Documentation for an Optimization Algorithm for the Parameters of the Thermal Analysis performed by the Corblivar Floorplanning Tool

Author: Timm Amstein
Student Business and Engineering (Bachelor's degree)
TU Dresden – Institute of Electromechanical and Electronic Design

Contact: timm.amstein@gmail.com

This documentation is designed to give insight and provide guidance for using the optimization script for the thermal analysis algorithm performed by the Corblivar floorplanning tool. This tool provides a multiobjective optimization for the design of functional units (Blocks) on a multilayered 3D-IC.

Table of contents

The thermal analysis concept **2**
Problem definition **3**
Initializing the program 8
The *parameters* function 8
Starting the program **10**
The script *optimization.m* 10
The function *HotSpotData.m* 11
The function *evalCorb.m* 11
The optimization loop 12
Strengths and Weaknesses **12**

The thermal analysis concept

Corblivar uses a 3-Step algorithm to generate a valid floorplan, where every power block has its own defined Space. After the floorplan generation the information will be translated into a 2D power distribution on the chip for further analysis (see fig. 1).

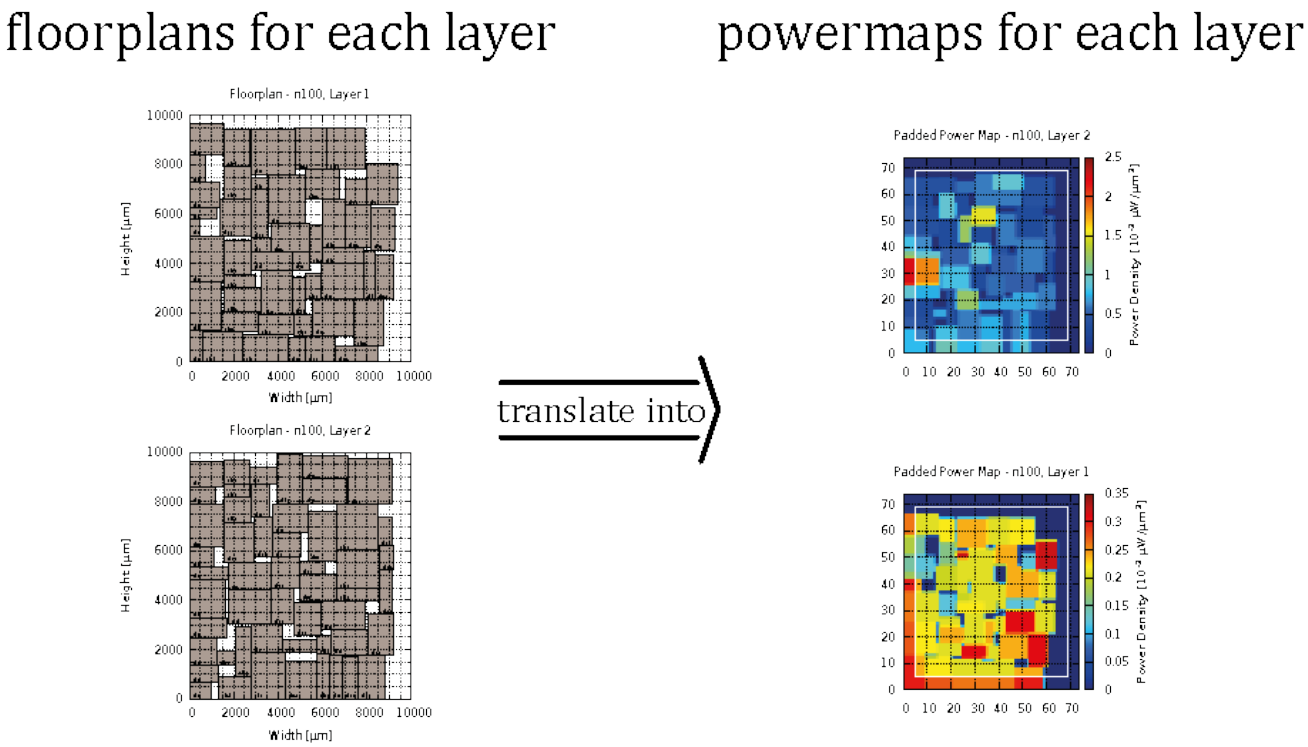


Figure 1 Generating powermaps

To generate the thermal profile of the chip Corblivar uses a 1D convolution approach. The idea is to convolute the power distribution of the floorplan with a previously defined Gaussian distribution. To get a 2D thermal profile the 1D convolution is performed twice (over both dimensions). In previous studies it was shown that you can get a very good approximation to comparable thermal profiles, generated with thermal networks or the finite element method, with considerable improvement in speed (see fig. 2).

