Combining the Combigrid method with the SG++ data-mining pipeline

Guided Research

Nikolaos Ioannis Bountos
Faculty of Informatics
Technische Universitaet Muenchen
Email: mpountos@outlook.com.gr

***Abstract*— In this guided research we attempt to extend the data-mining pipeline of SG++ to support the estimation of probability density functions using the combination technique. The intuition behind this paper is to employ the combination technique for the purpose of breaking down the main problem into multiple smaller tasks aiming for computational speedups. We expect this to majorly improve the performance of the current probability density estimation in the SG++ project, as well as create further opportunities for efficiency boosting via parallelization. We observe quality approximation of the original method and promising speed performance that creates a roadmap for future improvements. We present the performance of the method for grid levels 3 and 5 as well as for threads 1 and 4. We provide a comparison with the SG++ datadriven miner[1]. From our experiments, we conclude that the proposed method will provide the appropriate speed improvement when executed in a large number of threads. This comes as a consequence of the number of independent processes attempting to run on parallel in comparison to the provided number of threads. To be more specific, the maximum number of threads provided is 4 when the number of parallel processes can reach a few hundreds. This adds further overhead in the synchronization and does not utilize the independence of the sub problems.**

***Index Terms*— Probability Density Function, CombiGrid, Sparse Grids, Combination Technique, SG++**

I. Introduction

In this research project we attempt to utilize the combination technique, implemented in the combigrid module of SG++ project, in order to solve probability density estimation problems. In order to understand the combination technique we have to examine the core idea behind it, the **Sparse Grids**. The Sparse Grid [1] is a discretization method for multivariate problems. The ultimate gain from the use of Sparse Grid techniques is the computational speedup. This is a consequence of the heavy reduction of grid points when compared to full-grid techniques. To be more specific, Sparse Grids require only $O(N(logN)^{d-1})$ grid points, where d represents the dimensions and N the number of grid points. Additionally, despite this reduction in grid points, the accuracy is not affected significantly in comparison to a full-grid approach. [4]

[1]In this paper we will refer to the current PDF method of the SG++ project as " datadriven miner "

A. Sparse Grid Construction

The main component for the creation of a sparse grid is the multilevel basis. As the purpose of this paper is not to analyze the sparse grids, we will only present the 2 dimension case.

The main basis function used for sparse grids is the hat function $f(x) = \begin{cases} 1-|x|, \, if \quad x \in [-1,1], \\ 0, \, else \end{cases}$.

Then we create a set of equidistant grids $\omega_l \in [0,1]$ of level l and mesh width 2^{-l} . The grid points are defined as : $x_{l,i} = i * h_l, 0 \leq i \leq 2^l$. [4]

We then consider the family of functions :

$f_{l.i}(x) = f(2^l * x - i)$.

In order to create a sparse grid we have to put together the subspaces defined by these functions.

Let's call W_l the subspace at level l. Then, $W_l := span(f_{i,l}, i \in I)$ where I is the index set: $I_l := i : 1 \leq i_j \leq 2^l_j - 1, i_j odd, 1 \leq j \leq d$. [4]

By combining the subspaces W_l we get a Sparse Grid of level n, where $l_j \leq n + d - 1$.

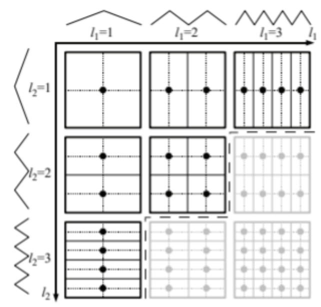

Fig. 1. Level 3 Sparse Grid

In Figure 1. we see the linear basis of level 3.

The basis functions can be extended to the d-dimensional case with the tensor product approach :

$f_{\vec{li}}(\vec{x})) := \Pi_{j=1}^d f_{l_j,i_j}(x_j)$, where $\vec{li}$ d-dimensional multi-indices for each dimension. [4]

Now that we defined the Sparse Grid method, we can employ the Sparse Grid Combination Technique in order to solve function interpolation efficiently. This technique along with the probability density estimation will be explained in the following sections.

II. Combination Technique

In order to define the combination technique we have to define a problem that needs to be solved. In this case we will consider the partial differential equation $Lu = f$ over the unit cube $\Omega = [0, 1]^d \subset R^d$ on which, the technique was originally defined.

