Proceedings of BS2013:
13th Conference of International Building Performance Simulation Association, Chambéry, France, August 26-28

ANNUAL HOURLY CFD SIMULATION: NEW APPROACH — AN EFFICIENT SCHEDULING ALGORITHM FOR FAST ITERATION CONVERGENCE

Yue Wang[1], and Ali Malkawi[2]
[1]Software Engineer, Google Inc
[2]Professor of Architecture, Harvard University

ABSTRACT

Computational fluid dynamics (CFD) is one of the branches of fluid mechanics that uses numerical methods and algorithms to solve and analyze problems that involve fluid flows. Annual hourly fast flow simulations are needed for some applications in building industry, such as the conceptual design of indoor environment, or coupled with energy simulation to provide deep analysis on the performance of the buildings. Year round simulation, which consists of 8760 (365×24) independent hourly simulations, is needed to help the designer investigate the problem clearly. However, CFD computation is time consuming, and usually only two or three extreme cases can be simulated in practice. Annual hourly simulation using the traditional method can be considered as a computational intractable problem. Based on previous researches (Yue Wang and Feng, 2012a)(Yue Wang and Feng, 2012b), even though the fast fluid dynamics (FFD) algorithm combined with the General Purposed Graphics Processing Unit (GPGPU) hardware acceleration can make CFD simulation much faster(400x), annual hourly simulation still requires CFD performance to be further improved by 10-20x to make it practical. In this study, a minimal spanning tree based scheduling algorithm is developed, which always gives the best CFD simulation strategy that reuses previous calculated results to generate new results, thus making iteration convergence much faster. It is shown in the paper that the annual hourly simulation by GPGPU accelerated FFD by using this new algorithm requires a similar simulation time to the one used to perform two or three extreme cases of simulation using the traditional method.

INTRODUCTION

Annual hourly simulation is important in many applications in the building simulation field. The building envelope changes its control parameters according to the outdoor condition and internal load, and both change rapidly throughout a year. The amount of solar energy received is the main cause of the daily and yearly temperature variations. These temperature variations also create forces that drive the atmosphere in its endless motions, which also changes the outdoor humidity. Indoor air control system relies on outdoor temperature and humidity values and calculates the enthalpy from it, and thus makes the next hour's control strategies. Moreover, occupants' activity will also be changed according to the daily and weekly schedule. As a result, the hour to hour the indoor condition is entirely different. This is the reason why most simulation algorithms in building simulation, such as energy simulation, were all performed annually.

Fast indoor airflow simulations are necessary for designing building emergency management systems, the preliminary design of sustainable buildings, and modeling real-time indoor environment control. The simulation should also be informative since the airflow motion, temperature distribution, and contaminant concentration is important. However, CFD computation is usually time-consuming and unable to simulate annual indoor air movement. Thus only one or two extreme cases (such as the hottest hour in the summer and coldest hour in the winter) will be simulated by CFD in most of the design analysis.

Extreme cases will give extreme results. Most realistic hourly results throughout the year will reside in much closer boundaries. Thus most assumptions and conclusions drawn from those extreme cases of simulations will be exaggerated. Moreover, while doing coupling simulation with other types of simulation (for instance, energy simulation or particle simulation) with hourly time step, a few extreme cases can hardly suffice. For instance (Djunaedy, 2005) suggests that CFD should be performed hourly when doing coupling simulation with energy simulation.

This leads to an important problem in the field. Given a geometry with yearly boundary conditions, it is important to speed up the traditional simulation strategies to cover annual simulation (8760 static state case) in a fairly reasonable amount of time and acceptable precision.

Previous paper (Yue Wang and Feng, 2012a)(Yue Wang and Feng, 2012b) provided a tentative three layered framework to optimize CFD to cover annual hourly simulation. Most of the literature was reviewed based on the framework. Based on the hardware acceleration, the fast fluid simulation algorithm can be utilized, it can be seen from the review that some techniques can dramatically improve the performance of the first two layers; however, a good method has not been

found yet for the third layer, which requires more work.
This research provides a solution to the third layer. This research provides a way to utilize various methods and algorithms in order to speed up the simulation, and gives a tentative solution to make annual simulations possible.
The new algorithm is based on four basic observations from previous research (Wang and Malkawi, 2013).
CFD simulation is always time-consuming and it is even harder to cover annual simulation (8760 static state case) in a fairly reasonable amount of time and acceptable precision. The research is attempting to investigate an optimal startegy for annual hourly CFD simulation by reusing previously calculated results. From the four observations in the case study, the following conclusion can be drawn:

- Case reusing is possible since many cases are similar, according to the first observation.
- The relationships discovered in observation two, three and four, give insight to construct a learning algorithm to capture the strong correlation with input difference and iteration steps needed, via the linkage of their output difference, rather than simply using the input-output pair as a testing and training set.

