=p_k(1-\frac{1}{N})+(1-p_k)\frac{1}{N}=p_k(1-\frac{2}{N})+\frac{1}{N}$$

$$p_{k+1}=p_k(1-\frac{2}{N})+\frac{1}{N} \qquad \text{eq.(3)}$$

Eq.3 as its point joined $p_{k+1}=p_k=\frac{1}{2}$ by what making the variable change $z_k=p_k-\frac{1}{2}$ we have the equation

$$z_{k+1}=z_k(1-\frac{2}{N}) \qquad \text{eq.(4).}$$

Eq.4 has a solution $z_k=(1-\frac{2}{N})^k z_0$ with $z_0=(p_0-\frac{1}{2})=-\frac{1}{2}$

From the above you can write the probability that after m steps a given bit will be inverted (that is, it has been inverted an odd number of times) as:

$$p_m=\frac{1}{2}-(1-\frac{2}{N})^m\frac{1}{2}=\frac{1}{2}(1-(1-\frac{2}{N})^m) \qquad \text{eq.(5)}$$

The possible values of m are between 0 (cero) and N . Fig.13 shows that eq.5 is visually, quasi-independent of N when plotted against m/N

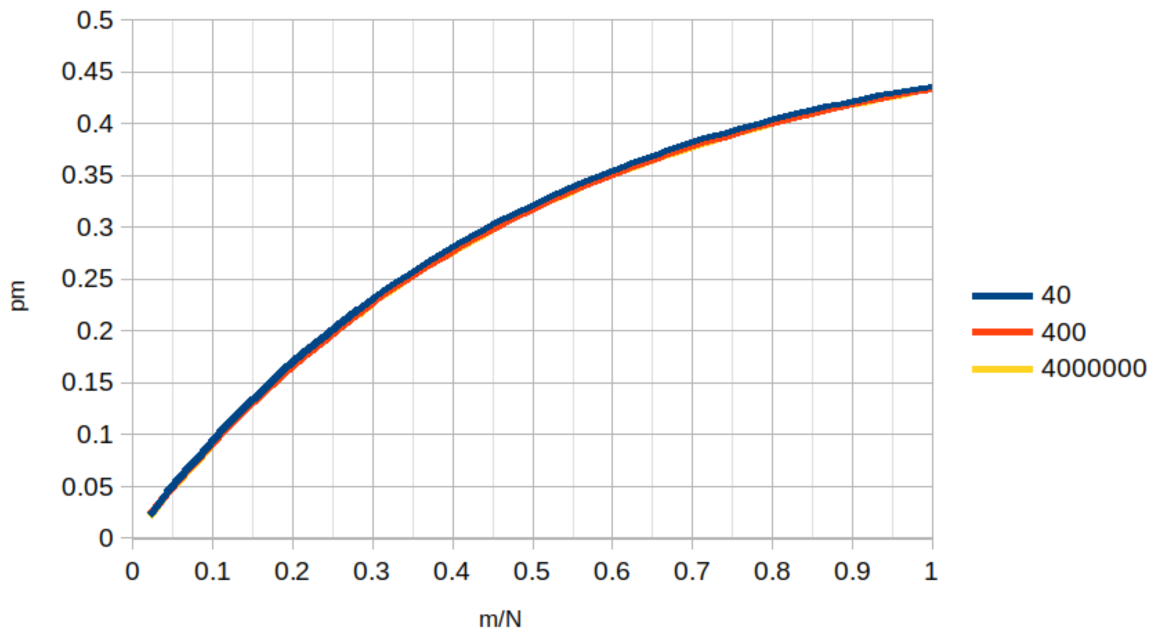


Fig. 13: Probability of mutation of each bit in a sequence of m mutations.

Then, the expected value of the inverted bits when applying m sequential mutations will be:

$$Np_m = \frac{N}{2}\left(1-\left(1-\frac{2}{N}\right)^m\right)$$ eq.(6).

