Technical Documentation For The Creation of Test Data GUI

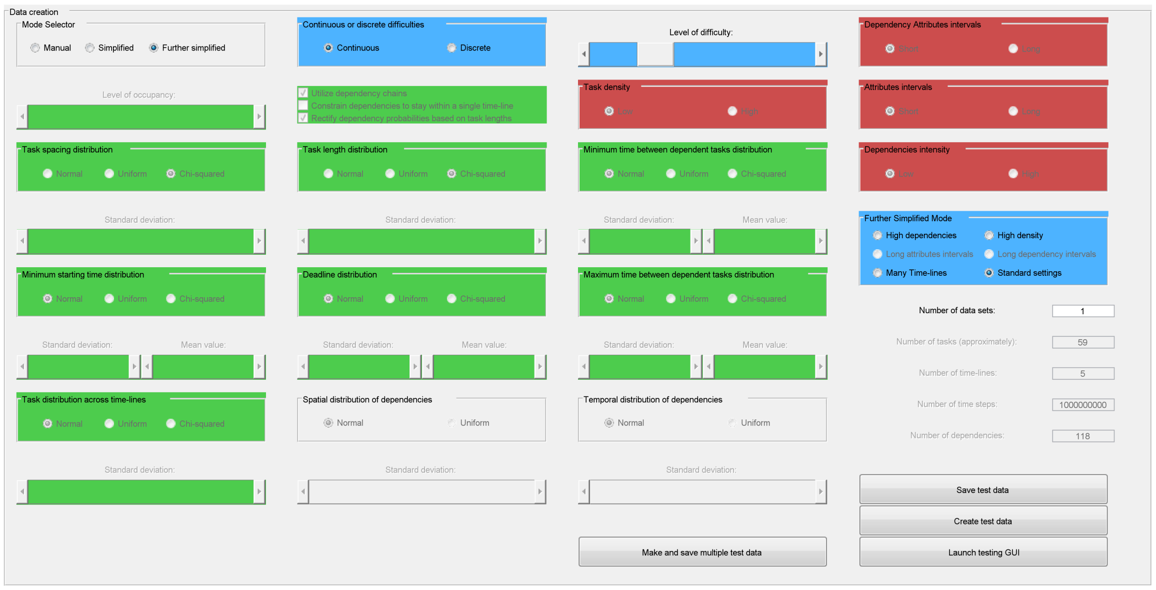
Isak Bohman
MAI, Linköping University

January 18, 2016

1 Introduction

The following document outlines the workings of the test data GUI. Note that a description of the interface itself is given in a separate manual.

Figure 1: The user interface where all data parameters for test data are specified, and using which test data are created and saved.



Data creation

Mode Selector: Manual Simplified [✓ Further simplified]

Continuous or discrete difficulties: [✓ Continuous] Discrete

Level of difficulty:

Level of occupancy:

[✓ Utilize dependency chains]
Constrain dependencies to stay within a single time-line
[✓ Rectify dependency probabilities based on task lengths]

Task density: [✓ Low] High

Dependency Attributes intervals: [✓ Short] Long

Attributes intervals: [✓ Short] Long

Dependencies intensity: [✓ Low] High

Task spacing distribution: Normal Uniform [✓ Chi-squared]
Standard deviation:

Task length distribution: Normal Uniform [✓ Chi-squared]
Standard deviation:

Minimum time between dependent tasks distribution: [✓ Normal] Uniform Chi-squared
Standard deviation: Mean value:

Further Simplified Mode: High dependencies, High density, Long attributes intervals, Long dependency intervals, Many Time-lines, [✓ Standard settings]

Minimum starting time distribution: [✓ Normal] Uniform Chi-squared
Standard deviation: Mean value:

Deadline distribution: [✓ Normal] Uniform Chi-squared
Standard deviation: Mean value:

Maximum time between dependent tasks distribution: [✓ Normal] Uniform Chi-squared
Standard deviation: Mean value:

Task distribution across time-lines: [✓ Normal] Uniform Chi-squared
Standard deviation:

Spatial distribution of dependencies: [✓ Normal] Uniform
Standard deviation:

Temporal distribution of dependencies: [✓ Normal] Uniform
Standard deviation:

Number of data sets: 1
Number of tasks (approximately): 59
Number of time-lines: 5
Number of time steps: 1000000000
Number of dependencies: 118

2 The Inner Workings of the Test Data GUI

Here, we first present the files used in the creation of test data, and describe the contents of each file. Note that function names in Matlab are equal to the corresponding file name, without the .m file extension. We will use the words "function" and "file" interchangeably throughout this section.