Based on the machine learning model a new scheduling algorithms is developed to speed up by reusing as many existing cases as possible.

MAKE THE BEST SIMULATION STRATEGIES BY REUSING SIMULATION

There are large amount of similar cases for annual simulation. The output similarities of cases can be directly predicted by the input similarities. Then the optimal simulation strategies, which give the list of simulation ordering to ensure the fastest converging speed between every two nearby cases, will turn into a graph-based problem. This research uses the output difference of two cases as the indicator to tell if the two cases are similar enough (which is closely related to the number of iterations it takes according to the previous assumptions). There are 8760 cases, and every two nearby cases have a weight which is the indicator defined before. The concept of minimal spanning tree algorithms in computer science is used to determine the optimal simulation strategy path efficiently.
The methodology is described as follows:

- Simulate a few cases to produce the differences of input and output used in the algorithm.
- A machine learning algorithm learns the cases' difference and generates a model for that. Thus, in the future when given the input difference, the model will be able to predict the output differences.
- The method also use minimal spanning tree algorithm to calculate the best simulation strategies and perform a walk on the graph, interpolate most of the nodes' results without simulation.

Predicting output differences

The simulation procedure will begin with random cases of simulation as a starting point, while at the same time training the machine learning algorithm for preparation of predicting results for other cases in the future. 20-30 cases will be simulated to calculate the difference of the input (400 samples).

For example, given two sets of input parameters, $A = (19\mathrm{C}, 1.5\mathrm{m/s}, 9\mathrm{C})$ and $B = (12\mathrm{C}, 2\mathrm{m/s}, 10\mathrm{C})$, the difference of the two inputs in both cases can be written as $(19\mathrm{C}, 1.5\mathrm{m/s}, 9\mathrm{C}) - (12\mathrm{C}, 2\mathrm{m/s}, 10\mathrm{C}) = (7\mathrm{C}, 0.5\mathrm{m/s}, 1\mathrm{C})$. Then, the difference of the output $\mathbf{v1}$ and $\mathbf{v2}$ is calculated. The final result is given as a percentage of the deviation. After the model is trained and given two inputs, it can *predict* the difference of the output. Because of the assumptions used in this research, this model can be very robust.

For instance, four cases A, B, C and D are chosen after the machine learning algorithm predicts all of the output differences between each pair of cases. Six relationships between each other can be seen: outputs of A and C have 1% difference; A and B have 1%; A and D have 1%; B and C have 3%; B and D have 3%; and C and D have 3%.

Assigning edge values in a graph

A graph is an abstract representation of a set of objects where some pairs of the objects are connected by links. The interconnected objects are represented by mathematical abstractions called vertices, and the links that connect some pairs of vertices are called edges. Typically, a graph is depicted in diagrammatic form as a set of dots for the vertices, joined by lines or curves for the edges. An undirected graph is one in which edges have no orientation.

In the undirected graph, the nodes represent all the cases and there are edges to link any two of the nodes. The weight assigned to the edges are the output differences between two nodes predicted by the machine learning model. Additionally, the output differences in the annual CFD simulation implies the number of iterations from one case to another. The algorithm should ensure that the traversal culmulative edge sum of all the nodes is minimal. There are a number of ways to calculate the optimal route to include all the nodes (the 8760 cases). This is a very classical problem in computer science called minimal spanning tree.

In the above example, an undirected graph with four nodes (see Figure 1) will be constructed, representing the four cases A, B, C and D. The percentage on each edge is the output differences between these four cases.

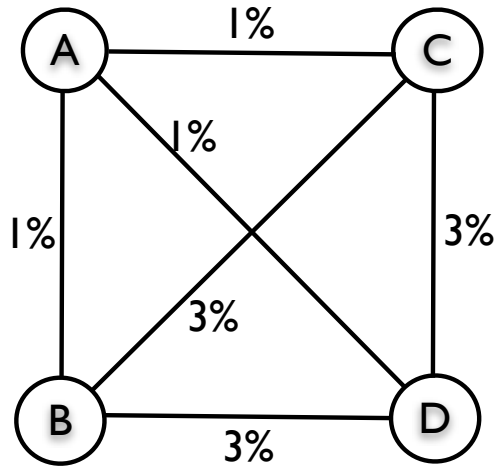

Figure 1: A weighted graph

Strategy planning algorithm

The goal of using a strategy planning algorithm is to visit all of the 8760 nodes (cases) by choosing the minimum sum of edges. Thus, the strategy planning algorithm was turned into a minimum spanning tree algorithm, which is efficiently solved by graph algorithms. Given a connected, undirected graph, a spanning tree of that graph is a subgraph which is a tree that connects all the vertices together. A single graph can have many different spanning trees. Each edge can be assigned a weight, which is a number representing how unfavorable it is, and this can be used to assign a weight to a spanning tree by computing the sum of the weights of the edges in that spanning tree. A minimum spanning tree or minimum weight spanning tree is then a spanning tree with weight less than or equal to the weight of every other spanning tree. More generally, any undirected graph (not necessarily connected) has a minimum spanning forest, which is a union of minimum spanning trees for its connected components.

