MIXED-CRITICALITY SCHEDULING USER GUIDE

SET OF ALGORITHMS FOR SOLVING MIXED-CRITICALITY SCHDULING

TABLE OF CONTENTS

PURPOSE OF THIS DOCUMENT

The next few pages presents a brief documentation of the MCScheduling application in form of a user guide. For detailed documentation of the application, please, refer to the */doc/index.html* and the article *On Non-Preemptive Mixed-Criticality Scheduling*.

MCSCHDULING GRAPHICAL USER INTERFACE

This section describes the graphical user interface of the MCScheduling application. The figure 1 shows how the main window looks like. The GUI is a collection of several tab pages that may be (in order of appearance) described as

1. *Test Instances* – This page contains a set of few methods for random generating of mixed-criticality instances.
2. *Mixed Integer Programming* – The page dedicated to a mixed integer linear programming solver (MIP solver).
3. *GA Configuration* – This page contains a configuration settings for a genetic algorithm that may be used for solving the given MC instances.
4. *Genetic Algorithm* – The page dedicated to a genetic algorithm solver (GA solver) using the configuration set up in the previous page.
5. *Simulated Annealing* - The page dedicated to a simulated algorithm solver (SA solver). It also contains a part for the configuration of the algorithm unlike in the case of GA solver it all fits on a single tab page.
6. *Clairvoyant EDF* - The page dedicated to a clairvoyant earliest deadline first algorithm solver (CEDF solver).
7. *DP* - The page dedicated to a dynamic programming algorithm solver (DP solver).

In the following subsections each of the pages is described in detail with an accompanying screenshot to make it more clear.

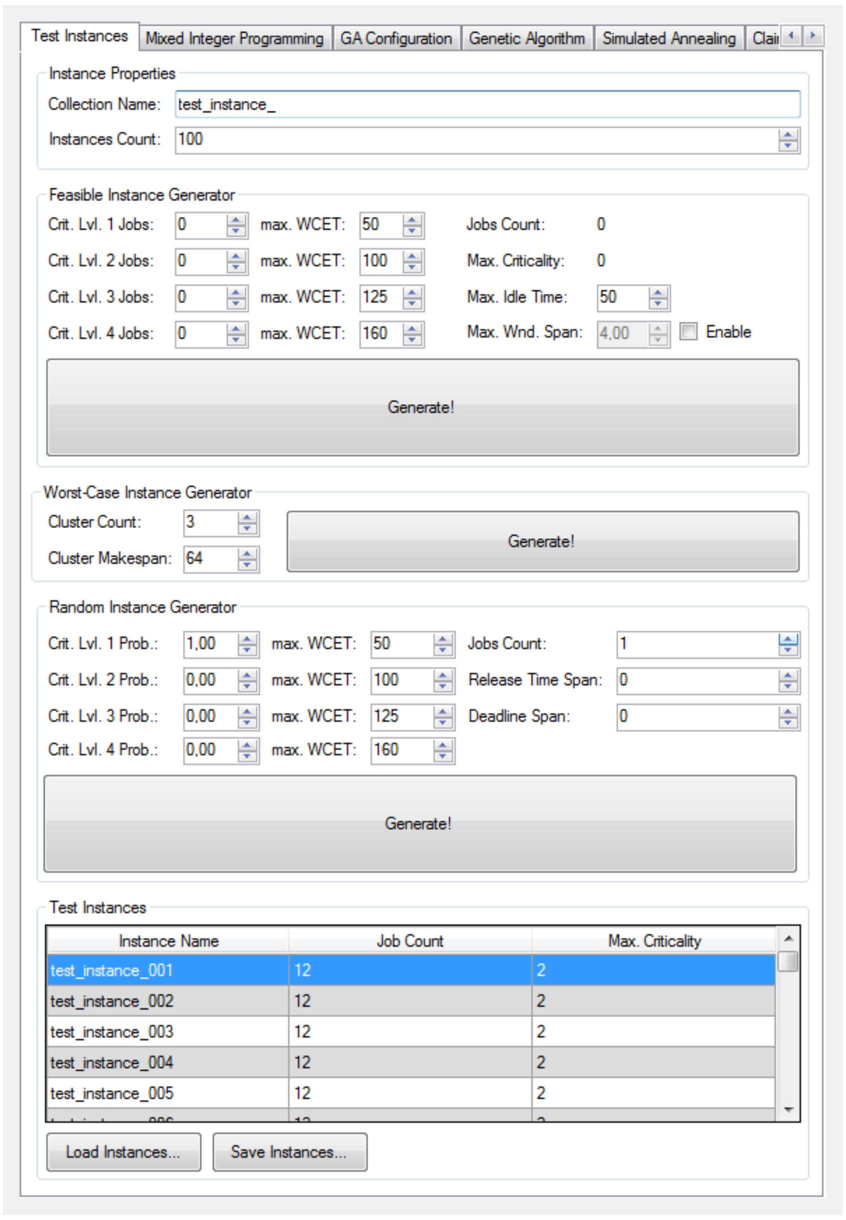
There is also a *command line user* interface that is discussed in the upcoming section. This interface is more practical for running multiple tests for a single test instance of a mixed-criticality instance.

TAB PAGE NO. 1: TEST INSTANCES

The Test Instances tab page provides 3 different methods for generating random mixed-criticality instances which may be instantly apparent from the big push buttons saying „Generate!“ in the figure 1.