$$C_m^N = \frac{N!}{m!(N-m)!}$$

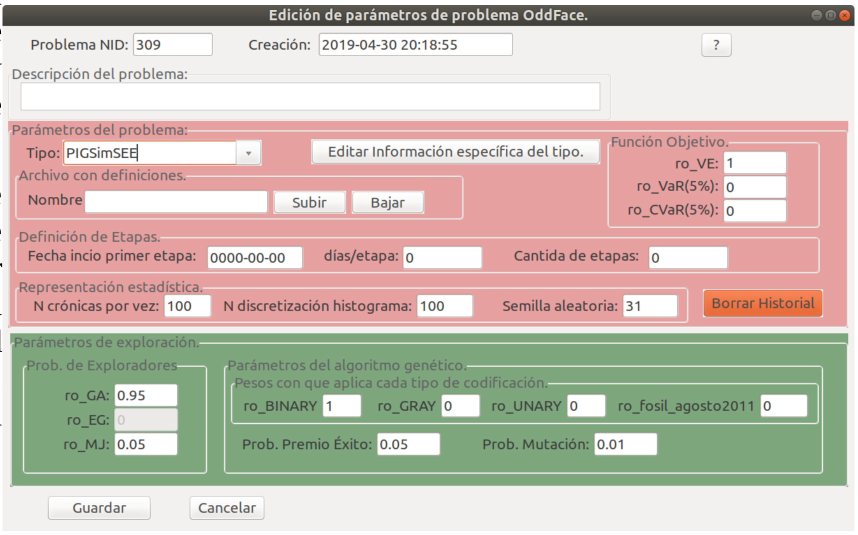
6. OddFace-PIG. Generation Investment Programming.

6.1. Introduction.

The use of the OddFace tool is explained in the OddFace Manual. The purpose of this document is to describe the OddFace-PIG application that allows the problem of optimization of the investment plan in generation to be defined on OddFace.

To evaluate an investment plan in generation, the user must provide a SimSEE Room and indicate which Actors of that Room are those on which it is possible to install new units. For this, the possible technologies are configured on which it is possible to expand (eg Wind, Solar, Open cycle turbines) and the time horizon in which it is possible to make decisions as well as the frequency with which it is possible to take them.

To define a problem of type OddFace_PIG you must use the OddFace_Prepare tool. In the User Manual of that tool you will find how to configure User access and how to start creating a new problem. When you start creating a problem, a form will open as shown in Fig.14.


Edición de parámetros de problema OddFace.
?
Editar Información específica del tipo.
Subir
Bajar
Borrar Historial
Guardar
Cancelar

Problema NID: 309 Creación: 2019-04-30 20:18:55

Descripción del problema:

Parámetros del problema:

Tipo: PIGSimSEE

Archivo con definiciones.

Nombre

Función Objetivo.

ro_VE:	1
ro_VaR(5%):	0
ro_CVaR(5%):	0

Definición de Etapas.

Fecha incio primer etapa: 0000-00-00 días/etapa: 0 Cantida de etapas: 0

Representación estadística.

N crónicas por vez: 100 N discretización histograma: 100 Semilla aleatoria: 31

Parámetros de exploración.

Prob. de Exploradores

ro_GA:	0.95
ro_EG:	0
ro_MJ:	0.05

Parámetros del algoritmo genético.

Pesos con que aplica cada tipo de codificación.

ro_BINARY 1 ro_GRAY 0 ro_UNARY 0 ro_fosil_agosto2011 0

Prob. Premio Éxito: 0.05 Prob. Mutación: 0.01

Fig. 14: Form for editing an OddFace problem.

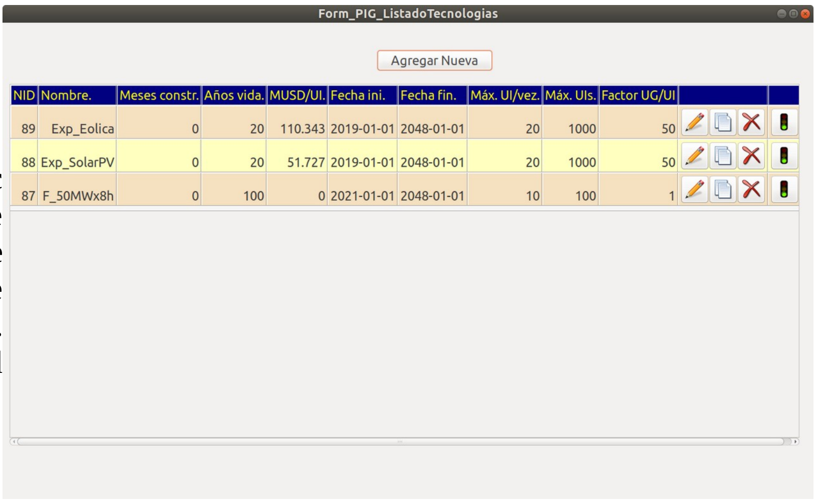
6.2. General Parameters.

