Applications Included in GLEAM and Integration of New Ones

Version V1.0

Wilfried Jakob
KIT, Campus North, Institute for Automation and Applied Informatics (IAI)
Hermann-von-Helmholtz-Platz 1, 76344 Eggenstein-Leopoldshafen, Germany
email: wilfried.jakob@partner.kit.edu

Contents

Release Notes

Changes to V1.0:
1.

1 Introduction

This paper contains a short description of the standard applications contained in GLEAM and HyGLEAM and uses a scheduling application to describe the procedure for integrating further applications via the interface provided for this purpose in the `appl` package, see [1].

Applications are selected by the application class of a MOD file, see [2, Sect 2.1].

The experiments prepared for some applications use the supplied files of the `InitFiles` directory. When loading an experiment file, it is essential to ensure that the language variant of the program matches the language of the experiment file. Otherwise the loading of the files will be aborted with error messages like:

```
Error loading experiment!
*** 2 messages at state "ERROR":
ERROR (AUFG_tsk_data/225): Parameter "Genmodell" not found!
ERROR (MEN_f_load/2 ): Error in "EXP"-file!
```

The reason for this are the name-value pairs of the entries in the experiment and in the TSK file. Already the first entry in the EXP file "Genmodell" is not known to the English program variant, because it must be called "Gene model" here.

If the model file specified in the experiment file describes local hill climbers (LHCs) or heuristics and this file is loaded into GLEAM instead of HyGLEAM, a note is issued that GLEAM does not know heuristics or LHCs:

```
Experiment successfully loaded.
  1 messages at state "OK":
Message (HMOD_mod_data): Gene model expects not implemented local hill
climbers or heuristics
```

The warning indicates that only the functionality of GLEAM is currently available.

On the other hand, if a model file is loaded in HyGLEAM that does not specify LHCs, only the functionality of GLEAM is available and no warning is given. You will notice this if you try to define something other than a GLEAM job in the "Evo/Opt menu" or if you try to select a different optimization method than GLEAM in the "Opt Params" sub menu. The functionality of the "Evo/Opt" menu is described in detail in [3].

The experiment files given in chapter 2 and 3 can be found in the `testfield` directories of GLEAM and/or HyGLEAM. Since the memetic extension of GLEAM for the robot experiment did not give satisfactory results, there are no experiment files for HyGLEAM either.

2 Test Functions

Test functions, also called mathematical benchmark functions (MBFs), are identified by the application class ID `MathFkt` in the MOD and TSK files, see [2, Sect 2.1 and 3, Sect. 3.2]. Since some functions have certain regularities parallel to the axes of the coordinate system, they should be rotated in space. This can be done by using the program parameter “rotation angle in degrees”, see [3, Sect. 2.2 and 3.2]. Examples for a rotation by 30 degrees are shown in Fig 1 and 2.

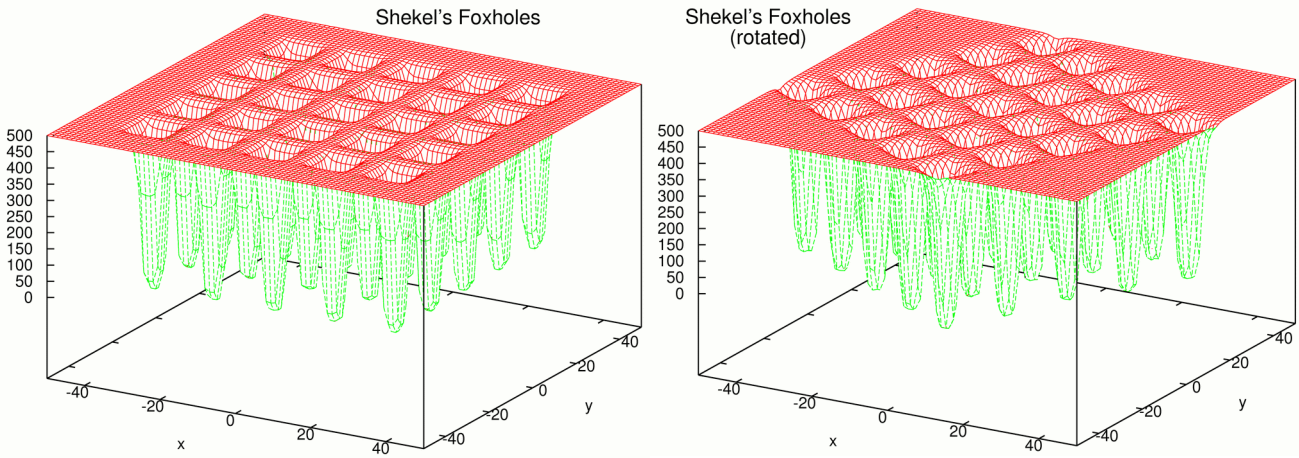


Fig. 1: Original and rotated version of the test function Shekel's Foxholes

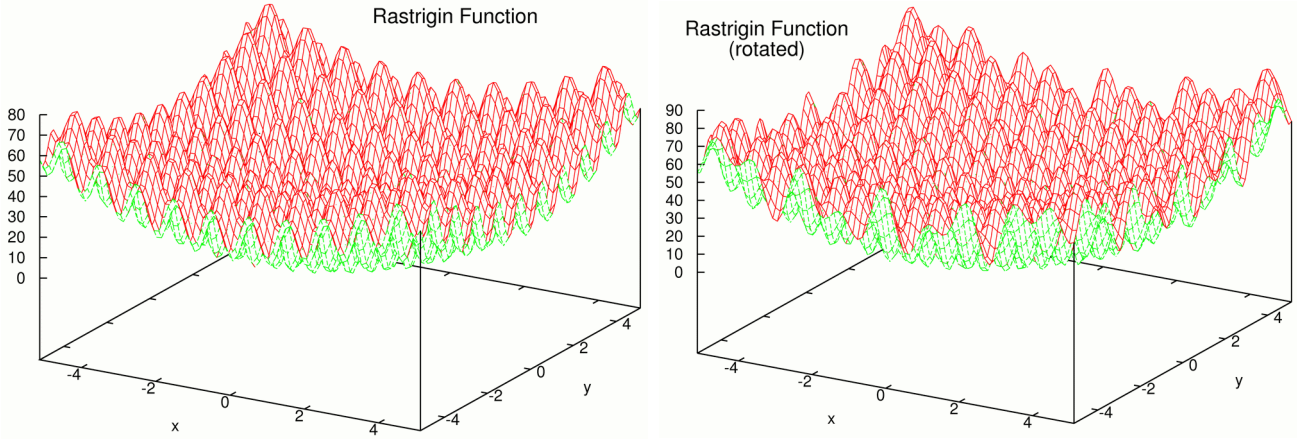


Fig. 2: Original and rotated version of the generalized Rastrigin function

Many function are taken from the *GENEsYs I* repository [4] and if so, the function numbers are given in brackets after the application ID. Some functions are also described in [5] with exact definitions. There the results of these and other test functions are also compared with EAs, which were created especially for the most exact solution of continuous optimization problems. It should be noted that GLEAM and HyGLEAM are not specifically designed for this purpose, but are more general with respect to the scope of application.