The first part is used to give an appropriate name to each of the generated MC instance. The *Collection Name* sets the common name prefix for the instances and *Instance Count* denotes the number of instances to generate. As a result of the setting displayed in the figure 1, one hundred MC instances named *test_instance_000* to *test_instance_100* is going to be generated.

FIGURE 1: TEST INSTANCES TAB PAGE



Instance Properties

Collection Name: test_instance_

Instances Count: 100

Feasible Instance Generator

Crit. Lvl. 1 Jobs:	0	max. WCET:	50	Jobs Count:	0
Crit. Lvl. 2 Jobs:	0	max. WCET:	100	Max. Criticality:	0
Crit. Lvl. 3 Jobs:	0	max. WCET:	125	Max. Idle Time:	50
Crit. Lvl. 4 Jobs:	0	max. WCET:	160	Max. Wnd. Span:	4,00 ☐ Enable

Generate!

Worst-Case Instance Generator

Cluster Count: 3

Cluster Makespan: 64

Generate!

Random Instance Generator

Crit. Lvl. 1 Prob.:	1,00	max. WCET:	50	Jobs Count:	1
Crit. Lvl. 2 Prob.:	0,00	max. WCET:	100	Release Time Span:	0
Crit. Lvl. 3 Prob.:	0,00	max. WCET:	125	Deadline Span:	0
Crit. Lvl. 4 Prob.:	0,00	max. WCET:	160		

Generate!

Test Instances

Instance Name	Job Count	Max. Criticality
test_instance_001	12	2
test_instance_002	12	2
test_instance_003	12	2
test_instance_004	12	2
test_instance_005	12	2

Load Instances... Save Instances...

The second part is the 1st generator called *Feasible Instance Generator* and is called this way from an obvious reason – it generates instances with guaranteed feasible solution. The first column of edit boxes serves to set the number of MC jobs for each criticality level (denoted by row). The second column sets the maximum value of the randomly generated execution time for each job at the given criticality level. The third, and last, column shows the total number of MC jobs and the criticality of to-be generated MC instances. It also allows the user to se up the maximum length of the generated idle time intervals and release – deadline time interval. Pushing the Generate button generates the MC instances according to the setting; their names a basic parameters are displayed in the last part called *Test Instances* at the end of the same page.

The third part is the next generator called *Worst-Case Instance Generator*. Worst case in sense of that a solution to each of the generated instances is also a solution to some 3-partition problem, which is known NP-hard problem. The first parameter of the generator called *Cluster Count* sets up the number of MC jobs – it corresponds to a number of partitions, i.e. 4-times the given value: three criticality 1 jobs and one criticality 2 job. The parameter C*luster Makespan* sets the exact sum of the criticality 1 jobs' execution times and the value of the criticality 2 execution time for the criticality 2 jobs (these job have no execution time in criticality level 1). For more details, consult the complexity proof given in the article. It is also guaranteed that the generated instances do all have a feasible solution.

The fourth parth is the last generator called *Random Instance Generator*. This generator does not guarantee a feasible solution to all generated instances, because all the parameters are generated randomly. The settings are quite similar to the *feasible generator*, but unlike it, the 1st column is used to set up the criticality level distribution for MC jobs and that is why the user has to specify explicitly the number of jobs. But the 2nd column has the same role as in the *feasible generator*. Eventually, in the 3rd column the user sets up the release time window length *R* and the deadline span *D*: Each MC job *i* gets a randomly generated release time (r_i) and deadline to a value from the [0, R) and [r_i, r_i + D) interval, respectively.

The two push buttons at the end of the page saying „Load Instances..." and „Save Instances..." both show a File Browser dialog, but in the case of the first button, the user may choose a file with previously saved generated MC instances, which are then loaded into the application (like if they have been generated). In the case of the second button being pushed, the user have to choose a directory, where the last set of the generated instances will be saved – it is recommended that a new directory is created for each new generated set.

A GENERATED MC INSTANCE FILE FORMAT

When a generated mixed-criticality instance is saved into a file, it is in a plain text format described below:

```
N C
T_1  r_1  d_1  c_1  p_1  p_2 ... p_C
T_2  r_2  d_2  c_2  p_2  p_2 ... p_C
...
T_N  r_N  d_N  c_N  p_1  p_2 ... p_C
```