There are now two algorithms commonly used, Prim's algorithm and Kruskal's algorithm. All three are greedy algorithms that run in polynomial time.

This research uses Prim's algorithm because it's fairly simple to implement and has relatively high performance. To implement Prim's algorithm, this research keeps all the edges adjacent to T in a heap. When the algorithm pulls the minimum-weight edge off the heap, it first checks whether both of its endpoints are in T. If not, it adds the edge to T and then adds the new neighboring edges to the heap. In other words, Prim's algorithm is just another instance of the generic graph traversal algorithm, using a heap as the 'bag'. If this algorithm is implemented this way, its running time is $O(E \log E) = O(E \log V)$.

The implementation can be accelerated by observing that the graph traversal algorithm visits each vertex only once. Rather than keeping edges in the heap, the algorithm can keep a heap of vertices where the key of each vertex v is the length of the minimum-weight edge between v and T (or ∞ if there is no such edge). Each time a new edge is added to T, the algorithm may need to decrease the key of some neighboring vertices. To illustrate the algorithm, consider the graph in Figure 1. This example arbitrarily chooses node A as the starting node. Now the algorithm might progress as described in Table 1.

Table 1: Illustration on Prim's algorithm

Step	$\{u, v\}$	B
Initilization	empty	$\{1\}$
1	$\{A, C\}$	$\{A, C\}$
2	$\{A, D\}$	$\{A, C, D\}$
3	$\{A, B\}$	$\{A, C, D, B\}$

When the algorithm stops, T contains the chosen edges $\{A, B\}$, $\{A, C\}$, and $\{A, D\}$. The pseudo-algorithm is described as follows:

1. Input: A non-empty connected weighted graph with vertices V and edges E (the weights can be negative, or ∞).
2. Initialize: $V_{\text{new}} = \{x\}$, where x is an arbitrary node (starting point) from V, $E_{\text{new}} = \{\}$.
3. Repeat until $V_{\text{new}} = V$:
   - Choose an edge (u, v) with minimal weight such that u is in V_{new} and v is not (if there are multiple edges with the same weight, any of them may be picked)
   - Add v to V_{new}, and (u, v) to E_{new}.
4. Output: V_{new} and E_{new} describe a minimal spanning tree

1. Input: A non-empty connected weighted graph with vertices V and edges E (the weights can be negative, or ∞).
2. Initialize: $V_{\text{new}} = \{x\}$, where x is an arbitrary node (starting point) from V, $E_{\text{new}} = \{\}$.
3. Repeat until $V_{\text{new}} = V$:
   - Choose an edge (u, v) with minimal weight such that u is in V_{new} and v is not (if there are multiple edges with the same weight, any of them may be picked).
   - Add v to V_{new}, and (u, v) to E_{new}.
4. Output: V_{new} and E_{new} describe a minimal spanning tree.

A simple implementation using an adjacency matrix graph representation and searching an array of weights to find the minimum weight edge to add requires $O(V^2)$ running time. Using a simple binary heap data structure and an adjacency list representation, Prim's algorithm can be shown to run in time $O(E \log V)$ where E is the number of edges and V is the number of vertices. Using a more sophisticated Fibonacci heap, this can be brought down to $O(E + V \log V)$, which is asymptotically faster when the graph is dense enough.

The minimum spanning tree can be obtained as the following tree. In such path, all these four nodes could be walked through from the starting point of A and the traversal cumulative edge sum of the nodes is minimum. For a highly complicated graph, the above algorithm is necessary to judge the minimum spanning tree.

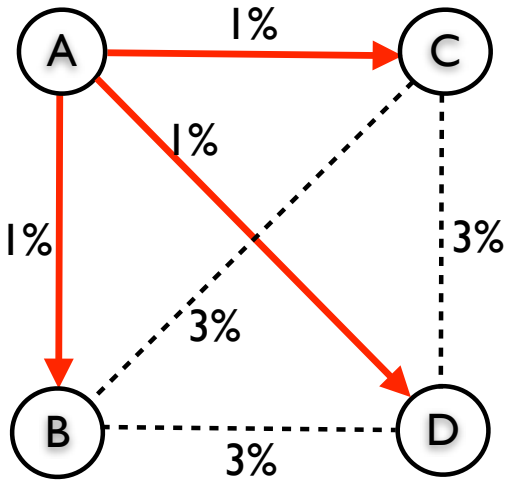

Figure 2: The MST for the weighted graph

Multivariate interpolation

When the minimal spanning tree sequence is obtained, it is difficult to interpolate the new result based on existing results. The interpolation is not a single variable interpolation, because the input variables are a well formed vector containing ventilation speed, temperature and other boundary conditions. However, those values should be treated differently. As mentioned in the assumptions, some variables have less effect on the problem, while others have more. The weight of the interpolation should be assigned in a good way.