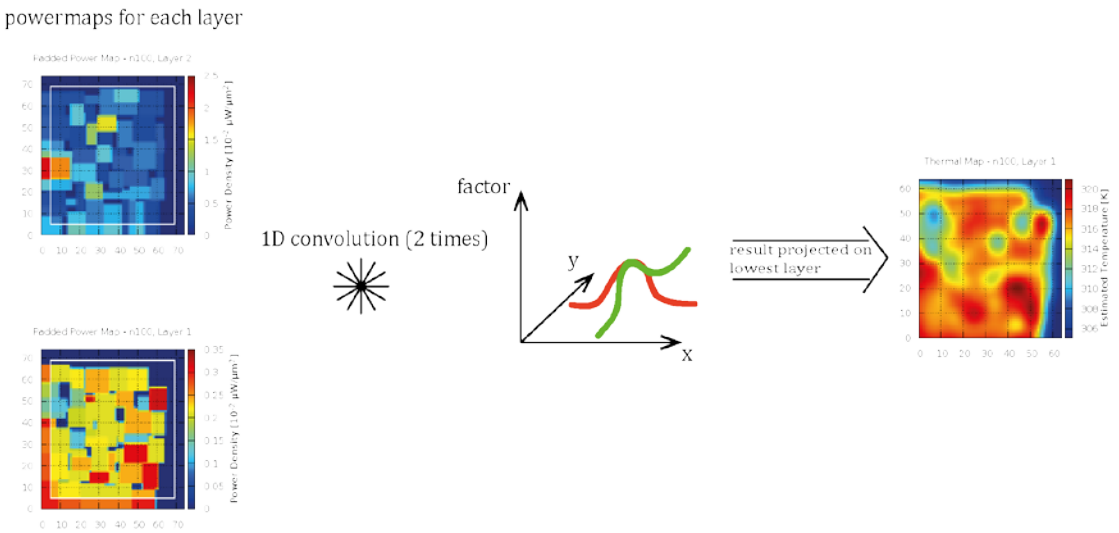


Figure 2 1D Convolution of all powermaps

Problem definition

The biggest disadvantage of the algorithm is the Gaussian function (see fig. 3) which has to be defined the precisely to get good results. The results are **two unknown parameters** that are highly influential. There are multiple ways to define a Gaussian. Corblivar uses the maximum value at the mean as the **impulse factor (I)** and the value at the boundary of the function as the **mask boundary (Mb)** value. The standard deviation is fixed as 1.