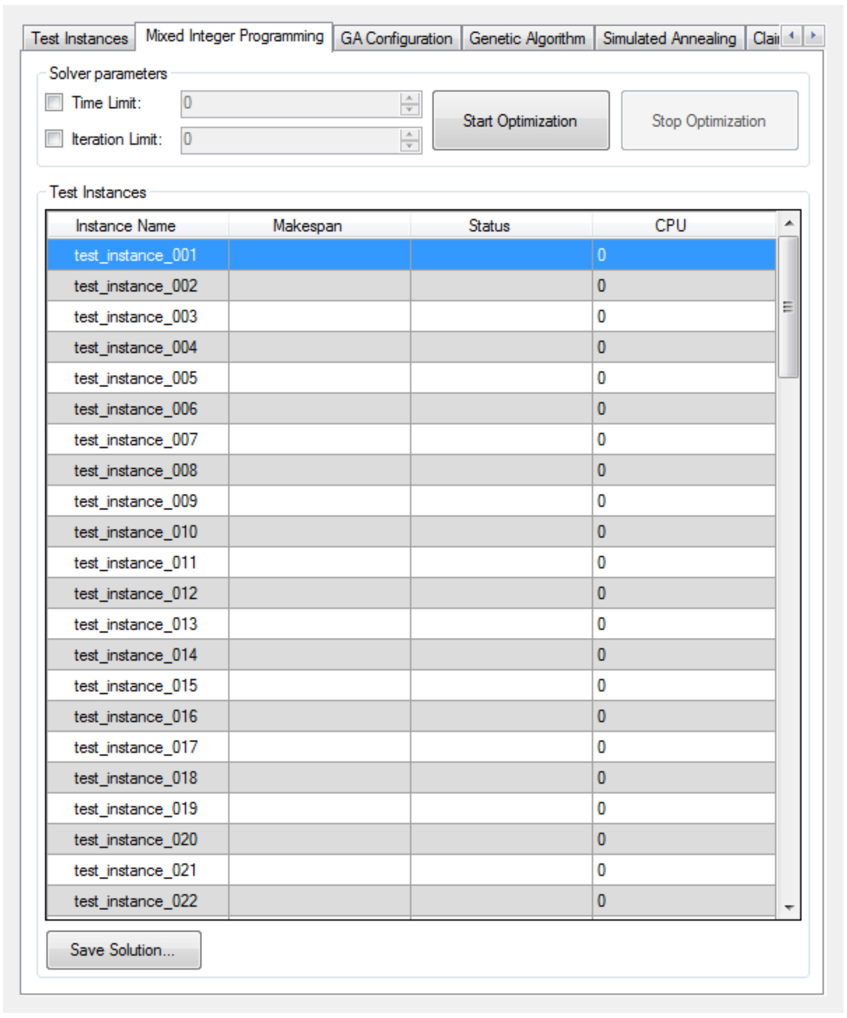
where *N* is the total number of MC jobs in the instance and *C* is the maximum criticality level of one of its jobs. After the first line N lines follow, each describing the parameters of an MC job *i*, which are: job's label T_i, release time r_i, deadline d_i, criticality c_i, and execution times p_1 to p_C. The execution times for higher criticality levels then the job's own criticality are set to 0.

TAB PAGE NO. 2: MIXED INTEGER PROGRAMMING

This page is dedicated to the 1st mixed-criticality scheduling solver based on a branch-and-bound algorithm implemented with Gurobi Optimizer library (version 4.5). The upper part of the page contains parameters that determines the time limit and/or the iteration limit for the solving algorithm. To set the wanted parameter, the appropriate checkbox needs to be selected first.

When the „Start Optimization" button is pressed the solving process is started. Each instance is being solved either to optimality or till the time/iteration limit is reached, then the makespan and the status of the soluton as well as the running time is displayed in the table for each of the instances.

FIGURE 2: MIXED INTEGER PROGRAMMING TAB PAGE



Solver parameters

Time Limit: 0

Iteration Limit: 0

Test Instances

Instance Name	Makespan	Status	CPU
test_instance_001			0
test_instance_002			0
test_instance_003			0
test_instance_004			0
test_instance_005			0
test_instance_006			0
test_instance_007			0
test_instance_008			0
test_instance_009			0
test_instance_010			0
test_instance_011			0
test_instance_012			0
test_instance_013			0
test_instance_014			0
test_instance_015			0
test_instance_016			0
test_instance_017			0
test_instance_018			0
test_instance_019			0
test_instance_020			0
test_instance_021			0
test_instance_022			0

The status of a solution may take one of these values for each of the described solvers:

- *Optimal* –An optimal solution has been found. The *Makespan* makes best value possible.
- *Suboptimal* – A suboptimal solution has been found. The *Makespan* makes an upper bound value.
- *Heuristic* – A result is not proven optimal and may be even infeasible. For solvers that output solutions with this kind of a result another column is provided where the total sum of the tardinesses of the jobs is presented.
- *Infeasible* – The solution does not exist – the MC instance has been proven infeasible.
- *Time Limit* – The time limit has been reached, no solution found.
- *Iteration Limit* – The iteration limit has been reached, no solution found.
- *Aborted* – No solving time has been given to such an instance, because the solver has been aborted.

This tab page looks exactly the same as the *Genetic Algorithm* page, but the solution table contains one more column *Lateness*, which shows the total sum of tardinesses of all the jobs in the given MC instance (see *Heuristic* solution status description given above).

TAB PAGE NO. 3: GA CONFIGURATION

This tab page (see figure 3) contains a configuration settings for a genetic algorithm that may be used for solving the given MC instances. The page is divided into several parts dedicated to setting of different kind of components the resulting genetic algorithm is made of.

The first part (*General*) is used to set up the population size and (if wanted) the number of elite individuals that will be always copied to the next generation during the evolution.

The second part (*Fitness*) is used to set up the coefficients of the fitness evaluation function, which is defined as

$$F(\pi) = \frac{1}{1 + E(\pi)}$$

where π is the individual (encoded solution to an MC instance) and $E(\pi)$ is the multicriteria evaluation function described in the section 5.1 of the article, here is just its definition:

$$E(\pi) = \alpha \sum_{i=1}^{n} \max\{0, S_{\pi(i)} + P_{\pi(i)}(\chi_{\pi(i)}) - d_{\pi(i)}\} + \beta C_{max}$$

The *Makespan Factor* and the *Lateness Factor* corresponds to the coefficient β and α, respectively. The user needs to change only one of these parameters, because the second one is recalculated automatically for their sum has to equal 1.