Problems of this type are usually solved using discretization techniques. By solving this with a normal full grid with mesh size $h_n = 2^{-n}$ in both directions (we use $d = 2$ for convenience) we end up in a linear system $L_{n,n}u_{n,n} = f_{n,n}$. The error of this solution is quite small ($O(h_n^2)$. However, this kind of computation is proportionate to the amount of grid points. To counter this issue we could use a sparse grid, as mentioned in Section A. To achieve this, the function of interest is discretized on a sequence of grids. The linear combination of these is the sparse grid representation.

To specify, let $\Omega_{i,j}$ be the uniform grid with mesh sizes $h_1 = 2^{-i}$ and $h_2 = 2^{-j}$. The linear combination of these grids is defined as :

$u_{n,n}^{Comb} = \Sigma_{i+j=n+1} u_{i,j} - \Sigma_{i+j=n} u_{i,j}$, $i, j \in [1, n]$. [2]

This approach reduces the computational cost heavily as it decreases the unknowns from $O(h_n^{-2})$ to $O(h_n^{-1} log(h_n^{-1}))$ while the error is bounded : $e_{n,n}^{comb} = O(h_n^2 * log(h_n^{-1}))$ [2]

The combination of the uniform grids can be seen in Figure 2.

For higher dimensions, we would have to decompose the finite element space that u is defined on as $S_{i,j} = \Sigma_{s=1}^{i} \Sigma_{t=1}^{j} T_{s,t}$. $T_{s,t}$ represents the subspace of $S_{s,t}$ that does not contain the grid points of $S_{s-1,t}$ and $S_{s,t-1}$. Then, $\forall u \in S_{i,j}$ is represented by : $u = \Sigma_{s=1}^{i} \Sigma_{t=1}^{j} u_{s,t}$, where $u_{s,t} \in T_{s,t}, s = 1, ..., i$ and $t = 1, .., j$ [2].

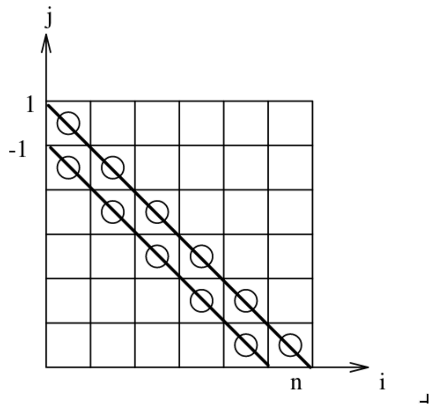

Fig. 2. The combination of the interpolants in the full grids of the Sparse Grid space with i+j=n and i+j=n+1.

In this work, we attempt to utilize the combination technique to efficiently solve the probability density estimation problem.

III. Probability Density Estimation

Probability Density Estimation is a problem where we try to reconstruct the density function from a given dataset. This problem is already tackled in the SG++ project in the datadriven module. The probability density estimation problem can be expressed as : $argmin_V \int_\Omega (f(x) - f_\epsilon(x))^2 dx + \lambda||\Lambda f||_{L^2}^2$,where $\lambda||\Lambda||_{L^2}^2$ is a regularization term and $\lambda > 0$ for smoothness. The optimization problem can then be transformed to :

$(R + \lambda C)a = b$, where C is a regularization matrix, $R_{ij} =< f_i, f_j >$, $R, C \in R^{NXN}, b \in R^N$ and N is the number of basis functions. This means that the number of unknowns depends only on the number of grid points. Additionally, for simplicity C can be set to the identity matrix. Thus, we end up with the following system of linear equations: $(R+\lambda I)a = b$ [3] where R , I and b are fixed. Additionally, λ is set manually to the value that allows the probability density estimation to generalize best to the test data. These allow us to perform offline steps for efficiency.

At this point, we employ the combination technique. We attempt to solve multiple probability density estimation problems using the datadriven module from the SG++ project on the (full grid) subspaces of the Sparse Grid.

IV. Implementation

A. PDFFitter

For this task we created a fitter in the standards of the SG++ project. It provides an object to the module that inherits the functionality of the standard SG++ fitters.

The fit function handles the online and offline phases of the density estimation sub problems. It creates the full grid and fits the model efficiently by splitting the computation in an offline and an online step. As parameters, it requires a $dataset$, a $Grid$ object and a $MultiIndex$ containing the levels of the grid for each dimension. The grid is then constructed in the $buildGrid()$ function.