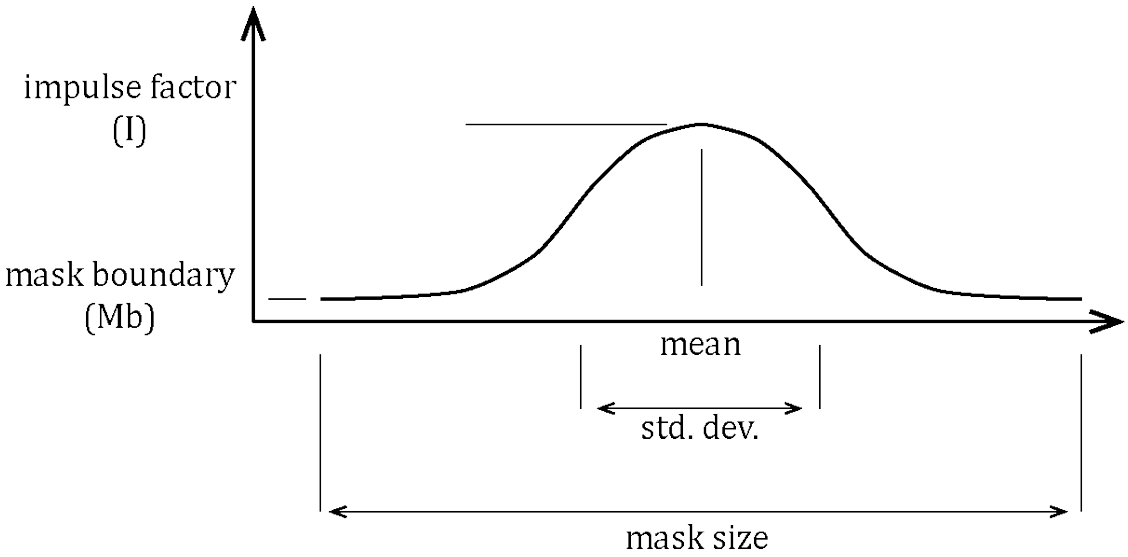


Figure 3 Definition of the thermal mask

Normally the Gaussian has to be explicitly defined for every single layer of the 3D-IC. That would create 4 more unknown variables. Corblivar avoids this problem by assuming that the mask parameters of the different layers are related. It defines an **impulse scaling factor (If)** which defines the impulse factor for every layer above the bottom layer as

$$I_{layer} = \frac{I}{layer^{If}}$$

Nevertheless, this leads to a **third unknown** mask parameter.

The **fourth** thermal mask parameter defined in the configuration file of Corblivar is a scaling factor that influences the **power density in the padding zones (PDPZ)** of the power maps. This parameter takes into account that at the boundary of the chip, because there is a huge difference in thermal conductivity from silicon to air, the heat can't get to the heat sink as fast as in the middle of the chip. To compensate this effect the power distribution is scaled up at the boundaries.

The **fifth** and last unknown parameter of the thermal masks is the scaling factor of the **power density in TSV regions (PDTR)**. The TSV (Through Silicon Vias) are an irregularity in the thermal profile of a chip because of their properties that differ from silicon. Through experiment there was shown that the effect of the TSVs could be simulated by assuming that at the TSV position all the power can be distributed to the upper layers. And at points nearby the effect would be similar but not as strong as directly at the position. The assumption of Corblivar is that the effect decreases exponential (see fig. 4). By defining the power at the TSV position as 0, PDTR only defines the steepness of the exponential function.

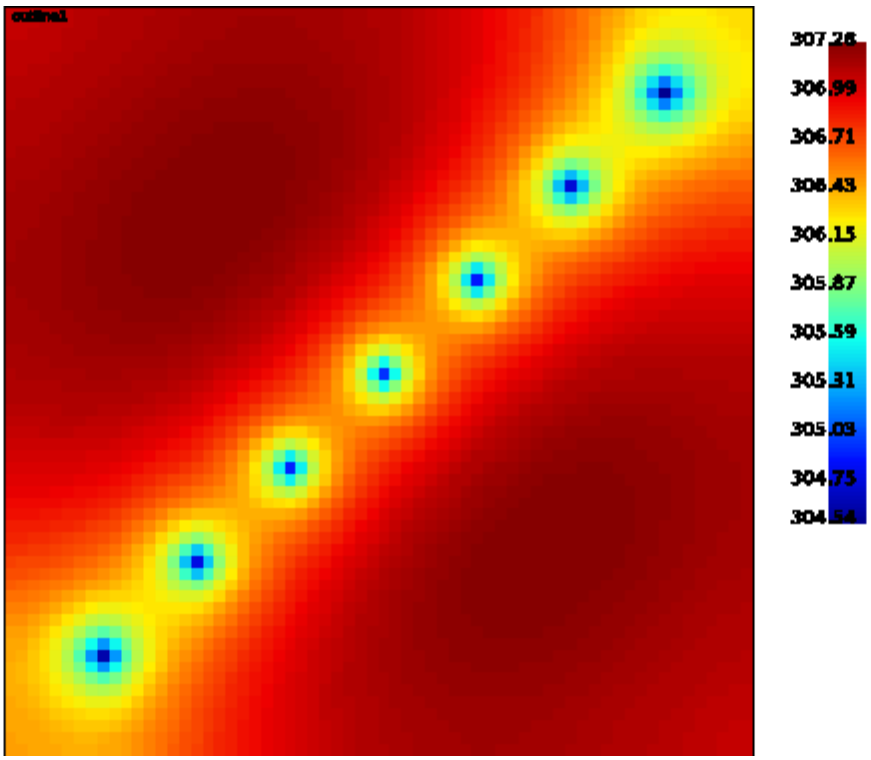


Figure 4 Thermal Map from HotSpot Analysis for the lowest layer of a chip with a uniform power distribution with TSVs placed on the diagonal

The optimization problem consists of **5 unknown variables** with the goal to get a **minimal difference** between the thermal map generated by the convolution approach and an exact solution generated by a thermal network (generated with HotSpot 5.0).