All functions are multi-modal except for the two spherical functions.

2.1 Sphere Function

The function can be found in [4, 5]. It is hard to some extend due to the large value range, which makes the function curve very flat near the optimum. The function is known to be easy for Evolution Strategy (ES), but hard for canonical Genetic Algorithms (GAs).

Application ID of the MOD file: `MBF-Sphere` (f1)

Value range: $-5.0 \cdot 10^{10} \leq x_i \leq 5.0 \cdot 10^{10}$

Dimensionality: 30, scalable

Optimum: $min(f(\vec{x})) = f(0,\ldots,0) = 0$

Prepared experiments:

1. `mbf_sphere_e.exp` based on decision variables in the range of $\pm 5.0 \cdot 10^6$
2. `mbf_sphere_lhc.exp` above experiment but suited for HyGLEAM

2.2 Weighted Sphere Function

The function can be found in [4]. It is hard to some extend due to the large value range, which makes the function curve very flat near the optimum. The function is known to be easy for ES, but hard for canonical GAs.

Application ID of the MOD file: `MBF-WSphere` (f15)

Value range: $-5.0 \cdot 10^{10} \leq x_i \leq 5.0 \cdot 10^{10}$

Dimensionality: 30, scalable

Optimum: $min(f(\vec{x})) = f(0,\ldots,0) = 0$

2.3 Generalized Rastrigin Function

The function can be found in [4, 5]. It should be rotated, which makes it much harder, see Fig. 2. Known to be hard for standard ES and local searchers.

Application ID of the MOD file: `MBF-Rast` (f7)

Value range: $-5.12 \leq x_i \leq 5.12$

Dimensionality: 5 if rotated and 20 otherwise, scalable

Optimum: $min(f(\vec{x})) = f(0,\ldots,0) = 0$

Prepared experiments:

1. `mbf_rast_rot_e.exp` rotated function with 20 decision variables and LHC
2. `mbf_rast_rot_lhc_clv.exp` above experiment but suited for the CLV
3. `mbf_rast_rot_lsv_clv.exp` above experiment for the German program variant

2.4 Function by Griewank

The function can be found in [4].

Application ID of the MOD file: `MBF-Griewank` (f22)

Value range: $-600 \leq x_i \leq 600$

Dimensionality: 10, scalable

Optimum: $min(f(\vec{x})) = f(0,\ldots,0) = 0$

Prepared experiments:

1. `mbf_griew_5par_clv_e.exp` 5 decision variables, with LHCs, suited for the CLV
2. `mbf_griew_5par_clv.exp` German version of the above file
3. `mbf_griew_5par_lhc.exp` 5 decision variables, with LHCs, prepared for AMMA run and CLV
4. `mbf_griew_5par_lsv.exp` German version of the above file

2.5 Function by Ackley

The function can be found in [4].

Application ID of the MOD file: `MBF-Ackley` (f9)

Value range: $-32.768 \leq x_i \leq 32.768$

Dimensionality: scalable

Optimum: $min(f(\vec{x})) = f(0,\ldots,0) = 0$

2.6 Function by Fletcher and Powell

The function can be found in [4, 5] and is of considerable complexity.

Application ID of the MOD file: `MBF-Fletcher` (f16)

Value range: $-\pi \leq x_i \leq \pi$

Dimensionality: scalable up to 30

Optimum: $min(f(\vec{x})) = 0$

2.7 Function by Bigg

The test function according to Bigg has an extremely flat course.

Application ID of the MOD file: `MBF-Bigg`

Value range: $0 \leq x_i \leq 20$

Dimensionality: 5

Optimum: $min(f(\vec{x})) = f(1,10,1,5,4) = 0$

2.8 Helical Valley Function by Fletcher and Powell

The function can be found in [6].

Application ID of the MOD file: `MBF-HValley`

Value range: $-100 \leq x_i \leq 100$

Dimensionality: 3

Optimum: $min(f(\vec{x})) = f(1,0,0) = 0$

2.9 Shekel's Foxholes

The function can be found in [4, 5]. This function should be rotated to achieve some additional difficulty, see Fig. 1. Known to be hard for standard Evolution Strategy and local searchers.

Application ID of the MOD file: `MBF-Foxholes` (f5)

Value range: $-500 \leq x_i \leq 500$ original: $-65.536 \leq x_i \leq 65.536$

Dimensionality: 2

Optimum: $min(f(\vec{x})) = f(-32,-32) \approx 0.998004$

Prepared experiments:

1. `mbf_fox_rot_e.exp` rotated function
2. `mbf_fox_rot.exp` rotated function (German version)
3. `mbf_fox_rot_lhc.exp` rotated function, with LHCs, prepared for an LHC run with the Rosenbrock procedure and CLV
4. `mbf_fox_rot_lhc.exp` rotated function, with LHCs, prepared for an AMMA run (German version)

2.10 Fractal Function by Weierstrass and Mandelbrot

The function can be found in [4, 5].

Application ID of the MOD file: `MBF-Fractal` (f13)

Value range: $-5.12 \le x_i \le 5.12$

Dimensionality: scalable

Optimum: unknown minimum, for some experiments a value of -0.05 was used.

2.11 Rechenberg's Real Multi-modal Test Problem

The function can be found in [7]. Unfortunately, the author did not give any information about the value range of the decision variables or about the optimum. It is only known that this is a maximum search and it is about finding the highest of 21 peaks.

Experiments showed that a value range of $-5 \le x_i \le 35$ is sufficient and that the best solution of a long AMMA run with a relatively large population of 60 individuals reached a value of 100.1932376621177. It turned out that the Rosenbrock procedure yields better results with less effort than the Complex algorithm and therefore quickly supersedes it. Since subsequent ASMA runs with the Rosenbrock method as meme already found this optimum reliably at a population size of 10 and a deme size of 6 and only required about 51,000 evaluations on average, this test function is probably not as difficult as the author assumed.