1) Online-Offline scheme: As mentioned in section III, the probability density estimation problem can be expressed as $(R + \lambda I)\alpha = b$. Given that $R + \lambda I$ are fixed we can look for a decomposition PQ such that $PQ = R + \lambda I$. This step is the offline step and happens before training. The reason for the existence of this step is the expected speedup in the online (training) phase, where we use the already computed PQ in order to solve the system.

B. PDF Combigrid

The main functionality of our contribution is being driven by the class $PDFCombigrid$. It is the core model of this process.

The PDFCombigrid class is responsible for initializing and combining the models for each subspace. In order to achieve this, the "$Combigrid$" and "$Datadriven$" modules are combined. The combination happens by creating an "$Operation$" for the combigrid module to execute, and a "$GeneralFunction$" to provide the functionality of this operation. In this general function the datadriven miner is called.

Listing 1. Operation Definition

```
auto operation = std::make_shared<sgpp::combigrid::
    CombigridOperation>(
grids, evaluators, levelManager, gf, exploitNesting);
```

In the above snippet we see the definition of an operation which makes use of a "$GeneralFunction$" gf. This function contains the core functionality that is applied in each subspace of the grid. The steps that this process follows are:

1) Create a full grid depending on the input (subspace) of the function.
2) Fit a model on the grid.
3) Evaluate the model on every point of the full grid.
4) Store the result in a $TreeStorage$ object.

The results, stored in the $TreeStorage$, are then combined to produce the result of the probability density estimation on a specific point.

This process can be executed both in sequential and parallel mode.

The models trained for each subspace are stored in a hashmap to be used in evaluation time.

The model is being initiated by setting "$combi$" : 1 and then "$threads$" : $numberOfthreads$ inside the Grid configuration.

C. Supporting code contribution

As mentioned, the grid is constructed in the function $buildGrid()$. This function is meant to extend the functionality of the $ModelFittingBase$ class by combining an extension of the $HashGenerator$ class to the existing implementation. To be specific, a function $full$ has been added to the $HashGenerator$ class, that constructs a grid with different levels for each dimension.

The above functions, along with some additional supporting ones created for the $FitterConfiguration$ as well as for $ModelFittingBase$ are the most critical extensions in the supporting code.

V. Results

In order to test the performance of the proposed method, we used five datasets of dimension $2, 4, 10, 15$ and 20 respectively. The datasets consist of real numbers in the domain $(0, 1)$. We examined the quality and speed of the procedure with different settings in regards to the datadriven module miner.

A. Qualitative Comparison

For the qualitative comparison, we use the $L2$ loss on the 2 dimensional dataset for meaningful visualization. We consider a grid of points in the area $[0, 1)$. In the following graphs we observe the probability density estimation of both procedures on the grid (Figures 3, 4).

We can easily see a similar structure in the results. We chose to use a grid of level 5 for this example. To get a clearer view we present in 5 a loss plot using the $L2$ loss. We mark the outliers with red. As outliers, we define the points where the difference between the estimation with the combination technique and the datadriven miner is larger than 0.01.

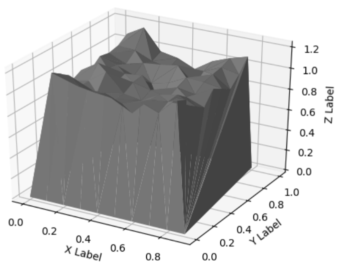

Fig. 3. Combigrid level 5 probability density estimation test on a grid in [0,1]

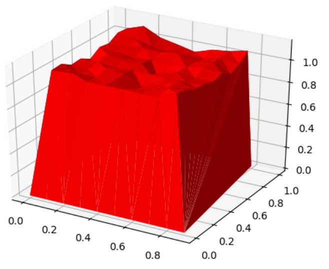

Fig. 4. Datadriven miner probability density estimation on a grid in [0,1]

To showcase the improvement of the quality of the approximation, caused by increasing the level, we present in Figure 6 the $L2$ loss scatter plot for level 3 on the same domain. Again we mark with red the points that have a loss larger than 0.01. We can see that most of the points are well approximated. However, the amount of " outliers " increases.

B. Speed Comparison

An important aspect of our contribution lies in the speed boost of the combination technique and the potential it creates. In this section, we investigate the computational consequences of the increase of the level in the proposed method. Additionally we compare the proposed method to the datadriven miner.