The next part called *Scaling* is a list of provided fitness scaling methods, which are used to scale (recalculate) fitness values of the individuals before the selection occurs. The methods are:

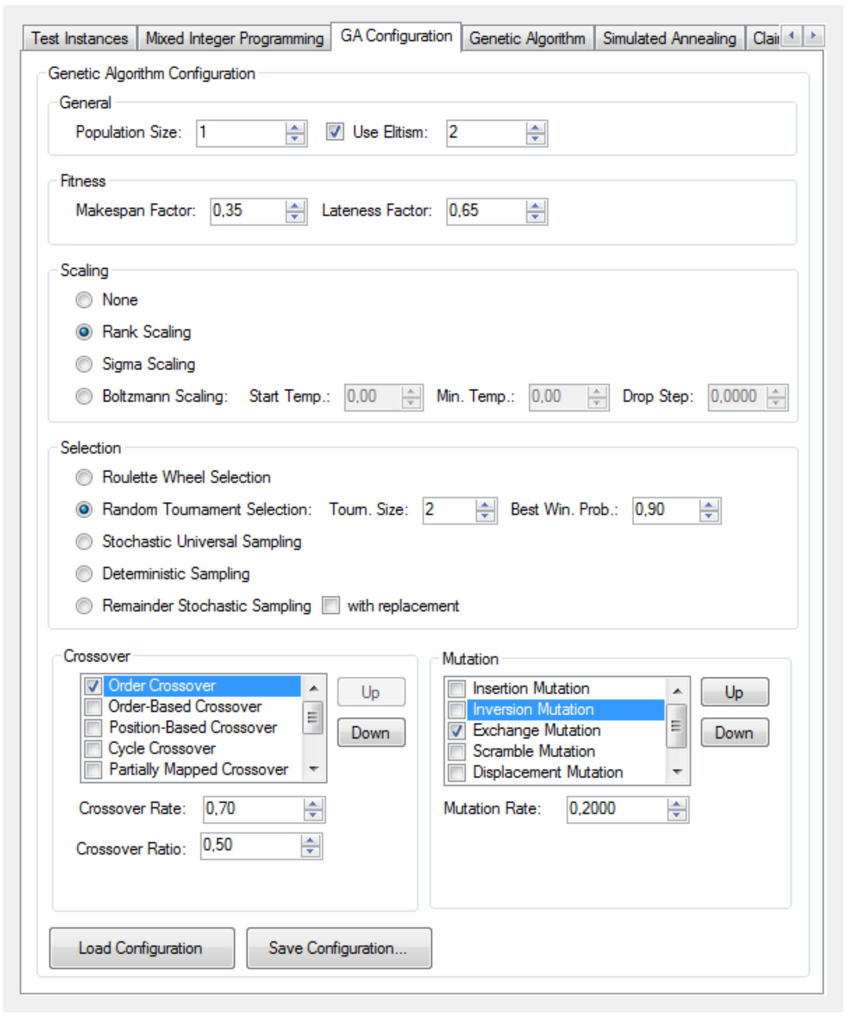
- *None* – no scaling is used.
- *Rank Scaling* – the individuals are sorted in ascending order according to their current fitness value. Then their new fitness value is set to the position in that sequence, resulting in the worst individual getting 1 and the best getting *n* (the population size).
- *Sigma Scaling* – the new fitness value F_{new} for each individial in the population is calculated from its old fitness value F_{old} according to $F_{new} = \frac{F_{old} - F_{\mu}}{2F_{\sigma}}$, where F_{μ} is the average fitness value and F_{σ} is the standard deviation of the fitness score.
- *Boltzmann Scaling* – the new fitness value F_{new} for each individial in the population is calculated from its old fitness value F_{old} according to $F_{new} = \frac{e^{\frac{F_{old}}{t}}}{Avg\left(e^{\frac{F_{old}}{t}}\right)}$, where e is the Euler's number and the term $Avg\left(e^{\frac{F_{old}}{t}}\right)$ denotes the average value of its argument over the whole population, t is the temperature parameter starting with value *Start Temp.* and slowly dropping by factor of *Drop Step* until reaching its minimum *Min. Temp.*

The fourth part (*Selection*) is used to pick a selection operator for the genetic algorithm. How each of these operators works is out of scope of this document. The provided selection operators are: *Roulette Wheel Selection* (most commonly used), Stochastic *Tournament Selection* (also frequently used for its simplicity a fast execution), *Stochastis Universal Sampling*, *Deterministic Sampling, Remainder Stochastic Sampling* (with or withour replacement).

The second to last part (*Crossover*) allows the user to select more then one crossover operator that will be used by the genetic algorithm. The parameters of the operators are *Crossover Rate* and *Crossover Ratio* (only some operators has it and it denotes the ratio of how much information is taken from „father“ and „mother“). The operators has to be checked to be selected and its order among the selected operators in the execution of the evolution may be change by Up and Down buttons. The crossover operator list contains: *Order Crossover, Order-Based Crossover, Position-Based Crossover, Cycle Crossover, Partially Mapped Crossover, Alternation Position Crossover, Heuristic Crossover,* and *Greedy Crossover*.

The last part (*Mutation*) allows the user to select more the one mutation operator that will be used by the genetic algorithm. The only parameter of the operators is *Mutation Rate*. The mutation operator list contains: *Insertion Mutation, Inversion Mutation, Exchange Mutation, Scramble Mutation, Displacement Mutation,* and *Displaced Inversion Mutation*.