Application ID of the MOD file: `MBF-Rberg`

Prepared experiment:

`mbf_rberg.exp` with LHCs, prepared for an ASMA run with the Rosenbrock procedure

3 Robot Standard Application

The task of planning a collision-free path for a movement of an industrial robot was the first application GLEAM was developed for and tested with [8, 9]. Therefore, GLEAM was designed for simultaneous mixed-integer and combinatorial optimization. This field of application was further developed by C. Blume in particular and a number of applications with different robots and scenarios were developed [10, 11]. The path planning task implemented here is based on the Mitsubishi RV M1 robot and serves as a benchmark task. It is described in [12] and [11, Sect. 6.1]. Figure 3 shows the start and target position as well as the obstacles to be avoided. The target point is behind the small column. Since this first experiment was performed in 1990, the simple graphic corresponds to what was possible at that time on a PC with 20 MHz.

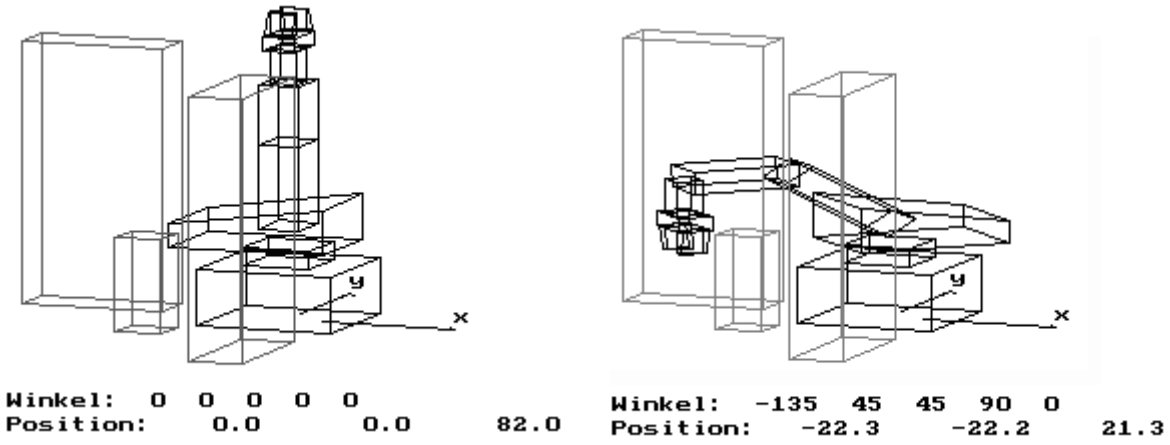


Fig. 3: Start and target position of a collision-free robot movement to be planned in the left and right part of the image

The evaluation of the generated solutions is based on the following criteria:

- target deviation: in % of a calculated maximum
- path deviation: in % of a calculated maximum. Since no straight line is possible for kinematic reasons, a deviation of 10% is selected as the target value.
- crash residual path: This criterion is implemented as a penalty function, since it is an undesirable result. In order to show the evolutionary search the way out of the realm of unsuitable solutions due to a collision, crashes with shorter residual distances are penalized less than those with longer residual distances. Only if the movement is free of collisions with obstacles and the robot itself, the fitness is not reduced.
- motion duration: This and the next two criteria deal with related goals: fast movements and few instructions.
- action chain length: Because of the actions for time control there may be more actions than derived instructions for a robot control.
- no of exec. instr.: Length of resulting list of instructions for a robot control
- power consumption: Calculation of a hypothetical value based on accelerations and velocities. It was not considered in the benchmark task.

Since the target values of the criteria were chosen to be attainable and result in a movement with good target achievement with the least possible path deviation, short movement times and instruction lists, the maximum fitness can be achieved in this benchmark task.

The robot task is configured in more detail by the TSK file. The following six application-specific program parameters assigned to this application serve this purpose, see [3], Sect. 2.2: `with Mitsubishi R500`, `with collision test`, `with end stop test`, `with depreciation on stop/crash`, `number of axes`, and `simulator cycle time [sec]`. The lines contained in the TSK file for the robot application define the following settings for the sample application:

```
with Mitsubishi R500                = 1
with collision test                 = 1
with end stop test                  = 1
with depreciation on stop/crash     = 1
number of axes                      =    5
simulator cycle time [sec]          =      0.1
```

Start and target are specified after the standard part of a TSK file. The application-specific part is introduced by a special comment starting with "#!" and contains the position and orientation of the gripper at the start and the target position in frame notation, see [13]. It looks as follows:

```
#! ------- start- and target-data for the robot application: --------
 0 0 0 0
 0 0 0 0
 0 0 0 0
 0 0 0 0

 -2.35619415 0.785398 0.785398 1.5707961
 0 0 0 0
 0 0 0 0
 0 0 0 0
```

It has been found that the two standard LHCs of HyGLEAM do not lead to any improvements compared to pure EA for this test task.

Prepared experiments:

1. `lesak_e.exp` robot experiment as described above
2. `lesak.exp` robot experiment as described above (German version)

4 Additional Scheduling Application

The application is described in [5] as follows: The scheduling and resource optimization task is taken from chemical industry [14] and deals with 87 batches with varying numbers of workers being required during the different phases of each batch. Figure 4 shows a diagram of varying manpower demand in the white chart. The batch starts with four workers, which are reduced to two and then to one and so on. The objective of scheduling these batches means the best reduction of the overall production time (makespan) as well as the maximum number of workers per shift (human resource). Restrictions like due dates of batches, necessary pre-products from other batches, and the availability of shared equipment also must be observed. A further complicating factor is that the required pre-products sometimes have to be produced in several batches, creating process chains with quantitative dependencies.

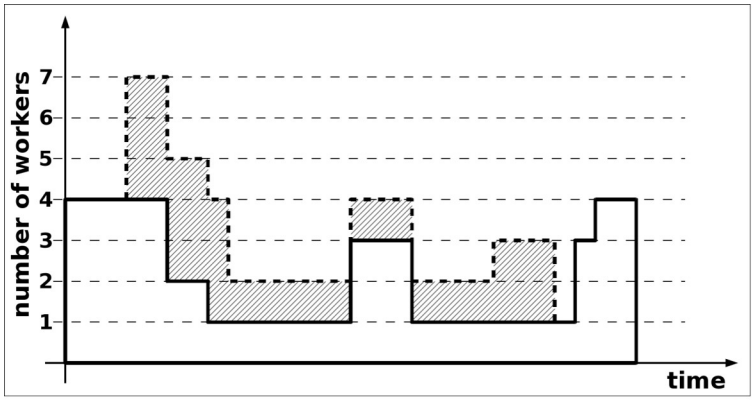
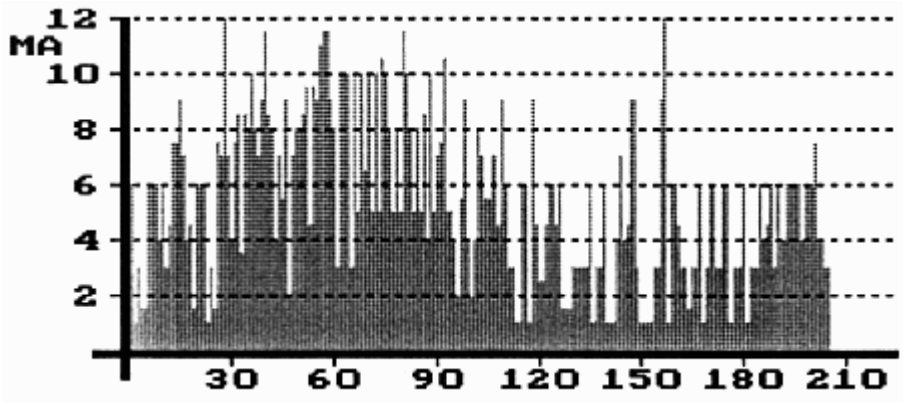