1) Complexity induced by level: In order to examine how the increase of level affects the execution speed of the process, we run the program for level 3 and 5 and measure the average component[2] execution time for the offline and online phase, as well as the full execution time for both parallel and sequential mode. The quality of the estimation can only get better as the level increases (as seen in subsection V-A), but it is obviously more time consuming.

[2]We refer to the independent subspaces as components

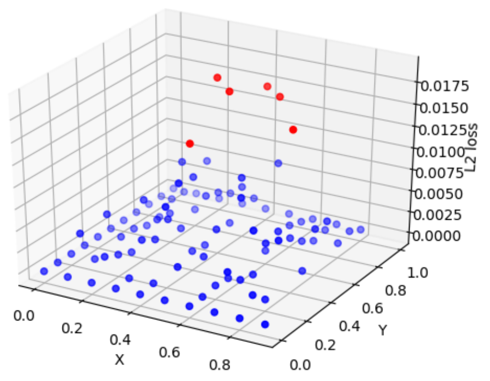

Fig. 5. L2-loss scatter plot for level 5

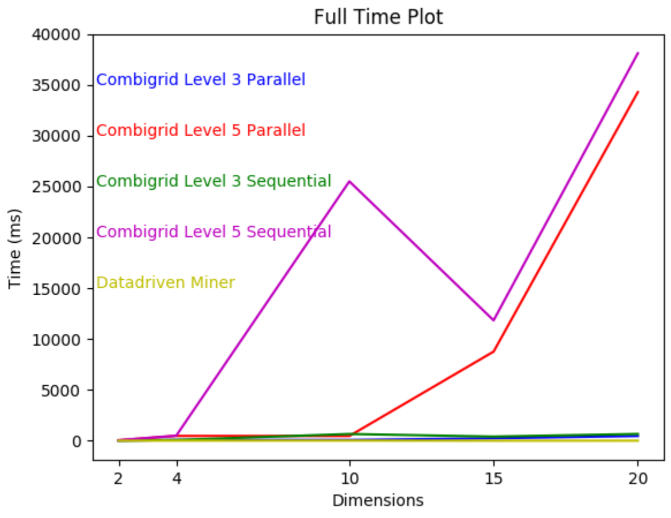

Fig. 7. Full Execution Time comparison

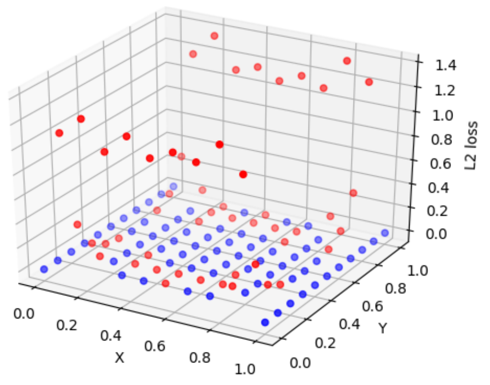

Fig. 6. L2 loss scatter plot for level 3

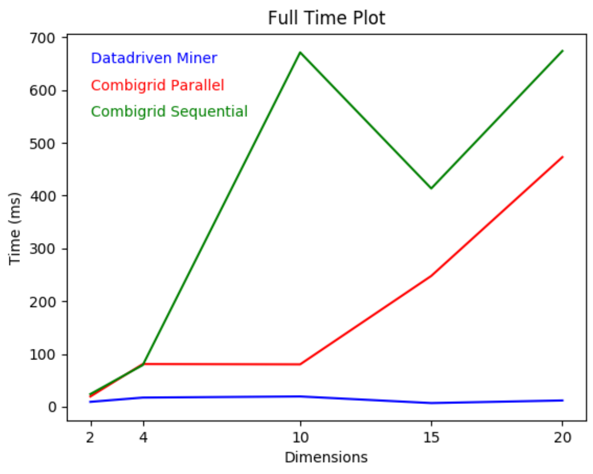

Fig. 8. Full Execution Time comparison for level 3 and datadriven miner

2) Comparison against datadriven miner: In order to compare the previous settings to the datadriven miner, we provide measurements for both the offline and online phases. It is worth noting that for the combigrid experiments, the offline and online execution times are measured as the average time spent for each independent component. As the dimensions and level grow, the number of the components increases. Given that, we would want to see a smaller component execution time in the plot.