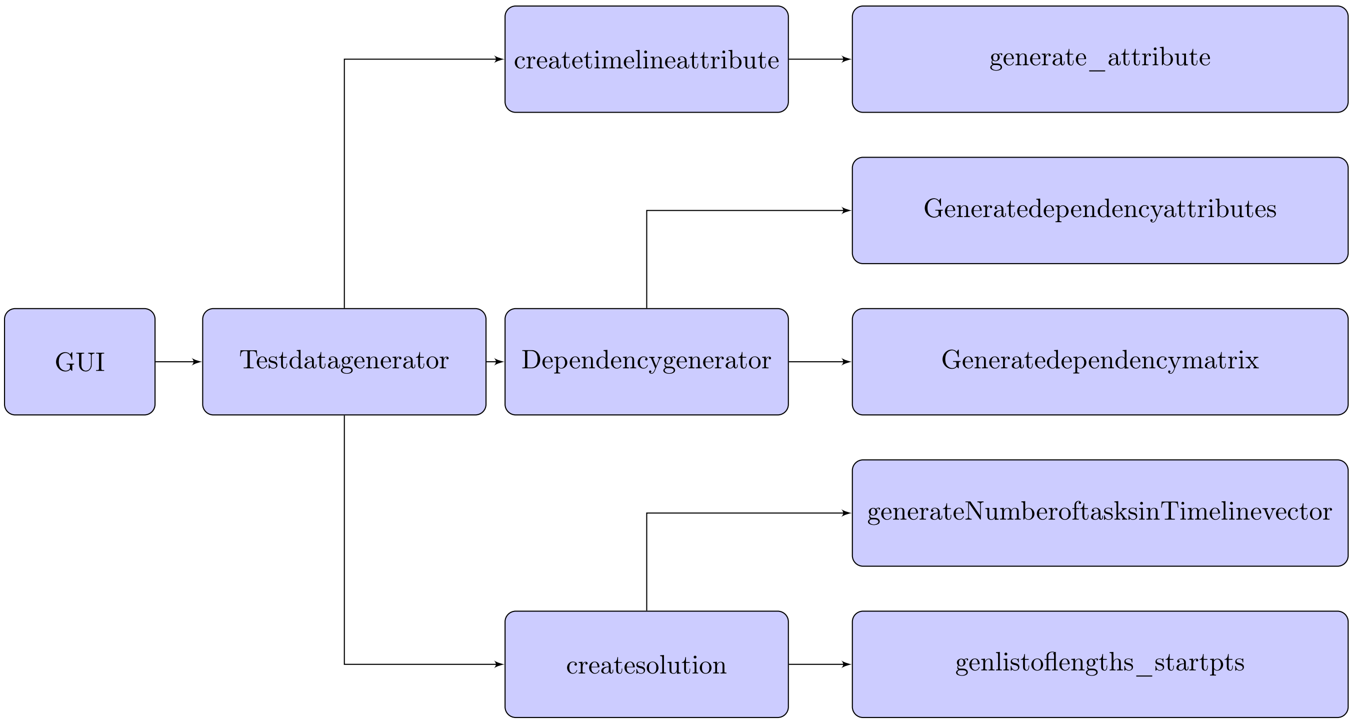
2.1 Overview of functions

File	Description
chi2_distribution.m	Function which generates chi2-distributed random variables.