Fig. 4: Cumulated manpower demand of two parallel executed batches. The second batch marked gray starts with three workers, continues with one, and ends with two. If it is started later, a peak of five workers instead of seven is achieved

Figure 4 shows an example of the summed up work force demand of two batches. The figure underlines that appropriate starting times of the batches can significantly reduce or increase the peak of workers required within a shift. Thus, it is not sufficient to generate an appropriate sequence of processing the batches. Instead of this common solution to scheduling, suitable starting times must be determined and the relevance of the sequence of genes representing the batches is reduced, as it is used for solving allocation conflicts only. Therefore, a gene consists of its batch identifier and the starting time coded as an integer.

The optimum is unknown in this case. Figure 5 shows a schedule that almost exploits the foreseen maximum 210 8-hour shifts (1680 hours) of the original manual planning practice. Note that the image shows the shift peaks and that each shift may still contain several unused hours. This makes it clear that there should be considerable potential for optimization compared to manual planning.

Fig. 5: Starting situation: Manually created schedule that almost makes use of the available time window



Depending on whether more emphasis is placed on reducing production time or peak demand for employees per shift, the results will vary. Figure 6 shows a schedule in which emphasis is placed exclusively on a short processing time, while Figure 7 shows that a relevant reduction of the maximum number of employees is possible if a slightly longer processing time is allowed. This is achieved by an appropriate weighting of the criteria in the evaluation.

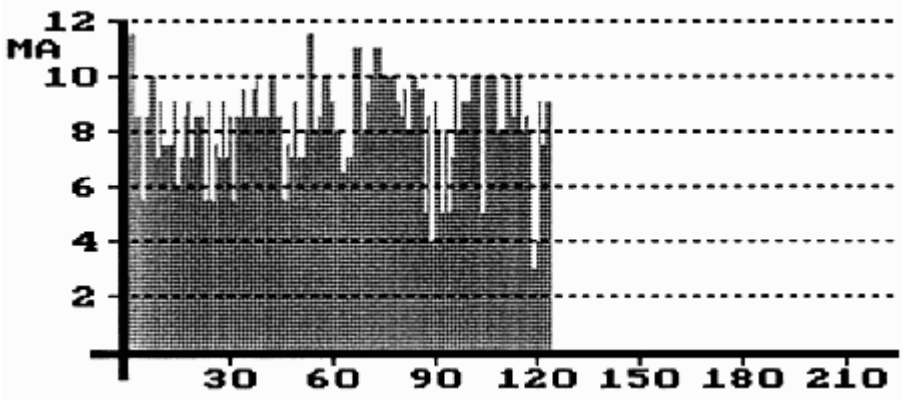


Fig. 6: Time-optimized planning achieving a reduction of 41 % savings in employee hours

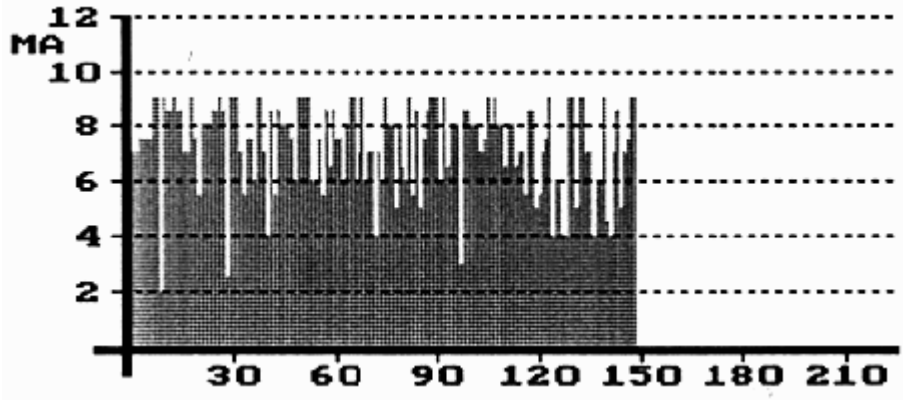


Fig. 7: Time and employee optimized planning achieving a reduction of 47 % savings in employee hours

The evaluation of the provided example is based on the objective of time and employee optimized scheduling.

4.1 Notes on the Evaluation of the Scheduling Application

Based on the task definition, there are four evaluation criteria:

1. The total processing time (TotalDuration)
2. The largest number of employees per shift, also called maximum shift peak (MaxShiftPeak)
3. Delay compared to the latest completion time (DeadlineDelay)

4. Non-compliance with the process chains and resulting reduced production (UnderProduction)

The last two criteria can be implemented as penalty functions, since they represent undesirable results. With this evaluation, however, only the maximum value of the shift peaks is considered. For example, a reduction in the number of shift peaks with a value below the maximum peak does not result in an improvement in fitness, although this is a necessary intermediate step to reduce the peak value at all. However, the evolutionary search process is solely guided by the evaluation and therefore the previous criteria are not sufficient to efficiently reduce the shift peaks. These should be reduced to a given maximum value and therefore it makes sense to record all exceedings of this value together with their duration and to evaluate them additionally as an auxiliary criterion. This is achieved by the "ShiftPeakExcess" criterion, which records all overruns of the value specified by the program parameter "Max. employee peak per shift" (see TSK file) and their duration. The complete list of criteria is as follows:

```
                          Assessment Criteria:

 Nr. Criterion             Prio.     Fitness   Fulfillment     Penalty Fmax
                           Class       Share  W.Fitn   Value PrioClass Cont
  1: TotalDuration       :   1         70000   26365    1360      1       yes
  2: MaxShiftPeak        :   2          8000    8000       9      1       yes
  3: ShiftPeakExcess     :   1         21998    2444      60   inactive   yes
  4: DeadlineDelay       :   1             1       1       0      1       yes
  5: UnderProduction     :   1             1       1       0      1       yes
  6: OffTime             :inactive  (      0)      0      72   inactive   yes
 ------------------------------------------------------------------------------
                                     100000
                                     ======
```

It has been observed that it happens relatively often that hours are planned without any allocation. Such "breaks" could best be avoided by genotypic repair. Instead, they were evaluated in the early 1990s, hence the criterion "off-time". Better than an evaluation, however, is a phenotypic repair, which simply extracts the break times.