Figure 7 compares the overall performance of the processes. For level 5 we see a steady improvement of the parallel mode against the sequential one. Additionally, we see that compared to level 3 and the datadriven miner, the execution time of level 5 is vastly bigger. In order to get a closer look to level 3 and the datadriven miner performances, we present the same Figure wthout level 5 in 8.

In Figure 8 we do not see the expected time improvement against the datadriven miner. Taking this into consideration we inspect further into the performance of the combination technique.

To do so, we examine the performance of each phase (online/offline) against the datadriven miner. For the combigrid version we plot the average time spent for each phase, among the components.

In Figure 9 we see the performance for the offline phase in a sequential setting. The processes are comparable until dimension 4. Given that for the combigrid plot we use the average component time, we should consider as true execution time a multiple of the one shown in the graph, since the amount of components grows as the number of dimensions grows. Taking that into consideration, the datadriven miner should be considered preferable.

However, we notice a great deviation of execution times in larger dimensions. If we take into consideration the subpace processes independence, then the combigrid version is in a very favourable position. That is because we can exploit this property with parallelization.

In Figure 10 we see a similar pattern as in 9. Again we consider the datadriven miner preferable. However, the performance on the datasets with many dimensions leave great promises again for the parallel execution of the combigrid version.

In general, the time difference between level 5 and level 3 for the offline phase is not great. However, this is not the case for the online phase.

Starting from Figure 11 we see level 5 going way over the datadriven and combigrid level 3. However, level 3 retains a decent performance. On the contrary, in the parallel mode, level 3 performs way worse than the datadriven miner. However, its performance on the sequential setting shows an opportunity for higher exploitation and optimization of the parallel version.