File	Description
createdat.m	Creates AMPL-friendly data
createsolution.m	Creates an admissible solution of the problem with specified parameters.
createtimelineattribute.m	A function which generates a time-line attributes matrix.
Dependencygenerator.m	A function which calls the function for generating dependencies and their attributes.
generate_attribute.m	Generates a single time-line attribute, which is the allowable interval of placement of a task.
Generatedependencyattributes.m	A function which generates the attributes of dependencies, given a dependency matrix. The attributes are the admissible intervals within which two tasks in a dependency must be completed.
Generatedependencymatrix.m	A function which generates a dependency matrix.
generateNumberoftasksinTimelinevector.m	Function for generating the number of tasks on each time-line.
genlistoflengths_startpts.m	Function for generting a list of starting points and lengths of tasks.
GUI.m	The file from which the GUI is run.
ModifiedAMPLmatrix.m	Helps transform data from the original implementation into an AMPL-friendly shape.
normal_distribution.m	A function for generating a folded Gaussian distributed random variable.
test_gui.m	The rudiments of a testing GUI, which is not used.
Testdatagenerator.m	A function ordering the creation of all data.
uniform_distribution.m	Function for generating a uniformly distributed random variable.

2.2 Flow chart

Here, a simple flow chart is exhibited, which attempts to clarify the situation to the reader. The relations between the most important functions are given here. Also, it shall be noted that several of these functions use the functions uniform_distribution, chi2_distribution or normal_distribution; this is however not displayed here in order not to obfuscate the picture.



2.3 Detailed function descriptions

Here, we intricately describe the functions being used, where the function name is just the file name, excluding the .m file extension.

2.3.1 chi2_distribution.m

Input arguments	Output
mu, sigma	num

This function simply creates a number (=num) based on a chi-squared distribution (+ a fixed length) based on a prescribed standard deviation (=sigma) and expected value (=mu). The number of degrees of freedoms used is 3.

2.3.2 createdat.m

Input arguments	Output
taskfile, depfile, datafilename	none

This function transforms the created data sets into a single file which can be used in AMPL. In order to accomplish this, the proprietary function createdat is used.

2.3.3 createsolution.m

Input arguments	Output
N, L, T, generateNumberoftasksinTimelinevector, occupancy, genlistoflengths_startpts, std7, distrib7, distrib8, std8, distrib4, std4	TimelineSolution

Creates a time-line solution. This file is essentially a collection of function calls to the functions genlistoflengths_startpts and generateNumberoftasksinTimelinevector. For every time-line in the cell matrix created by generateNumberoftasksinTimelinevector, lists of lengths and starting times of tasks are created and stored.

2.3.4 createtimelineattribute.m

Input arguments	Output
TimelineSolution, L, std3, distrib3, mu11, distrib5, std5, mu3	TimelineAttribute

This function creates the attributes of tasks, i.e. the admissible intervals of placement for tasks. It does so by iteratively calling the function generate_attribute.

2.3.5 Dependencygenerator.m

Input arguments	Output
TimelineSolution, Generatedependencymatrix, Generatedependencyattributes, Ndependencies, L, N, T, rectify, distrib6, std6, mu4, std2, distrib2, mu2, constrain, chains	DependencyMatrix, DependencyAttribute

Contains function calls to the functions Generatedependencymatrix and Generatedependencyattributes and nothing more.

2.3.6 generate_attribute.m

Input arguments	Output
TimelineSolution, L, mu1, sigma1, distribution1, mu2, sigma2, distribution2	TimelineAttribute

This function generates single dependency attribute, which is an admissible interval of placement for a task on a timeline. In order to accomplish this, the function must know a number of things. First a ballpark measure of the typical lengths of intervals. Here, we have made it so that the number of intervals tasks (with lengths comparable to the length of the corresponding task) is approximately 25 %, the proportion of tasks with intervals comparable to the length of the entire timeline is also 25 %, while a medium length setting is applied to the other half of all tasks. The precise lengths are adjusted using the mu3 and mu11 parameters (which set the expected lengths of the lower and the upper limits, respectively), standard deviations are given by std3 and std5 while the distributions being sampled are dsitrib3 and distrib5. The input arguments variance and mu are not used. For each task in the given TimelineSolution, a timeline attribute is generated using the function generate_attribute; previously, a function called Attributegenerator was used, although this approach was subsequently scrapped in favor of a more general one, where the selection of distributions is made explicit.