4.2 Integration of a New Application Using OPAL/V as an Example

The optimization of a new application with GLEAM requires at least the integration of a corresponding evaluation. This can be done by new routines for coupling an external simulator or by the direct implementation of chromosome interpretation and evaluation. The latter is likely to be common for scheduling tasks, since scheduling operations are coded in the chromosome and lead to the filling of allocation matrices for the resources to be allocated. This is also the case in this task and a matrix for the process plants to be occupied over time is needed. In addition, volume data for the various intermediate and end products. Since it is not about a concrete resource planning for the employees but only a summary one, a vector is sufficient to indicate per time unit how many employees have been assigned. From all this, the evaluation criteria can be derived after the processing of a chromosome. Whichever of the two alternatives comes into question, Module `appl_sim.c` is intended for the implementation of this task.

It should be noted that knowledge of the software documentation [1] is strongly recommended for further reading.

For a new application, in addition to the evaluation (see module `appl_sim.c`), it can also be useful to design the following functions to be application-specific:

1. Module `appl_fio.c` for application-specific I/O. For the task at hand, the production processes to be planned are thus read in, each with employee requirement profile, required equipment, produced quantity and any required quantity of pre-product.
2. Module `appl_go.c` for application-specific genetic operators, if required.

3. Module `appl_rep.c` for application-specific genetic repair, if required. Here the above mentioned genetic repair to avoid “breaks” without any occupancy could be realized.
4. Module `appl_gen.c` for application-specific generation of chromosomes, if required. Here, e.g. one or more heuristics for the generation of "good" chromosomes, which serve for the creation of a part of the initial population, would be accommodated.

For the present task the functions of points 2 to 4 are not required.

4.2.1 Registering a New Application

The first step in integrating a new application is to make it known to the system. For this purpose, a new application class ID is defined in the file `appl.h`, in our case by the following line:

```
#define OPAL_KENNUNG     "OPAL/V"   /* application class ID in the MOD file */
```

Next a code is required for that ID and in our case it is defined by

```
#define OPAL_APPL        301        /* application ID of "add_appl_code"    */
```

Note that all `…_APPL` codes must be different. The function `read_add_appl()` of module `appl_if.c` is called, if no standard application class ID is detected in the MOD file. The function must be extended accordingly and the new code must be assigned to variable `add_appl_code`, see:

```
if (strcmp (appl_id_str, OPAL_KENNUNG) == 0)
   add_appl_code = OPAL_APPL;                                  /* OPAL */
else
```

Next, the `set_appl_texts` function must be extended and appropriate settings must be made, see:

```
case OPAL_APPL:            /* here the simplified OPAL/V implementation */
strcpy (appl_name,      OPAL_NAME); /* see appl_gb.h file               */
strcpy (appl_text,      OPAL_TEXT); /* see appl_gb.h file.No longer used*/
strcpy (extSimSubDir,   "");        /* sub directory for external simu. */  A
strcpy (opt_param_name, "");        /* for display of decision variables*/  B
mit_ext_simu       = FALSE;         /* T: external simulator is used    */
displayableParams  = FALSE;         /* T: displayable decision variables*/  B
extractChrParams   = FALSE;         /* T: extract dec.vars for ext. simu*/  C
sim_up             = TRUE;          /* T: "simulator" ready to be used  */  D
std_simu_interface = TRUE;          /* T: standard simulator interface  */  E
always_sim         = TRUE;          /* T: simu regardless of chr.-flag  */
*std_ak_anwndg     = TRUE;          /* T: standard chromosome generation*/  F
*std_segm_gen_appl = TRUE;          /* T: Standard chr.gen. and modific.*/  G
mit_gen_rep_par    = FALSE;         /* T: uses gen.repair control param */  H
mitOptionPhaenoRep = TRUE;          /* T: with opt.of phenotypic repair */  I
mitPhaenoRepair    = TRUE;          /* T: phenotypic repair active      */  I
break; /* OPAL_APPL */
```

Remarks:

A: Used by some interfaces to external simulators. Not in use at present.

B: If it is useful to display the decision variables with the names stored in the MOD file, `displayableParams` must be set to TRUE and a heading of the variable list can be specified with `opt_param_name`. This is usually not meaningful for scheduling tasks.

C: External simulators may be called with the chromosome itself or with the decision variables extracted.

D: Usually TRUE. External simulator interfaces will set this variable to FALSE, if their connection is lost or cannot be established.

E: Usually TRUE. Is set to FALSE for the internal robot application only.

F: Controls together with `std_segm_gen_appl` the standard generation and modification of chromosomes, see Sect. 4.2.7.

G: Controls together with `std_ak_anwndg` the standard generation and modification of chromosomes, see Sect. 4.2.7.

H: A possible genetic repair of one chromosome can be controlled by a parameter. This can be activated by setting this variable. It is always tried to perform a genetic repair of one chromosome by calling `repair_1_chain()` of the module `segm_gen.c` from package `hmod`. There it is decided what has to happen. In the present version only the robot application uses a genetic repair and in case of an additional application the function `add_appl_repair_1ch()` from module `appl_rep.c` is called, see also Sect. 4.2.6.

I: The variable `withOptionPhaenoRep` controls whether a phenotypic repair of one chromosome is available and can be activated, whereas `withPhaenoRepair` activates or deactivates that repair. Since phenotypic repair typically occurs during interpretation and/or simulation of a chromosome, this function is part of the SW included in the `appl_sim.c` module. In the case of the OPAL/V application this is done in the routine `bewerten()` of the module `opal_bew.c` from the additional Package `opal`, see the `appl_packages` directory. If phenotypic repair is possible and is implemented during interpretation and/or simulation, it may be useful to set `always_sim` to TRUE, e.g., to issue messages about this. This is especially true if data is created during interpretation and/or simulation for a later processing of results.