FIGURE 3: GA CONFIGURATION TAB PAGE



Population Size: 1 ☑ Use Elitism: 2

Makespan Factor: 0,35 Lateness Factor: 0,65

○ None
● Rank Scaling
○ Sigma Scaling
○ Boltzmann Scaling: Start Temp.: 0,00 Min. Temp.: 0,00 Drop Step: 0,0000

○ Roulette Wheel Selection
● Random Tournament Selection: Tourn. Size: 2 Best Win. Prob.: 0,90
○ Stochastic Universal Sampling
○ Deterministic Sampling
○ Remainder Stochastic Sampling ☐ with replacement

☑ Order Crossover
☐ Order-Based Crossover
☐ Position-Based Crossover
☐ Cycle Crossover
☐ Partially Mapped Crossover

Crossover Rate: 0,70
Crossover Ratio: 0,50

☐ Insertion Mutation
☐ Inversion Mutation
☑ Exchange Mutation
☐ Scramble Mutation
☐ Displacement Mutation

Mutation Rate: 0,2000

The configuration of the genetic algorithm may be saved into a text file and then later loaded back into the application by the two push buttons located at the bottom of this tab page. The format of this configuration file is described at the end of this document.

TAB PAGE NO. 4: GENETIC ALGORITHM

This page is dedicated to the 2nd mixed-criticality scheduling solver based on the genetic algorithm that has been configured on the previous page. The solver want work if there is no valid configuration present. This tab page looks in essence the same as the mixed integer programming tab page, so please, refer to the the description of that particular tab for more information.

TAB PAGE NO. 5: SIMULATED ANNEALING

This page (see figure 4) is dedicated to the 3rd mixed-criticality scheduling solver based on a simulated annealing algorithm.

The upper part of the page contains parameters that determines the time limit and/or the iteration limit for the solving algorithm. To set the wanted parameter, the appropriate checkbox needs to be selected first.

The process of optimization is initiated or aborted by pressing the push button saying „Start Optimization“ or „Stop Optimization“, respectively. The description of the results of the optimization are mentioned in the previous Mixed Integer Programming Tab section.

The middle part of the page called *Simulated Annealing Configuration* deals with, as its name suggests, the configuration of the simulated annealing algorithm.

The first two parameters that reside in the *Solution Evaluation* part are the same as for the genetic algorithm, please refer to the description of the GA Configuration tab page for more information.

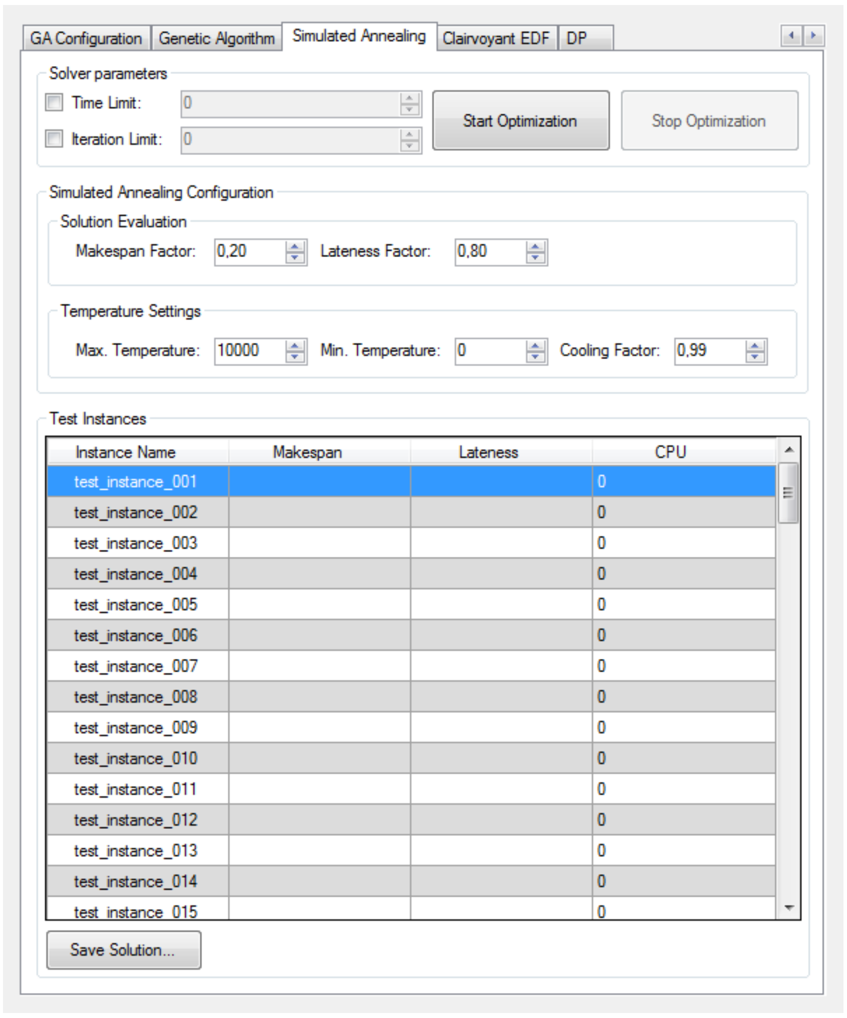
The parameters in the *Temperature Settings* part directly influence the process of „crawling“ in the neighbourhood of a specific solution of an MC instance. The simulated annealing algorithm works basically as follows, first it starts with some (preferably good) initial solution to the given MC instance *S*. The solution *S* is then tweaked with („perturbed“) a little bit and a new solution is created, lets call it R. If the quality of R is better then the quality of S (measured by the multicriteria evaluation function *E* described above), then S is reassigned with R and the process continues in a loop. On the other hand, if R is worst then S, there is still a chance that S will be reassigned with R and this chance is described by the cummulative distribution function