In this problem editing form (Fig.14),), you must select the PIGSimSEE type of problem in the “Problem Parameters” Panel, in the "Type" selector. Once the type of problem has been selected with the "Upload" button, select the SimSEE Room file (file with extension ".ese") and the upload of the Room to the server will begin.

Before uploading the room, identify the Actors on which you want to perform the expansion. You must enter the name of those Actors as technology options in the edition of the specific problem parameters. You must ensure that these actors have only one Units Record with ZERO units installed with a date prior to the start of the simulation. This is important because if you leave records with units installed in the Actors that are an expansion option, then you will not be able to differentiate between the units installed by the optimizer and those that already existed in the Room.

6.3. Configuration of a technology.

Pressing the "Edit specific information of type" button in the problem editing form (Fig.14) opens the technology editor that will be the investment options shown in Fig.15. This list is initially created empty and



NID	Nombre.	Meses constr.	Años vida.	MUSD/UI.	Fecha ini.	Fecha fin.	Máx. UI/vez.	Máx. UIs.	Factor UG/UI
89	Exp_Eolica	0	20	110.343	2019-01-01	2048-01-01	20	1000	50
88	Exp_SolarPV	0	20	51.727	2019-01-01	2048-01-01	20	1000	50
87	F_50MWx8h	0	100	0	2021-01-01	2048-01-01	10	100	1

Fig. 15: List of investment options technologies.

the “Add New” button introduces the technologies that are investment options.

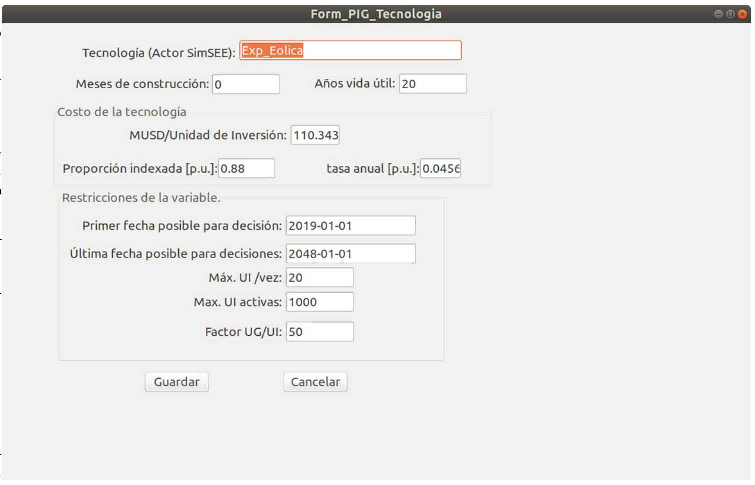
In Fig.15 it is shown that three technologies were configured. "Exp_Eolica", "Exp_SolarPV" and "F_50MWx8h". These are just examples for a SimSEE Room in which there are Actors with those names intended to expand the generation based on installing units in them.

In each line of the list there is a keypad: . The "pencil" allows you to edit the technology parameters, the "two sheets" allow you to add a new technology to the list by copying the parameters of an existing one, the "cross" allows you to delete a technology and the "traffic light" allows you to activate or deactivate the technology (only the active ones are considered in the optimization).

Keep in mind that if you change any parameter of a technology or activate or deactivate or delete or add technologies you will be changing to the structure of The Problem and therefore if it was already running you should delete the evaluations you already made using the “Clear History” button of the problem editing form (Fig.14).

Pressing the “Add New” button in the list of technologies (Fig.15) or cloning an existing one with the button will open the form for editing a technology shown in Fig.16.

In the field "Technology (SimSEE Actor):" you must enter the name of the SimSEE Actor that will be used to expand the generation based on installing Generation Units by adding the corresponding records in the Actor.



Form_PIG_Tecnologia

Tecnología (Actor SimSEE): Exp_Eolica

Meses de construcción: 0 Años vida útil: 20

Costo de la tecnología

MUSD/Unidad de Inversión: 110.343

Proporción indexada [p.u.]: 0.88 tasa anual [p.u.]: 0.0456

Restricciones de la variable.

Primer fecha posible para decisión: 2019-01-01

Última fecha posible para decisiones: 2048-01-01

Máx. UI /vez: 20

Max. UI activas: 1000

Factor UG/UI: 50

Fig. 16: Formulario de edición de una tecnología.

The "Construction Months" field allows you to define the estimated months that will pass from the investment decision until the projects of the type of technology in question are put into operation. The use of this field is optional. If you specify a value of 0 (Zero) you will simply be representing that the units enter at the same moment the decision is made. If you specify a value greater than zero, you should keep in mind that if the investment is set up as specified in section 6.3.b , then the investment is made on a date and the new generation will enter after the specified number of months has elapsed.