The 5 variables are all related to one another in an unknown way. Because of that it is not possible to build up a multidimensional objective function to prove minimality mathematical. It is also important to notice that the effect of the different parameters depends a lot on the power distribution.

Optimization Algorithm

For advanced functionality in data manipulation and because of the big library of available predefined functions, all the optimization scripts use the **Octave environment**.

Therefore you will need the Octave to run the program.

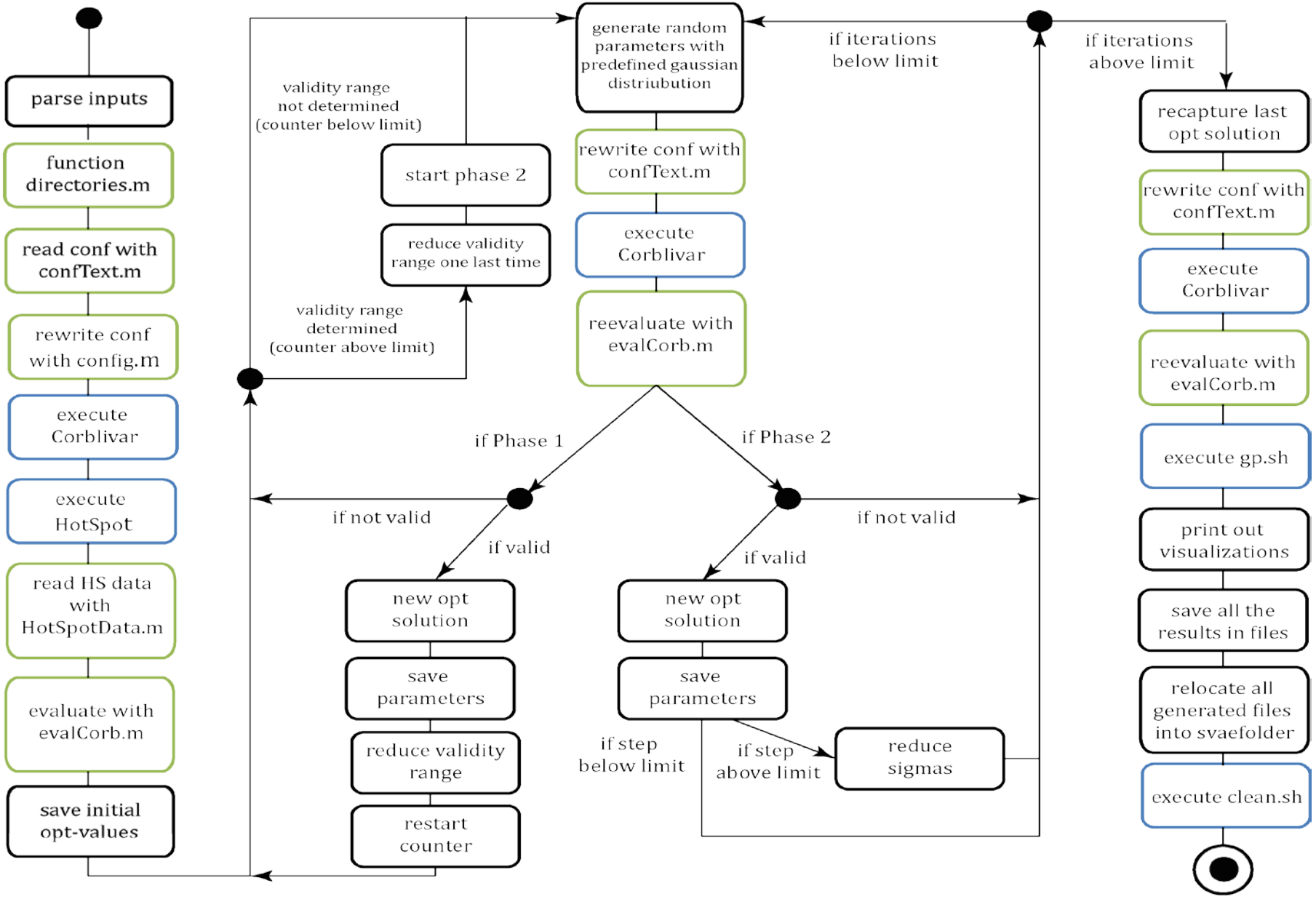
If you don't have octave already installed on your device use

sudo apt-get install octave for Linux

or download the program from your favourite source.