2.3.7 Generatedependencyattributes.m

Input arguments	Output
TimelineSolution, DependencyMatrix, L, N, T, distrib6, std6, mu4, std2, distrib2, mu2	DependencyAttribute

This function generates dependency attributes (=DependencyAttribute) for every entry in the dependency matrix (=DependencyMatrix), i.e. a minimum amount of time that must pass between the end of the first task in the dependency and the onset of the second task, as well as a maximum such time. The distribution of interval lengths is determined in the same way as timeline attributes, with distrib6, std6 and mu4 pertaining to the lower limit of the interval and std2, distrib2 and mu2 determining the upper limit.

2.3.8 Generatedependencymatrix.m

Input arguments	Output
TimelineSolution, Ndependencies, rectify, constrain, L, chains	DependencyMatrix

A function which generates dependencies between tasks. This function is by far the second longest currently in use, and will require some clarification. The argument Ndependencies determines the number of dependencies that will be created. The arguments variance and mu are not in use, as of current. The argument rectify determines if dependency probabilities are determined in proportion to lengths of tasks, or if each task has an equal probability of partaking in a dependency, regardless of its length. The argument constrain determines if we are to require that all dependencies should be between tasks on the same timeline. Chains have been implemented and can be utilized by setting the argument chains to 1. The specifications of chains were provided by the project customer and are the following. The minimum length of a chain is 3 tasks, while the maximum length is 10 tasks; each chain jumps from one timeline to another exactly once. The length requirements are approximately satisfied in the current implementation, while the jump requirement is always satisfied. Before the implementation of chains, this function was fairly easy examine, which may not be the case now. If chains are not to be used, the procedure of creating chains is rather simple. In short, this approach is randomly select a candidate task from the timeline, which is to take part in a dependency. This selection process obviously depends on what have have chosen the argument rectify to be. Now, depending on what we have chosen the argument constrain to be, an 'admissible set' of tasks from which we are to choose the second task in the dependency is created. However, in the creation of this

set, we don't take into account the tasks the current task may be dependent upon. Hence, we will have to create a set of tasks that the current task is dependent upon and take this into account when randomly selecting the next task in the dependency. This is essentially what is done.
If we shall utilize dependency chains, the implementation above must be modified. If the use of chains is activated, we first randomly determine if we shall begin on a chain or if a regular dependency is to be created. For the latter choice we proceed as previously described. If, however, a chain is to be created, then we first set the chain counter and the current task in the chain. Since we are at the first element of the chain, we first check if a dependency shall be generated by going backward or by going forward on the timeline. We first attempt to create one by going in the forward direction, but this may not always work out, if, for instance, the current task is the last one on a timeline. Then, we will have to check if we can create one by going in the backward direction instead. In rare, cases, even this might not work, and the chain will have to be scrapped. Subsequently, an admissible set is created, which takes into account all eventualities. This admissible set differs in its creation when compared to the admissible set when chains are not utilized. Here, we subtract the elements that the current task is dependent upon already from the beginning, so that the second task in the dependency can be chosen by simply randomizing a number from the admissible set.
In slightly more detail, the creation of the admissible set proceeds as follows. There are a few different variables that give rise to cases, and combinations of these. The various cases differ slightly, or significantly, from each other. First, there is the binary-valued direction parameter, the function of which has already been explained. Then there is the constrain parameter, which also has been explained. The parameter first_pass determines if we are to prioritize that every task receives a dependency. When it is activated, the admissible set is created so that every task involved in at least one dependency is excluded. If, however, this set turns out to be empty, we enter an amended mode where we don't demand that we only choose tasks without any dependencies. The two parameters timeline_jumps and pot_jump work in essentially the same fashion, although the functions of these have not been completely implemented (for instance, if timeline_jumps is 1, then it should be completely impossible for the chain to jump between two timelines). Experiments show that there is a high degree of variability as to the proportion of tasks involved in chains when compared to those which are not. The creation of dependencies also takes into account that it is of high priority that every task partakes in at least one dependency. This condition is essentially guarranteed to be satisfied if the number of dependencies is at least as great as the total number of tasks, which is the case for all test data sets used in the project.