4.2.2 Module Overview of the OPAL/V Application

Five of the six modules of the appl package can be used as a framework for an application-oriented new implementation. The sixth, the interface module `appl-if.c`, has to be extended and adapted as described in Sect. 4.2.1. The modules of a new application are located in a new directory to be created below the `appl_packages` directory, which is a sub-directory of the `sources` directory, see also Sec. 2.5 in [1]. In our case this is the `opal` directory. The five modules are:

1. `appl_sim.c`: This module is replaced by the `opal_bew.c` module.
2. `appl_fio.c`: This module is replaced by the `fio_if.c` and the `opal_io.c` module. The first one contains all routines from the `appl_fio.c` module, while the second one implements the reading of the VL file. The VL file contains data describing the batches, like manpower profiles, quantities of necessary pre-products etc. and all other data of the task at hand for the initialization of the evaluation.
3. `appl_go`: Not used by the opal implementation. It contains functions for application dependent mutations and crossover operators.
4. `appl_rep`: Not used by the opal implementation. It contains functions for application dependent genetic repair of one or two chromosomes.
5. `appl_gen`: Not used by the opal implementation. It contains functions for application dependent generation of chromosomes and gene sequences.

The three modules not used by the application at hand remain unchanged and are translated and linked to the program as described in Sect. 4.2.8.

4.2.3 Chromosome Evaluation and Phenotypic Repair

The `opal_bew.c` module replacing the `appl_sim.c` module implements the `do_appl_sim`() function, which includes the optional phenotypic repair of a chromosome.

4.2.4 File-IO for Evaluation and Utilization of Results

The modules `fio_if.c` and the `opal_io.c` replace the `appl_fio.c` module and implement reading the VL file, which initializes the evaluation implemented by the `opal_bew.c` module. For more than a pure test implementation the output of the scheduling result would be required. The routine `add_appl_save_erg()` is available for this purpose. In the interactive version the menu item "`SaveSimRes`" writes result data as shown by the "ShowEval" menu item into a RES text file and calls that routine. The same happens in the CLV, if the program parameter "with result simulation" is set in the TSK file. In case of the OPAL/V test case nothing is implemented by `add_appl_save_erg()` and an error message is issued instead:

```
------------------------ Simulator --------------------------
1: SimuStart    2: ShowEval     3: PlotEval     4: SaveSimRes
                                7: State
9: Log.on/off                   B: DelSimVal    C: MemEvalUpd
D: Close
-------------------------------------------------------------
Select menu item: 4

          Save of Recent Simulation Results

"res"-FileSpec (without ext) [kompakt_lhc_rep]: tmp
No save done!
  1 messages at state "OK":
Message (OPAL_fio_if): Function not yet implemented!
```

In contrast to the message, a file is created as described above, but no further result data are written. In a real application this would be e.g. the start times of the batches, the occupancy data of the equipment to be used and an employee requirement plan.

4.2.5 Application-specific Genetic Operators

Application-specific genetic operators are implemented by the two routines `do_appl_mut()` and `do_appl_xo()` of module `appl_go.c`. To distinguish them from the standard operators, they get negative operator codes, see [15] and there in particular section 5. If negative codes are given in the EVO file, `do_appl_mut()` or `do_appl_xo()` is called, depending on whether the operator shall produce one or two offspring.

In the case of crossover or recombination, it can happen that one offspring of fixed-length chromosomes is missing genes that are too many for the other. For standard operators, the routine `repair_2_aks()` of the module `evo_gsw.c` from the evo package is called for this purpose. For application-specific operators, however, `add_repair_2_chs()` is called from module `appl_rep.c`, see also the next section.

4.2.6 Application-specific Genetic Repair

If none of the standard applications is specified in the MOD file and consequently the variable `appl_code` is set to `ADD_APPL` by function `read_add_Appl()`, a call to `add_appl_repair_1ch()` or `add_repair_2_chs()` from module `appl_rep.c` is made after the execution of the corresponding genetic operators, regardless of whether they are standard or application-specific. If the same repair is to be performed on chromosomes of fixed length as with the standard crossover or recombination operators, in `add_repair_2_chs()` simply `repair_2_aks()` of the module `segm_gen.c` from the `evo` package can be called, as is the case with the OPAL/V application.

If necessary, the additional parameter `gen_rep_par` can be used, provided that `mit_gen_rep_par` is set to TRUE, see Section 4.2.1, remark H. In this case, the strings `gen_rep_par_query` and `gen_rep_par_anzg` are to be set appropriately. The texts of the `GORBA_…`-applications may serve as an example, see module `appl_if.c` and file `appl_gb.h`. Due to the last usage of an application-specific genetic repair the parameter is used as an execution rate for repairing a chromosome. There is a program parameter assigned to the variable `gen_rep_par`, which is accordingly called "genetic repair rate [%]", see [3, Sect. 2.2]. When loading, its value is divided by 100, since it is a percentage value. This is done in the routine `processReadTskData()` of the module `tsk_data.c` from Package `aufg`.

4.2.7 Application-specific Generation of Chromosomes

The preparatory measures described below for the integration of an application-specific generation of chromosomes or gene sequences for mutations have not been used in any application so far. Instead, application-specific methods for chromosome generation have been implemented outside of GLEAM, e.g. by suitable heuristics or similar, and the results have been written to a chromosome file. This file was then read at the start of GLEAM or HyGLEAM, and the contents was used to generate the initial population. On the basis of experience so far, this seems to be the more flexible and simpler approach.

The application-specific generation of chromosomes or gene sequences is controlled by the two Boolean variables `std_ak_appl` and `std_segm_gen_appl`.

If `std_ak_appl` is set to FALSE `add_anw_ch_gen()` from module `appl_gen.c` is called. It replaces the standard generation of chromosomes. If `std_segm_gen_appl` and `std_ak_appl` are both FALSE, `add_gen_act_seq()` from module `appl_gen.c` is called. It is called in function `gen_act_seq()` (module `segm_gen.c` from package `hmod`) and replaces the standard generation of gene sequences performed by `do_gen_act_seq()`, also from module `segm_gen.c`. The function `do_gen_act_seq()` is called by some mutations.

However, both variables are processed elsewhere in the module `segm_gen.c` and this currently leads to (fatal) errors there. The following functions are affected: `parameterize()`, `calc_par_anz()`, `modify_param()` und `kettenlaenge()`. These occurrences need an appropriate program extension. Both variables are not processed outside the module `segm_gen.c`.

4.2.8 Effects of a New Application on the make-Files

Some of the modules of the `appl` package will be replaced by the new ones of the new application and thus the definition of the set of sources of the `appl` package to be compiled and linked as described by the `APPL_OBJ` variable of the `appl_inc.mak` file cannot be used. Nevertheless this file is included by `common_def.inc` because it also describes the dependencies of the modules and this is needed at least for the module `appl_if.c` which is always to be used.