Because of the undefined relationship between the parameters, the optimization relies on a guided random search algorithm. The parameters will all be defined separately with a random function. This functions use predefined boundaries and a standard normal distribution. These parameters will be implemented in the thermal analysing algorithm of Corblivar so the new thermal profile can be generated. After that, HotSpot defines the best solution via termal network analysis.
The both solutions will be evaluated after that, with a two-step algorithm. Before evaluating the error at every point of the map the algorithm checks both thermal maps for local maxima and the *HotSpot* - Data for local minima.
The first evaluation checks if the error at the local extreme points is in a predefined valid range. If that's the case the set of parameters will be set as valid and the parameters will be saved as the new optimum. After that the validity range (*p.opt.accuracy*) will be reduced automatically until the optimal parameters can't create a thermal map where all the extreme points are in valid range anymore. If that happens new random parameters will be generated until a new valid solution and therefore a new optimal solution was found. Through this process the range of accuracy will be minimized. If there isn't a new optimal solution at half of the predefined iteration count, phase 1 of the optimization has ended and the minimal accuracy has been defined as the reduction of the last accuracy that was met by the optimal configuration.
After that the new iteration scheme of phase 2 begins. The the evaluation is based on the error values. For every positive error outside the range of accuracy the value of 6 is given (Corblivar overestimates the temperature of the chip in comparison to HotSpot). For every negative error outside the range the value of 4 is given (Corblivar underestimates HotSpot which is not as bad as overestimation). If the error is inside the range of accuracy the evaluation is set to 0. To find the best solution, the sum of the evaluation matrix must be as small as possible.
Because step two is built as refinement for phase 1 the standard deviation of the random parameter creation functions will be reduced after one tenth of the predefined iteration count. After ten refinements the best possible solution was found. The program will recreate that optimal thermal analyser run, safe all the data into a predefined folder and will tidy up the working folder after that. For visualization of the whole process see figure 5.

The Process:


parse inputs
function directories.m
read conf with confText.m
rewrite conf with config.m
execute Corblivar
execute HotSpot
read HS data with HotSpotData.m
evaluate with evalCorb.m
save initial opt-values
generate random parameters with predefined gaussian distriubution
rewrite conf with confText.m
execute Corblivar
reevaluate with evalCorb.m
validity range not determined (counter below limit)
start phase 2
reduce validity range one last time
validity range determined (counter above limit)
if Phase 1
if Phase 2
if not valid
if valid
new opt solution
save parameters
reduce validity range
restart counter
if not valid
if valid
new opt solution
save parameters
if step below limit
if step above limit
reduce sigmas
if iterations below limit
if iterations above limit
recapture last opt solution
rewrite conf with confText.m
execute Corblivar
reevaluate with evalCorb.m
execute gp.sh
print out visualizations
save all the results in files
relocate all generated files into svaefolder
execute clean.sh

Initializing the program

Before actually starting the program, please make sure you provide the most recent version of

Corblivar and
HotSpot.

Also make sure that the programs are compiled properly and you provide all the benchmark files you need to use Corblivar. After that, check the optimization folder for completion. Inside the folder there has to be:

optimization.m
parameters.m
directories.m
confText.m
config.m
HotSpotData.m
evalCorb.m
extrema.m

If all these prerequisites are given, please open the configuration sheet for defining the Corblivar floorplanning algorithm. If not defined otherwise, it is located inside the *exp* - folder of the Corblivar program folder and is named **Corblivar.conf.** The file can be relocated freely.

The *parameters* function

The parameters function is used to predefine the configurations of the optimization algorithm.
This file should be manipulated after defining the Corblivar configuration file.
With that function you can define all the important parameters used to configure the algorithm.

Accuracy: (recommended between 10 and 20)
With that value you can define the validity range for phase 1. If you choose that value very high the possibility that phase 1 takes longer is high. If you choose a value to low it is possible that the random search can't find any optimal solution that meets the conditions.