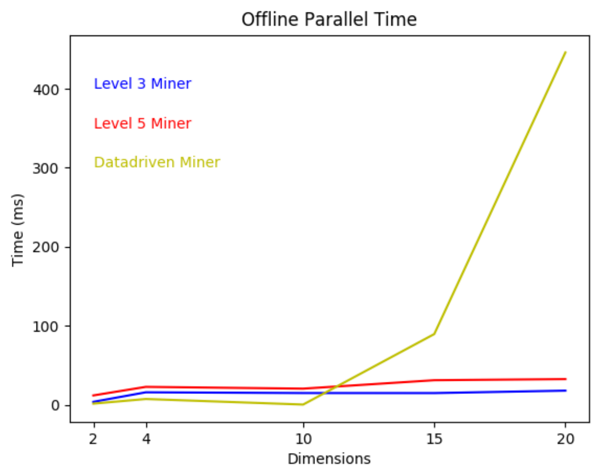

Fig. 10. Offline phase comparison for parallel mode

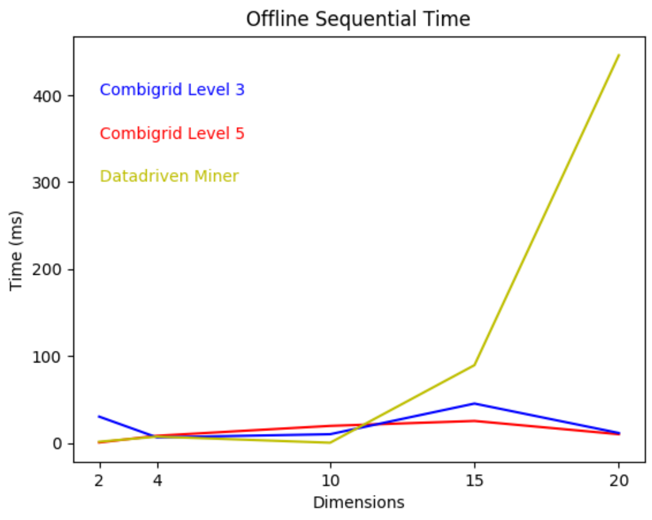

Fig. 9. Offline phase comparison for sequential mode

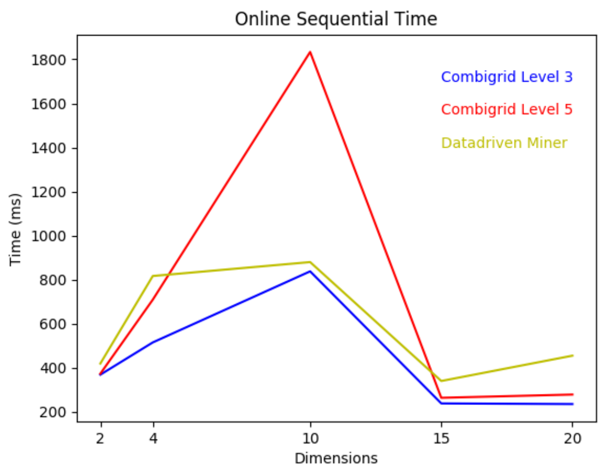

Fig. 11. Online time comparison for sequential mode

The combigrid module's performance on the online phase affects greatly the final execution time, preventing it from achieving the expected improvement in speed. This behavior might be caused by the demand of the process for higher level of parallelization. Additionally, further optimizations can be performed in the offline phase. In particular, we can exploit the nature of the independent components, and correlate their decompositions. That way, we avoid the calculation of multiple decompositions, heavily reducing the offline execution time.

To summarize the results of the experiments, we observed a worse performance of the combigrid probability density estimation compared to the datadriven miner. Especially for level 5 the execution time was discouraging. However, level 3 showed promising results in both the offline and online phases. In the online phase, which consumes the most execution time, it performs better per component when compared to the datadriven miner. Given that the components are completely independent from the others, a higher level of parallelization would produce the desired results. In this setting, only 4 threads were used, which heavily underestimates the program's parallelization capacity.

VI. Conclusion

In this paper, we presented a different approach for tackling the probability density estimation problem. In particular, we utilized the combination technique to split the problem in multiple sub problems. We performed a qualitative comparison of the process with the datadriven miner. We showed results quite close to the results produced by SG++ current method. Furthermore, we compared the different settings of the proposed method in both a qualitative and speed context. The outcome was promising, leaving open space for future improvements. In particular, it showed potential for heavy speedups by optimizing the parallelization and by providing more threads. This comes from the performance of the combigrid in the sequential mode. Given that the

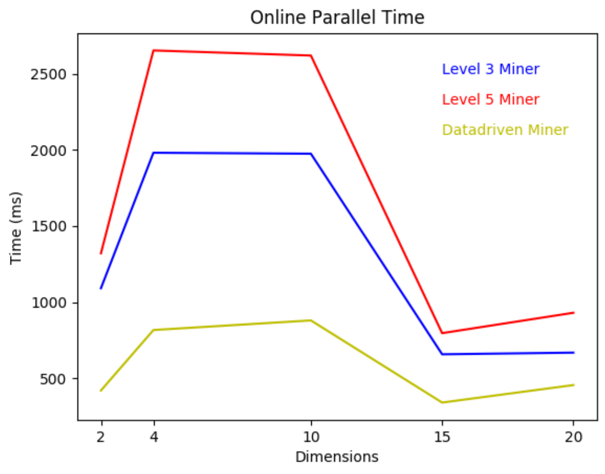

Fig. 12. Online time comparison for parallel mode

amount of components increases vastly as the dimensions and level grows, the level of parallelization used in these experiments could not utilize the independence of the sub problems, resulting in extra synchronization overhead and sequential execution. However, by using a larger number of threads, we would take advantage of the sub problems mutual independence and produce even faster results. Additionally, we could optimize the offline phase by reducing the number of computed decompositions, exploiting the relationship between the components decompositions.

VII. Appendices

A. Datasets

The main dataset used for our experiments is the 10 dimensional friedman dataset. The rest of the datasets were produced by manipulating the base dataset.

The friedman dataset is produced by the following formula: $y(X) = 10 * sin(pi * X[:, 0] * X[:, 1]) + 20 * (X[:, 2] - 0.5) * *2 + 10 * X[:, 3] + 5 * X[:, 4] + noise * N(0, 1)$, where X are independent features uniformly distributed on the interval $[0, 1]$.

B. Experiment Settings

The experiments were conducted on Intel Core i7-7500U CPU @ 2.70GHz x 4 , using all 4 threads for the parallel mode.

References

[1] Jochen Garcke. Sparse grids in a nutshell.
[2] Michael Griebel. The combination technique for the sparse grid solution of pde's on multiprocessor machines. 2016.
[3] Benjamin Peherstorfer. Density estimation for large datasets with sparse grids, February 2013.
[4] M. Griebel. T. Gerstner. Sparse grids. 2008.