$$P\left(U < e^{\frac{E(R)-E(S)}{t}}\right)$$

where *U* is a radom variable drawn from a uniform distribution on the unit interval [0, 1]. Here the *temperature* parameter comes to light, because *t* denotes temperature. The temperature initial value is set to the value of the parameter *Max. Temperature.* At the end of each loop of the algorithm the temperature *t* drops down by a factor given by the parameter *Cooling Factor* , but it never falls below the value of the parameter *Min. Temperature*.

The process of optimization is initiated or aborted by pressing the push button saying „Start Optimization“ or „Stop Optimization“, respectively. The description of the results of the optimization are mentioned in the previous Mixed Integer Programming Tab section.

FIGURE 4: SIMULATED ANNEALING TAB PAGE



Solver parameters

Time Limit: 0

Iteration Limit: 0

Simulated Annealing Configuration

Solution Evaluation

Makespan Factor: 0,20 Lateness Factor: 0,80

Temperature Settings

Max. Temperature: 10000 Min. Temperature: 0 Cooling Factor: 0,99

Test Instances

Instance Name	Makespan	Lateness	CPU
test_instance_001			0
test_instance_002			0
test_instance_003			0
test_instance_004			0
test_instance_005			0
test_instance_006			0
test_instance_007			0
test_instance_008			0
test_instance_009			0
test_instance_010			0
test_instance_011			0
test_instance_012			0
test_instance_013			0
test_instance_014			0
test_instance_015			0

TAB PAGE NO. 6: CLAIRVOYANT EDF

This page (see figure 5) is dedicated to the 4th mixed-criticality scheduling solver based on an adaptation of clairvoyant earliest deadline first algorithm by Cecilia Ekelin.

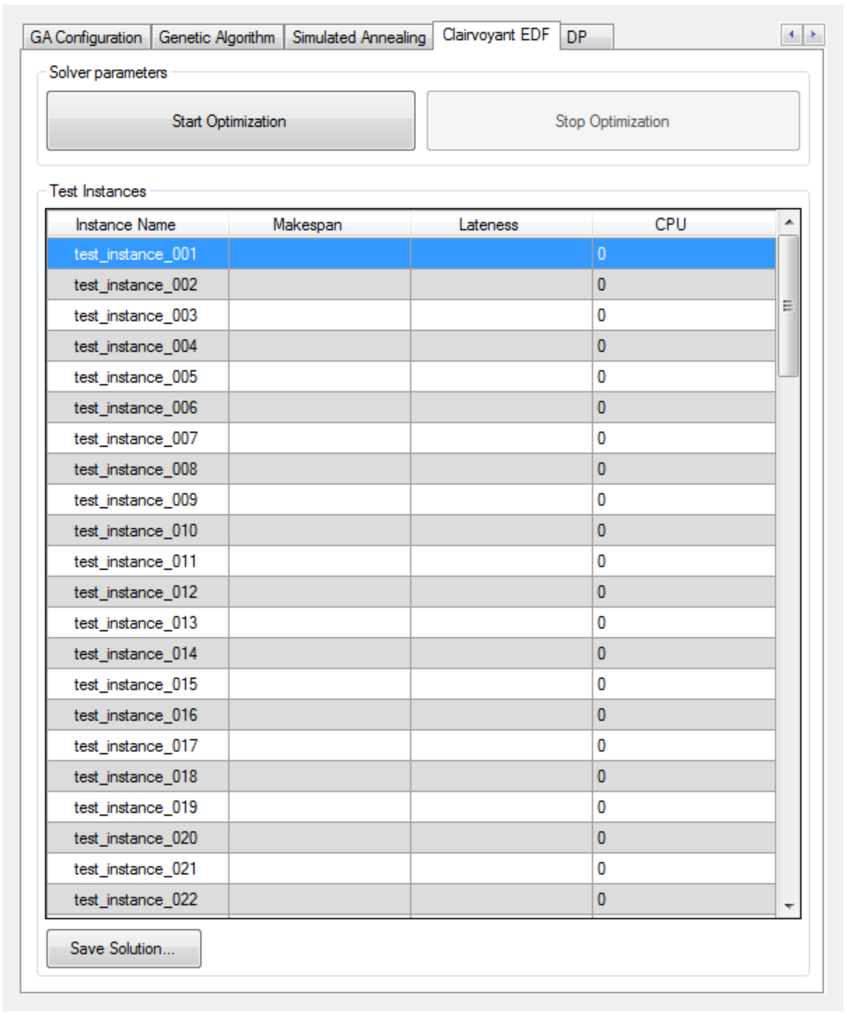
The page contains only a table of MC instances to solve and two push buttons. The process of optimization is initiated or aborted by pressing the push button saying „Start Optimization“ or „Stop Optimization“, respectively. The description of the results of the optimization are mentioned in the previous Mixed Integer Programming Tab section.