Iterations: (recommended between 50 and 200)
That value defines how many iterations the algorithm will take until you get a result. It defines

also the ending of phase 1 (half of the iteration count after the last new valid solution) and the number of iterations done in phase 2.

Step: (recommended denominator between 5 and 10)
Step defines the number of iterations performed between the refinements of the parameters. If you choose that value to be too low the effect of the refinement process will be very low. Please notice that the parameter is related to the iterations so the important value to define is only the denominator.

Update factor: (recommended between 0.8 and 0.95)
The update factor is used to reduce the standard deviation after one step in phase 2. It is also used for reducing the accuracy after a new optimal solution was found. It basically says that after the update the value is only 90 % (when set as 0.9) of the value before. Please recognize that if you choose the value too small the update on the standard deviation and on the accuracy will be too large to get a good result.

Temperature offset: (recommended 293)
The Offset defines the basic start value of the temperature. Because Corblivar only defines the temperature difference it chooses 0 as basis, while HotSpot uses 293K (room temperature). Because of the improved performance this value will be redefined automatically as the minimal value of HotSpot inside the optimization algorithm. So changing the value doesn't have a big effect. It is recommended to just leave it at 293.

Define initial mask parameters:
The mask parameters are initialized in this script. For every parameter, you can define the initial value (also defines the mean of the distribution) and the initial sigma. The sigma is the value plus/minus the mean where ~60% of the random values are included. The range of these parameters differs very much for different benchmarks, so it is not possible to really recommend a range. Nonetheless there are some rules that must be followed.

All of the values are nonnegative!
The initial value of I must always be bigger that the initial Mb.
The initial PDPZ must be bigger than 1 and the max value shouldn't be bigger than 2.

If you don't follow these rules it is highly possible that the initial Corblivar floorplanning run will fail and because of that, there will be no solutions at all.

Starting the program

To start the optimization, open the terminal go to the directory where the *optimization.m* file is located and type:

octave optimization.m benchmark dir/NameOfConfigurationFile.conf DirectoryOfBinary

This bash command will start the process. If you use the standard configuration of Corblivar and got the optimization files in a folder which is located in the same directory as the Corblivar binary you don't have to define the directory of the binary. If you use the *Conf* file inside the *exp* folder of Corblivar you also don't have to define the directory, just the name of the .conf file. Of course you can use all of them differently if you wish.

The script *optimization.m*

The script *optimization.m* serves as guidance and could be called the main program. All the other files are interactive octave functions with input and output parameters. All the functions are designed so they will output reusable data structures that combine many important parameters.

The optimization scripts starts with clearing all the data from previous sessions. It does that inside octave and also via the *clean* shell-script inside the working directory.

It also parses the different directories of the Corblivar-binary, the *conf* file and the working as well as the experiments directory, where the safe folder will be located. All of that will be done by the function *directories.m*. The directories function needs the arguments given by the user via terminal.

After all the locations are defined the parameters will be parsed via *parameters.m.* This function doesn't need input and outputs a data structure *p* which includes all the predefined parameters.

When the parameter function has finished the safe folder will be created and the time counter starts.

The next step is the reading of the Corblivar *conf* file which holds important information and parameters for the optimization, most importantly the layer count.

That is done by the *confText.m* function which needs the *conf* (Corblivar configuration parameters) and the *p* (optimisation parameters) data structure and adds something to them. It basically reads the Configurations file line by line builds up a string array and saves the layer count. After that the Configuration text exists inside octave and doesn't have to be read from the text file again.

After reading it, the text-file must be rewritten with the new parameters defined by the user. That is done by the *config.m* function. That one uses the red text of *confText.m* and compares the strings with different sample strings. If one of them fits it uses a line counter to change the numerical value two lines below the string. Because of that the overall design of the *conf* file is predefined. The value is always two lines below the description. *Config.m* creates no octave output because it only rewrites the *conf* file of Corblivar.

After the rewriting, the initial Corblivar floorplanning will be performed. To initiate this program the system command of octave is used. This imitates the bash commands in terminal.
The next step is the use of HotSpot to get the exact solution. HotSpot can be performed via the *HotSpot.sh* shell script which is provided by Corblivar. That shell script can also be called via system command. After the HotSpot run, the minimum temperature will be saved as the new basis for Corblivar by the *HotSpotData* function.