The field “Years of useful life” allows you to specify the useful life of a power plant of the technology in question. In Fig.16, 20 years were specified, which implies that when OddFace adds a file with technology units in the Room on a given date, on the same date 20 years later, it modifies the units record to represent the withdrawal of the system units.

The “Technology cost” Panel allows you to specify the fixed costs associated with the installation of Investment Units (UI) of the technology and is explained in detail in section 6.3.b

In the Panel “Variable restrictions” you must limit the decision variable (the number of units to be installed).

You must specify the "First possible date for the decision" and the "Last possible date for decisions". These two fields act as an additional temporary filter to the definition of the Decision Stages and can only restrict the time horizon defined by the Stages.

In the “Max. IU / time” the user must specify the maximum number of units of technology that is reasonable to be installed at a decision stage. For example, if the demand growth of the system is 100 MW per decision stage and an IU represents a 50 MW wind farm, the demand increases will be covered with 4 parks per stage (assuming a capacity factor of 0.4) and assuming that in the same stage you can remove previous parks a fair value could be 8 and to have some slack you could configure “Max. IU / time = 10 ”.

The “Max. Active UI ”specifies the maximum amount of active investment units that is reasonable in the system. As an example, if the Maximum Demand is 2000 MW and if an IU were a 50 MW wind farm with a capacity factor of 0.4 it would not make sense to keep assets much more than 2000 / (50 * 0.4) = 100 Investment Units.

It is important to reduce the ranges of the variables as much as possible, given that the greater the range, the more difficult the optimization algorithm will have to explore the domain of possible solutions and therefore the calculation time will be greater.

The “UG / UI” parameter determines how many Generation Units are incorporated into the corresponding Actor of the SimSEE Room for each Investment Unit. In the example of Fig.16 el valor de 50 implica que por cada Unidad de Inversión de OddFace, the value of 50 implies that for each OddFace Investment Unit, 50 new generation units will be installed in the “Exp_Eolica” Actor of the Example Room. As in this Room the Actor Exp_Eolica is configured with 1 MW units, a UI corresponds to 50 MW installed in the Room.

a) Fixed costs such as payments for room availability.

The Actors within the SimSEE Room have a field (in their main form or in the dynamic parameter records) that allows specifying a payment for availability in USD / MWh (dollars per MW and per hour available). This payment for availability (sometimes called Payment for Power) appears in the Room as a cost. When adding units to the Actor, the increase in fixed system costs is automatically being considered as this availability payment is applied.

This way of considering fixed costs is simple, but it makes it difficult to consider investment costs with time decay. In order to consider this type of costs with decay, the configuration option explained in the following section 6.3.b.

b) Fixed costs as specific investments in OddFace-PIG.

In Section 6.3.a, a way to consider fixed costs (Investment + O&M) in the Actors of a Room through Availability Payment was specified. This section shows an alternative mechanism, which allows to represent in a simplified way the decay over time of investment costs that are generally associated with improvements in the production of technologies.

The “Technology cost” Panel (see Fig.16) allows establishing the initial investment cost for each Investment Unit (UI) of the technology and its evolution over time.

The parameter “MUSD / Investment Unit” must be completed with the millions of dollars that an Investment Unit of the technology costs in the instant t_0 .

The parameter “Indexed proportion [p.u.]” must specify what proportion of the value specified in the previous parameter is subject to a decay in time. In the example of Fig.16, the value 0.88 is indicating that in this case it is considered that 88% of the amount of the investment per unit is subject to a decay in time and indirectly that the remaining 12% remains constant.

The parameter “annual rate [p.u.]:” must contain the annual rate at which the indexed portion falls. Eq.Error: no se encontró el origen de la referencia shows how the cost of the Technology Investment Unit C_t is calculated in the instant t .

$$C_t = C_{t_0}\left[f_V \left(\frac{1}{1+\alpha} \right)^{t-t_0} + (1-f_V) \right]$$

where the instant t_0 is the Start of the simulation or the date of Save of the Simulation (whichever is later). Both values are those specified in the SimSEE Room used. The value C_{t_0} is the present value of the fixed costs at the instant t_0 . The factor f_V corresponds to the portion of the cost that falls annually to the α rate of decay. In the eq.Error: no se encontró el origen de la referencia both t and t_0 are expressed in years.