2.3.9 generateNumberoftasksinTimelinevector.m

Input arguments	Output
Ntasks, Ntimelines, distrib8, std8	NumberoftasksinTimelinevector

This function call for the assignment of tasks to different timelines. The output, NumberoftasksinTimelinevector, is a vector consisting of a list of numbers which are the number of tasks that each timeline shall contain. These numbers are randomly selected from the distribution distrib8, with an expected value equal to the average number of tasks per timeline (Ntasks/Ntimelines) and the standards deviation std8.

2.3.10 genlistoflengths_startpts.m

Input arguments	Output
L, N, occupancy, distrib4, std4, std7, distrib7	start_time_list, length_list

This function is a combination of two functions which were previously used. However, a more unified approach is obtained when combing these, as done here. For a single timeline, this function generates a list of starting points and a list of lengths of tasks. This is carried out in the following way. First the level of occupancy is be used in order to determine the expected amount of space between tasks as well as the expected level of occupancy of the timelines. Subsequently, lists of task lengths and starting points are generated simultaneously in an iterative fashion, where an element is added to each list every other time in the iteration. The iterator terminates when the length of the timeline has been matched or exceeded. The lengths of tasks are determined from the distribution distrib4, with a standard deviation std4 and expected value as previously described. The spacing's between tasks are determined by the distribution distrib7, with a standard deviation of std7 and expected value as previously described.

2.3.11 GUI.m

Input arguments	Output
none	none

The main and by far the longest function. This is where the user interface is defined, using which data are created. Much of the function contains logic pertaining to rather mechanical and repetitive tasks such as handling callbacks of button presses, sliders, radio buttons and checkboxes; all of this will not be described here as a space-saving measure. The more interesting parts of the function are the following. It is possible to save data using a button in the interface, the functionality of which is detailed in the function testdatasave_callback. This function will save a created test data set in a new folder, which in its name will contain the current date as well as the type of data created, i.e. type A, B, C, or D data, or possibly X or Z if data sets have been created in any the more customizable modes. The button "Create test data" calls upon the function testdatagen_callback and does roughly the following. First, global variables pertaining to previously saved test data sets are cleared. Then, the function Testdatagenerator is run in order to create new test data sets. However, since the user may be requesting the creation of multiple test data sets, created using the same parameters, we iterate over the number of data sets that are to be created. Subsequently, the data sets returned from the function Testdatagenerator are converted into the forms in which they are saved, when the button "save test data" is pressed. In order to illustrate the results, the function print_correct is run, whereupon three different figures are updated.

The button "Make and save multiple test data" exhibits roughly the same functionality as the button "Create test data", only that it iterates over several different modes and difficulty settings, while saving the created test data sets between iterations. The exact basis of this was decided in an ad-hoc manner due to the specifics of the project at hand.

2.3.12 ModifiedAMPLmatrix.m

Input arguments	Output
fid, pname, p, colhdrtype, varargin	count

A proprietary function used for creating data suitable for application in an AMPL setting.

2.3.13 normal_distribution.m

Input arguments	Output
mu, sigma	num

This function generates a number (=num) using a folded normal distribution. Since the numbers generated are required to be non-negative, the input arguments of standard deviation (=sigma) and expected value (=mu) are only approximately accurate.

2.3.14 Testdatagenerator.m

Input arguments	Output
N, L, T, generateNumberoftasksinTimelinevector, Generatedependencymatrix, Generatedependencyattributes, Ndependencies, occupancy, genlistoflengths_startpts, rectify, std1, std2, std3, std4, std5, std6, std7, std8, std9, mu11, mu21, mu3, mu4, distrib1, distrib2, distrib3, distrib4, distrib5, distrib6, distrib7, distrib8, distrib9, constrain, chains	TimelineSolution, TimelineAttributeList, DependencyMatrix, DependencyAttribute

This the the function which is called upon when pressing the "Create data" button in the interface. This function contains a collection of calls to other functions; among them createsolution, createtimelineattribute as well as Dependencygenerator. It shall be noted that createtimelineattribute is run iteratively, in order to obtain timeline attributes for the entire timeline solution.

2.3.15 uniform_distribution.m

Input arguments	Output
mu, sigma	num

This function takes as input a standard deviation (=sigma) and expected value (=mu), using which a number (=num) is generated from a distribution exhibiting the desired characteristics.