In case of the OPAL/V application `appl_sim.c` and `appl_fio.c` of the `appl` package are replaced by the three modules of the new `opal` package, see Sect. 4.2.2. For the latter the include file `opal_inc.mak` lists the modules and describes the dependencies:

```
#-------------------------------------------------------------------------
#      Definition of Objects:
#-------------------------------------------------------------------------
OPALV_OBJ = $(OPALV)/opal_io.o  $(OPALV)/fio_if.o  $(OPALV)/opal_bew.o

#-------------------------------------------------------------------------
OPALV_TXT = $(OPALV)/opal_d.h    $(OPALV)/opal_gb.h
OPALV_ALL = $(OPALV)/opalInt.h   $(CTIO)/ctio.h      $(FBHM_INC)        \
            $(CHAINDEF)          $(HMOD)/hmod.h
```

```
#-------------------------------------------------------------------------
#      Description of the Dependencies:
#-------------------------------------------------------------------------
$(OPALV)/opal_io.o : $(OPALV_ALL)      $(SYS)/sys.h
$(OPALV)/fio_if.o  : $(OPALV_ALL)      $(GLOB_TXT)       $(SYS)/sys.h       \
                     $(APPL)/appl.h
$(OPALV)/opal_bew.o: $(OPALV_ALL)      $(OPALV_TXT)      $(SYS)/sys.h       \
                     $(APPL)/appl.h    $(AUFG)/aufg.h    $(SIMU)/simu.h
```

Since the integration of the new application OPAL is the same in both the interactive and the command line version, it is done in the include file `glob_inc.mak` which is common to both versions. The affected areas are highlighted in yellow:

```
#--------------------- Specific Directory Declarations: ---------------------
OPALV = $(ALLG)/appl_packages/opal
         . . .
include $(OPALV)/opal_inc.mak

#-------------------- Integration of the Remaining APPL Package: --------------
APPL_OPALV_OBJ = $(APPL)/appl_if.o   $(APPL)/appl_go.o   $(APPL)/appl_rep.o   \
                 $(APPL)/appl_gen.o

#-------------------------------------------------------------------------------
#      Definition of the Basic Object Groups:
#-------------------------------------------------------------------------------
BASIC_OBJ     = $(SYS_OBJ)  $(FBHM_OBJ)  $(LGSW_OBJ) $(CHIO_OBJ) $(HMOD_OBJ)  \
                $(ROCO_OBJ) $(OPALV_OBJ) $(APPL_OPALV_OBJ)
TUI_BASIC_OBJ = $(CTIO_TUI) $(BEW_TUI) $(AUFG_TUI) $(CHED_TUI) $(LSKP_TUI)    \
                $(SIMU_TUI) $(MEN_TUI)
CLV_BASIC_OBJ = $(CTIO_CLV) $(BEW_CLV) $(AUFG_CLV)             $(LSKP_CLV)    \
                $(SIMU_CLV) $(MEN_CLV)

# -------------------------- For OPAL/V-Applications: -------------------------
TUI_OBJ    = $(BASIC_OBJ) $(TUI_BASIC_OBJ)
CLV_OBJ    = $(BASIC_OBJ) $(CLV_BASIC_OBJ)
```

4.2.9 Integration of Application-specific Files

The example application uses an initialization file that shows the batches to be scheduled and their properties such as process chains, employee profiles and latest completion times. In another application, the data contained in this file could also be distributed, for example, to two files, one containing the master data of products such as information about suitable equipment, employee profiles and possible process chains, while the second contains order-related data such as the quantity of current orders with completion times and priorities for rush orders. It might also be necessary to output the results in the form of shift, work and/or occupancy plans. In the explanations on the integration of one initialization file, the two outlined extensions are also dealt with afterwards.

Due to the application-specific nature of these files, their description is included in the gene model file, see also [2], Sect. 2.1. The standard MOD file header contains the following data:

```
************************ Gene model for GLEAM/AE ***************************
GLEAM/AE   OPAL/V                 05.03.2001              revised: 18.8.2020
1  2  2         gen_len_mode (1,2)  gen_akt_mode (1,2,3)  gen_segm_vert (1,2)
87  87   4  6   min_ketten_len  max_ketten_len  min_abschn_len  abschn_delta
87              number of gene types
 6    0.005     akt_roh_erg_werte of (Ext)Simu    small_change_frac
 1              number of additional files
0 vl  Processlist  ProcList
```

For the interpretation of the first 6 lines, which describe the OPAL/V application with its chromosome type 2 (fixed length and relevant gene order) and 87 genes, see [2], Sect. 2.1. The part concerning the additional files is marked yellow. One additional file is specified here and the following

line describes the file in detail. It is a read-only file (`ioCode` is 0) with the file extension `.vl`. Its name is“ProcessList” and the corresponding menu item is “ProcList”.

Assuming that the three previously sketched files are to be integrated, it could look like this:

```
 3              number of additional files
0 md   MasterDataList   MasterData
0 odl  OrderDataList    OrderData
2 rs   ScheduleResults  Schedules
```

Accordingly, the extended form of the EXP file is to be used. In case of the VL file the part for the application file looks like this, see also [3], Sect. 3.

```
#! --------------- optional list of application specific files ------------
opal_standard.vl        # application specific process list
```

In case of several files, they must appear in the EXP file in the same order as they are specified in the MOD file.

4.2.10 Remarks to the Initialization Files of OPAL/V

The task of OPAL/V differs from many other scheduling tasks in that a scheduling operation with OPAL/V does not try to schedule the job at the earliest possible time, but rather that suitable start times are sought. These can be specified by the genes either directly or as a delay to the earliest possible start time. In the first case, the gene sequence then only plays a role in case of occupancy conflicts, where the system waits until the desired facility becomes free long enough at the earliest possible time. In the second case, however, the sequence is of much greater importance. In the example here the first alternative was chosen.

Based on the task specifications, a time frame of 1680 hours is considered, which would mean start times in the range 0 to 1679. In fact, however, the start times in the gene model are given with much more restricted ranges, since minimum times for possible pre-products are considered for the limits as well as latest completion times and the like. It should be noted at this point that it is always useful to check in scheduling tasks whether the time requirements can be met at all by determining, for example, the time required for the critical paths and what effect this has, for example, on reasonable start time intervals.

Finally, when looking at the MOD files contained in the distribution, it is noticeable that the gene model with the gene parameter `VerfIdx` contains a parameter in which lower and upper limits are identical. If such fixed parameters are at the end of a parameter list, they are ignored by evolution. With this trick, a fixed reference to the corresponding master data of the product is included in the genes per batch. The alternative would have been a table that maps the batch number identical with the gene ID to the corresponding product.