TAB PAGE NO. 7: DP

This page is dedicated to the last mixed-criticality scheduling solver based on a dynamic programming algorithm quite similar to an algorithm for solving TSP. Because the algorithm has exponential memory demands for its execution, it is useful only for really small instances (containing up to 12 jobs).

The page looks the same as the previous *Clairvoyant EDF* tab page (see figure 5). It contains only a table of MC instances to solve and two push buttons. The process of optimization is initiated or aborted by pressing the push button saying „Start Optimization“ or „Stop Optimization“, respectively. The description of the results of the optimization are mentioned in the previous Mixed Integer Programming Tab section.

FIGURE 5: CLAIRVOYAND EDF TAB PAGE



Instance Name	Makespan	Lateness	CPU
test_instance_001			0
test_instance_002			0
test_instance_003			0
test_instance_004			0
test_instance_005			0
test_instance_006			0
test_instance_007			0
test_instance_008			0
test_instance_009			0
test_instance_010			0
test_instance_011			0
test_instance_012			0
test_instance_013			0
test_instance_014			0
test_instance_015			0
test_instance_016			0
test_instance_017			0
test_instance_018			0
test_instance_019			0
test_instance_020			0
test_instance_021			0
test_instance_022			0

SAVING AN MC INSTANCE SOLUTION

Each of the solver tabs containts at its buttom a single push button saying „Save Solution...“. When it is pressed a File Browser dialog appears and the user may choose a directory, where all the solutions is going to be saved. The saved solution file is given the name of the MC instance with a file extention „*.sol“ and its plain text content has the following format

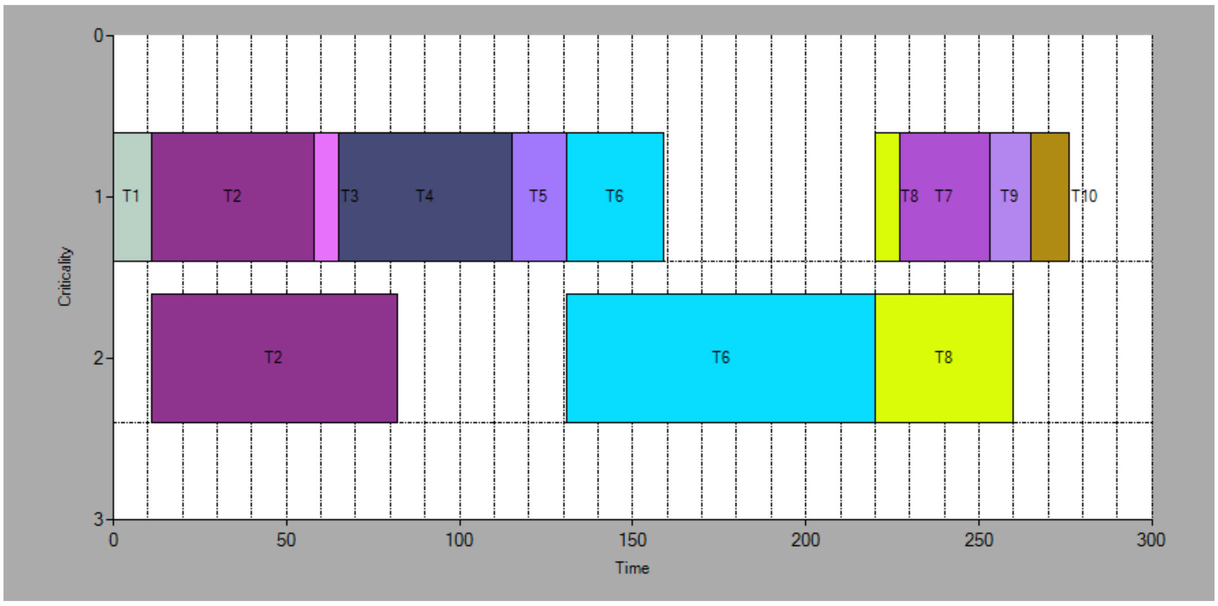
$$
\begin{array}{l}
M\ \ L \\
T_1\ \ s_1 \\
T_2\ \ s_2 \\
\ldots \\
T_N\ \ s_N
\end{array}
$$

where M is the makespan of the schedule, L is the sum of all the jobs' tardinesses. After the first line N (the number of jobs in the solved MC instance) lines follows, each of them contains the name of the job with its given start time.

PREVIEWING AN MC INSTANCE SOLUTION

Once an MC instance is solved a user may double-click on the specific MC instance in the table (of the solver tab page) and a solution preview in form of a Gantt chart will be displayed. An example how this chart looks like, see figure 6 below.

FIGURE 6: AN EXAMPLE OF THE MC INSTACE SOLUTION PREVIEW



COMMAND LINE USER INTERFACE

The MCScheduling application may be also executed from the command line with one or more arguments that are described in this section. This way no graphical user interface is available.

```
MCScheduling -s <solver type> -f <solver config. file> -t <time limit>
             -i <iteration limit> -c <number of repetitions>
             -in <input file> -out <output file>
```

The solver type argument `-s` may take one of the following values:

- *mip* – The selected solver type is mixed integer programming.
- *ga* – The selected solver type is the genetic algorithm.
- *sa* - The selected solver type is simulated annealing algorithm.
- *cedf* - The selected solver type is the clairvoyant EDF algorithm.
- *dp* - The selected solver type is the dynamic programming algorithm.