3 Specifications of test data sets

Test data are represented by collection of TimelineAttribute matrices, a dependency matrix and as well as an admissible solution (called TimelineSolution), which, however, is not saved. Every data set is written in a separate directory, labeled with the data type, complexity, an index as well as a time stamp. To see how data are stored, let us consider some examples. A20_2_2015-11-16T15-02-57 is a valid directory name and its meaning is the following. 'A' is the type of data which is stored inside the directory, in this case type A data. 20 is the level of complexity. 2 is an index indicating the order of the test data set in a succession of test data sets created using the same parameters, each of which data set is stored in a unique directory. The rest of the name is just a time stamp indicating the time of creation. Inside of each folder there are four files, which are described below.

The data will be arranged in columns and rows where each row corresponds to a task and every column corresponds to a data parameter according to table 2.
In addition to these, a TimelineSolution is created, which exhibits an admissible solution to the problem.

3.1 The Dependencies File (Dependencies.dat)

A dependency is described by a DependencyMatrix element and a DependencyAttribute matrix element as well as a dependency id appended as the first element, as described below. The number of rows equals the number of dependencies.

The first column is simply the dependency id, which is provided for compatibility reasons. The second column is the id of the first task in the dependency, while the third is the id of the second task; these columns correspond to a condensed form of the DependencyMatrix. The fourth and fifth columns are, in the notation established in the appendix or somewhere else, f_d^{min} and f_d^{max}, respectively, or the minimum and maximum times allowed between the end of the first task and the onset of the second task in a dependency; these columns correspond to the DependencyAttribute matrix. In a dependency, the first task has to be completed before the second task begins.

3.2 The Tasks File (Tasks.dat)

The tasks file consists of what is called a TimelineAttribute matrix, which, in turn, consists of five columns, representing the attributes of all tasks (ignoring dependencies) on all time-lines. Each row corresponds to the attributes of a certain task in a time-line. The first column corresponds only to the id's of tasks, which are not really necessary to have, yet are provided here for compatibility reasons. The second column corresponds to the minimum allowed starting times of tasks, while the third column represents the corresponding deadlines. The fourth column represents the time-lines of tasks, while the fifth represents the lengths of tasks.

3.3 The Settings File (Testinfo.dat)

This file contains all data settings that were used when creating the corresponding data set. The parameters saved in this file are, in order, the following.

Parameter	Meaning
dummy2_selector	Which type of data selected for the further simplified mode.
N	The expected total number of tasks
L	The length of any timeline
T	The number of timelines
Num_data	The number of data sets created using the same settings
Ndeps	The total number of dependencies
occupancy	The level of occupancy across all time-lines
constrain	Binary variable telling us if dependencies were constrained to stay within time-lines.

Parameter	Meaning
rectify	Binary variable telling us if dependency probabilities were rectified based upon task lengths
std1	Temporal distribution of dependencies standard deviation. Not used.
std2	Maximum time between dependent tasks standard deviation.
std3	Minimum starting time distribution standard deviation
std4	Task length standard deviation
std5	Deadline distribution standard deviation
std6	Minimum time between dependent tasks standard deviation
std7	Task spacing distribution standard deviation
std8	Task distribution across time-lines standard deviation.
std9	Spatial distribution of dependencies standard deviation. Not used.
mu1	Minimum starting time distribution expected value
mu2	Maximum time between dependent tasks expected value
mu3	Deadline distribution expected value
mu4	Minimum time between dependent tasks expected value
difficulty_number	The level of difficulty of the data created, where 0 and and 100 are the lower and upper limits, respectively.
dist_sel1	Temporal distribution of dependencies. Not used
dist_sel2	Maximum time between dependent tasks distribution
dist_sel3	Minimum starting time distribution
dist_sel4	Task length distribution
dist_sel5	Deadline distribution
dist_sel6	Minimum time between dependent tasks distribution
dist_sel7	Task spacing distribution
dist_sel8	Task distribution across time-lines
dist_sel9	Spatial distribution of dependencies. Not used.

3.4 The AMPL data file (AMPL.dat)

The AMPL data file contains all of the above information, except for the information found in the settings file. The file is written in the special syntax used by AMPL. In particular, the sets being used are defined in the rather idiosyncratic way that one only encounters when attempting to get a grasp of AMPL.