Both local search methods contained in HyGLEAM are basically suitable to improve the integer start times, whereby the values of both methods are to be rounded. It has been shown, however, that firstly the Rosenbrock method produces significantly better results and secondly the step size of the method should also be set adaptively. Therefore `hyGleamOpalv` contains a correspondingly extended implementation of the Rosenbrock method. This is reflected in the LHC lines part of the MOD file by an additional parameter `StepSize` for the Rosenbrock method, which has to be adapted.

The evaluation described in Sect. 4.1 must be newly created with the interactive program version `hyGleamOpalv`. The corresponding procedure is described in the User Manual [3] in sections 3.3 and especially 4.2.6.

In addition, an EVO file must be created to configure the appropriate genetic operators. See the User Manual [3], segment 3.4 and especially the Evo-File documentation [15]. For chromosomes with meaningful gene sequence and fixed length all standard standard crossover or recombination

operators of GLEAM and the mutations shown in Table 1 are suitable. They are a subset of the mutations described in [15].

Operator-code	Short Description	Designator in an EVO file
0	Random redetermination of a randomly selected gene parameter	par_change_new
1	Change of one or more randomly selected parameter values of a gene	par_change_rel
2	Randomly select all parameters of a randomly selected gene	act_new_param
3	Change of one or more parameters of all genes of a randomly selected segment	segm_change_rel
4	All parameters of all genes of a randomly selected segment are randomly chosen	segm_new_param
9	Shifting a randomly determined gene to a randomly determined position	act_translocation
13	Shifting a randomly determined segment behind another randomly determined segment	segm_transl
14	Arranging the genes of a randomly selected segment in reverse order	segm_inversion
15	A randomly selected segment is merged with the following one	integr_nachb_segm
16	Shifting a randomly determined segment before another randomly determined segment and merging the two	integr_segm
17	A randomly determined segment border is shifted to the right or left by a randomly determined number of genes up to the next segment border at maximum	schieb_segm_grenz
18	A randomly determined segment is divided at a randomly determined location, provided it has at least two genes	teile_segm
20	Small change in one or more randomly selected parameter values of a gene	par_change_small
21	Small modification of one or more parameters of all genes of a randomly selected segment	segm_change_small

Table 1: Suitable Subset of Standard Mutations of GLEAM for the OPAL/V Task

Prepared experiments:

1. `kompakt_norep_e.exp` no phenotypic repair, prepared for a CLV run
2. `kompakt_norep.exp` German version of the above file
3. `kompakt_rep_e.exp` with phenotypic repair, prepared for a CLV run
4. `kompakt_rep.exp` German version of the above file
5. `kompakt_lhc_rep.exp` with phenotypic repair and LHCs, prepared for an SMA run using the Rosenbrock procedure and the CLV
6. `kompakt_lsv_rep.exp` German version of the above file

5 Literature

[1] W. Jakob: *GLEAM and HyGLEAM - Software-Documentation*. Technical Paper, V5.0, IAI, 2020.
See `package-_and_SW-Docu_v5.0.pdf`

[2] W. Jakob: *MOD File Documentation*. Technical Paper, V1.3, IAI, 2020.
See `MOD-File-Docu_V1.3.pdf`

[3] W. Jakob: *HyGLEAM - Hybrid General Purpose Evolutionary Algorithm and Method: User Manual.* Technical Paper, V1.0, IAI, 2021.
See `HyGLEAM-Manual_V1.0.pdf`

[4] T. Bäck: *GENEsYs 1.0.* 1992. http://www.cs.cmu.edu/afs/cs/project/ai-repository/ai/areas/genetic/ga/systems/genesys/0.html Accessed: 19 Oct. 2020

[5] W. Jakob: *A general cost-benefit-based adaptation framework for multimeme algorithms.* Memetic Computing, 2(2010) 201-18 doi: 10.1007/s12293-010-0040-9
A preprint is included in the `Literature` sub directory of the documentation directory.

[6] R. Fletcher, M.J.D. Powell: *A Rapidly Convergent Descent Method for Minimizatio*n. The Computer Journal, 6(2), 1963, pp.163-168 doi: 10.1093/comjnl/6.2.163

[7] I. Rechenberg: *Evolutionsstrategie'94*. (in German) Frommann-Holzboog, Stuttgart-Bad Cannstatt, 1994. pp.154.

[8] C. Blume: *GLEAM – A System for Simulated "Intuitive Learning"*. In: H.-P. Schwefel, R. Männer (eds): Conf. Proc. of Parallel Problem Solving from Nature. PPSN 1990. LNCS 496, Springer, Berlin, Heidelberg, pp. 48-54. doi: 10.1007/BFb0029730

[9] C. Blume, W. Jakob: *GLEAM - An Evolutionary Algorithm for Planning and Control Based on Evolution Strategy*. In: E. Cantù-Paz (ed.): Conf. Proc. of Genetic and Evolutionary Computation Conference (GECCO 2002), New York, Vol. Late Breaking Papers (LBP), 2002, pp.31-38
A preprint is included in the `Literature` sub directory of the documentation directory.

[10] C. Blume: *Optimized Collision Free Robot Move Statement Generation by the Evolutionary Software GLEAM*. In: S. Cagnoni (eds): Real-World Applications of Evolutionary Computing. EvoWorkshops 2000. LNCS 1803. Springer, Berlin, Heidelberg, pp. 330-341. doi: 10.1007/3-540-45561-2_32

[11] C. Blume, W. Jakob: *GLEAM - General Learning Evolutionary Algorithm and Method: Ein Evolutionärer Algorithmus und seine Anwendungen*. In German. KIT Scientific Publishing, Karlsruhe, 2009. doi: 10.5445/KSP/1000013553

[12] W. Jakob, M. Gorges-Schleuter, C. Blume: *Application of Genetic Algorithms to Task Planning and Learning*. In: R. Männer, B. Manderick (eds.): Conf. Proc. of Parallel Problem Solving from Nature 2 (PPSN-II), Brussels, Belgium, Elsevier, Amsterdam, 1992, pp.293-302

[13] C. Blume, W. Jakob, J. Favaro: *PASRO - Pascal and C for Robots*. Springer, Berlin, 1987.

[14] C. Blume, M. Gerbe: *Deutliche Senkung der Produktionskosten durch Optimierung des Ressourceneinsatzes*. (in German) atp 36(5): 25-29, 1994, Oldenbourg Verlag, München.

[15] W. Jakob: *EVO File Documentation*. Technical Paper, V1.0, IAI, 2020.
See `EVO-File-Docu_V1.0.pdf`