The genetic algorithm and simulated annealing algorithm require a configuration in order to execute and for that purpose there is the `-f` argument which takes a file path to an appropriate configuration file.

The `-t` argument is used to set up the time limit in seconds and it takes only integer values.

The `-i` argument is used to set up the iteration limit for the solving algorithm and it takes only integer values.

The `-c` argument takes an integer value that determines the number of repetitions. Each instance is solved as many times as it is the value of this argument. The final solution is then taken from as the average of all the repetitive solutions.

The `-in` argument determines the file path to the MC instance to solve.

And, eventually, the `-out` argument determines the file path where the solution to the specified MC instance will be saved.

Example:

```
mcscheduling.exe -s mip -t 60 -in test01.ins -out mip_test01.sol
```

This command will execute the mixed integer programming solver on the instance in the file *test01.ins* and it will run maximally for 1 minute; the solution is then saved to a file named *mip_test01.sol*.

COMMAND LINE MC INSTANCE SOLUTION FILE FORMAT

The solution of an MC Instance solved from the command line differs from the format described above in the case the GUI is used. It is a plain text format like this

```
M
L
T
S
```

where *M* is the makespan of the schedule, *L* is the sum of all the jobs' tardinesses, T is the total running time of the solver in seconds and *S* is the status of the solution that takes one of the following values:

- *optimal* –An optimal solution has been found. The *Makespan* makes best value possible.
- *suboptimal* – A suboptimal solution has been found. The *Makespan* makes an upper bound value.
- *heuristic* – A result is not proven optimal and may be even infeasible. For solvers that output solutions with this kind of a result another column is provided where the total sum of the tardinesses of the jobs is presented.
- *infeasible* – The solution does not exist – the MC instance has been proven infeasible.
- *time limit exceeded* – The time limit has been reached, no solution found.
- *iteration limit exceeded* – The iteration limit has been reached, no solution found.
- *aborted* – No solving time has been given to such an instance, because the solver has been aborted.
- *error* – An error occurred during execution.
- *unknown* – The result is unknown. The instance has not been solved, probably.

GENETIC ALGORITHM CONFIGURATION FILE FORMAT

This section contains the description of a genetic algorithm configuration file format. The genetic algorithm configuration file is a plain text file that has to have filled a few lines in the fixed order. Each of these lines determines a configuration of a particular component of the genetic algorithm.

The file format has this form

```
name
popsize
elitecnt
MML alpha beta
scaler
selector
crossoverOp1
crossoverOp2
...
mutationOp1
mutationOp2
...
```

The first line is the name of the configuration; it may be any non-empty string. The second line must contain an integer value that determines the size of the population. The *elitecnt* determines the number of fittest individuals (elite) that will be copied to the next generation in every iteration of the algorithm.

The 4th line sets up the multi-criteria fitness function. The arguments are (sadly) required to be decimal values that add up to 1.

The 5th line determines the fitness scaling method and may take one of the following values:

- *none* – no scaling.
- *rank* – rank scaling.
- *sigma* – sigma scaling.
- *boltzmann startTemp minTemp coolStep* – boltzmann scaling with 3 decimal arguments.

These scaling methods and their arguments are discussed above in the section about the genetic algorithm configuration.

The 6th line determines the selection method and it may take one of the following values:

- *RWS* – Roulette Wheel Selection.
- *RTS tourSize fitterWinProb* – Stochastic Tournament Selection.
- *SUS* – Stochastic Universal Sampling.
- *DS* – Deterministic Sampling.
- *RSS withRepl* – Remeainder Stochastic Sampling (with replacement if *withRepl* is 1). withRepl may be 0 or 1.

These selection methods and their arguments are mentioned above in the section about the genetic algorithm configuration.

The next lines have to contain consecutive list of crossover operators, but at least one operator. Every operator has one decimal argument that denotes the crossover rate and should be between 0 and 1 (inclusive). Moreover, some operators have a second argument which determines the ratio of information taken from the parents. The crossover operators may be defined with their arguments like this:

- *OX rate ratio* – Order Crossover.
- *OBX rate ratio* – Order-Based Crossover.
- *PBX rate ratio* – Position-Based Crossover.
- *CX rate* – Cycle Crossover.

- *PMX rate* – Partially Mapped Crossover.
- *APX rate* – Alternating Position Crossover.
- *HX rate* – Heuristic Crossover.
- *GX rate* – Greedy Crossover.

Then has to follow a consecutive list of mutation operator, at least one mutation operator has to be specified. Every operator has one decimal argument that denotes the mutation rate and should be between 0 and 1 (inclusive). The mutation operators may be defined with their argument like this:

- *DM rate* – Displacement Mutation.
- *IM rate* – Insertion Mutation.
- *IVM rate* – Inversion Mutation.
- *EM rate* – Exchange Mutation.
- *SM rate* – Scramble Mutation.
- *DIVM rate* – Displaced Inversion Mutation.

SIMULATED ANNEALING CONFIGURATION FILE FORMAT

This section contains the description of a simulated annealing algorithm configuration file format. The simulated algorithm configuration file is a plain text file that has to contain five lines of text that determine the setting of different parameter of the algorithm. The file format has this form

```
alpha
beta
coolFactor
minTemperature
maxTemperature
```

The first two lines determine the coefficients of the multi-criteria evaluation function described in one of the preceding sections. The maximum and minimum temperature as well as cooling factor is described (quite in detail) in the Simulated Annealing Tab section.