The function *HotSpotData.m*

This function uses the parameter *bench* and the structure *p* to create the structure *HS* where all the information of the HotSpot Analysis is located.

At first the HotSpot thermal map of the bottom layer will be parsed as a vector and the absolute minimum and maximum values will be saved separately. Following that the Data will be structured into a matrix which represents the chip. After that the matrix will be plotted in a 3D field. This procedure will be done for all the different active layers defined by the *conf* file. After creating the plots the maps will be searched for local maxima and minima.

That is done by the function *extrema.m* which uses the thermal map, a defined value for the boundaries and a Boolean, for defining if it has to look for minima (0) or maxima (1). The function creates a list as well as a map with all positions of extremes.
It searches for extremes by stepping through all the points and comparing them to their neighbours. If the search is for minima the value must be smaller or equal and if it is for maxima the value must be bigger or equal.

After both programs created their thermal maps the first evaluation is performed by the function *evalCorb.m*.

The function *evalCorb.m*

This function does the core work of the algorithm. To do that it uses the *p*, and the *HS* data structure as well as the parameter bench, so it can output the *eval* and the *Cbl* structure.

At the beginning the Corblivar thermal map data is parsed and the minimum and maximum values are saved. After that it is reshaped and restructured to have matching positions with the prepared HotSpot matrix. The *extrama.m* function is also used here to find the local maxima in the Corblivar map.
After that the interpretation of the solution begins. The error matrix is built by taking Corblivar minus the lowest layer of HotSpot. It is important to notice that the whole evaluation focusses on the lowest layer, which is the critical one. That’s because it is the one most far away from the heat sink.
The error matrix shows the real difference between Corblivar and HotSpot. For further evaluation of solutions it is also important to build the average error and average squared error. After that the evaluation matrix will be generated like mentioned before. Everything positively bigger than the accuracy, gets a penalty of 6 and everything negatively, gets a penalty of 4. The validity check will also be performed by the function.

After all that, the history data and also the first optimal values are saved. That data will be saved at the end for further analysing.

Now the optimization loop starts.

The optimization loop

The loop starts with the random generation of all the five different parameters. That is realised is separate do-until loops which enforce the boundary conditions. After the new parameters are generated the *conf* file is rewritten with *config.m*. With the new *conf* file, Corblivar will do a rerun with the given solution and without printing to the terminal. After that a new evaluation is done with *evalCorb.m* to check if a new valid or optimal solution was found. If its phase 1 the next step is to see if a new valid solution was found and if that's the case, to update the accuracy and revaluate with *evalCorb.m*. If it is still valid, it has to be updated again. Every new found valid solution will also be seen as a new optimal solution because the range of accuracy is always the better than at the last valid solution. After phase 1 the minimal accuracy will be defined. It has to be one step of accuracy below what the best optimal solution could accomplish, because if not the phase two can't use its full potential. The reason for that is that the valid solution has a very good value at the latest accuracy level and to get something better you have to get a value below that. If you refine the evaluation by diminishing the accuracy the comparison will be much better.

At the end of the loop the program reloads the optimal parameters out of the data structures and uses them to recreate the Corblivar plots with the shell script gp.sh. It also uses the *evalCorb.m* again to regain the individual data.
After that a lot of plots are created, like the matrix error, the evaluation matrix and the maximum points.
In the end everything that could be important for the user, will be saved in files and copied inside the predefined safe folder.
After that the working directory will be cleaned.

Strengths and Weaknesses

The algorithm has a solid basis and creates valuable results and values that can be reused in similar experiments. The biggest time problem is the HotSpot Analysis that has to be done at least once to get a solution. The algorithm is designed very flexible and works for all benchmarks that are defined properly. Also through the two phase approach the mistakes made are minimized. A problem is that especially in phase 2 the number of new optimal solutions is not very high. It might be possible to use the whole algorithm without phase 2. That would be a tremendous timesaver but also a reduction in accuracy. In the future, this algorithm could be uses to train other advanced machine learning applications to recognize patterns or structures which can be used to understand mathematical relationships. With such trained programs, if they are well designed, it should be possible to get a good approximation to the optimal solution in a matter of seconds without running the optimization script itself.