Staggeringly, a multivariate input interpolation problem can be reduced to a single variable by simply using the difference numbers predicted by the machine learning algorithm. If case A and case C are predicted to differ x percent, and case C and case B are predicted to differ y present, it's fairly reasonable to interpolate C using case A and B by $(xA+yB)/(x+y)$. Thus, the predicted indicator not only gives precise information to form the minimal spanning tree, but also provides insight for the interpolation process.

However, the output result is multivariate too. Notably, the interpolation parameter for temperature field and vector field should be treated differently. During the machine learning model's training process, given two inputs, the model can be revised to produce the differences of different output parameters (such as temperature field, velocity field) as well; thus, this is one of the power ful aspects of the model. In the machine learning process, independent differences based on different parameters are learned and predicted. Though the combination of these difference values are used in the graph minimal spanning tree algorithm calculation, independent values are used in interpolation. Thus, the predicted value by the machine learning algorithm is a multivariate variable based on temperature and the velocity field.

CASE STUDY

In order to validate the previous planning algorithm and interpolation method work, the following case is constructed.

Figure 3 shows a mechanically ventilated room with floor heating that generated a mixed convection flow. The flow pattern was influenced by both the inertial force and buoyancy force. The case was accompanied with experimental data from (D Blay and Niculae, 1992). The model room of 1.04 mby 1.04 m with supply air horizontally injected into an empty room at a speed. The inlet height and outlet height were 0.018 m and 0.024 m, respectively. The temperature of the floor was 25.5 C and that of the other walls was outdoor temperature. The measured data showed that the flow pattern was two-dimensional. Thus, the FFD(Fast Fluid Dynamics) simulations used two-dimensional meshes. Three different meshes (20 by 20, 60 by 60, and 120 by 120) were used and the results showed that the mesh with 60 by 60 was sufficiently fine. All cases simulated reached convergence.

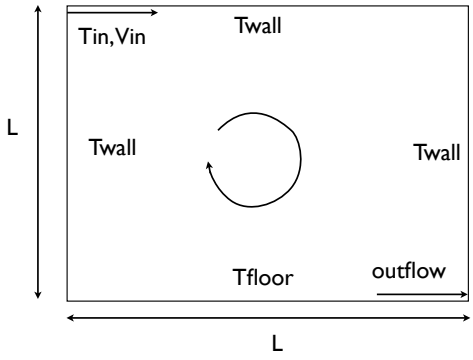

Figure 3: Mixed convection case settings

After the machine learning algorithm predicts the output differences based on input differences between every two cases, an algorithm based on the previously discovered minimal spanning tree algorithm is used to predict the best simulation strategy.

Solving cases that can be directly interpolated

The algorithm will interpolate all the small edges in the graph directly without simulation. For example, case `id` 200 and case `id` 8744 are found by the algorithm linked by case `id` 344. Their outdoor temperature, inflow air speed and ventilation temperature are shown in Table 2.

Table 2: Input parameters for case `id` 200, 8744 and 344

case `id`	outdoor temperature	inflow speed	inflow temperature
200	7.8C	0.674m/s	18.00102C
8744	11.1C	0.666m/s	18.10021C
344	9.4C	0.670m/s	18.00061C

Figure 4 shows the temperature and velocity distribution result for case `id` 200 and case `id` 8744.

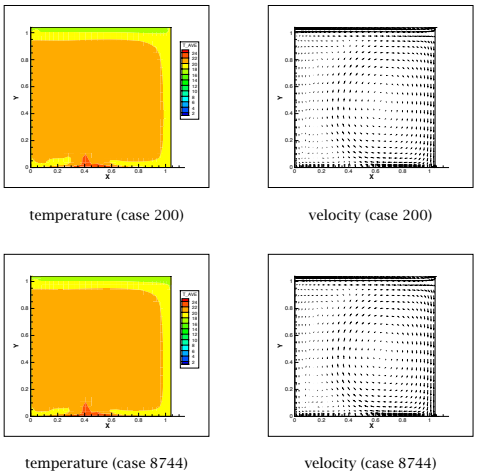

Figure 4: Temperature and velocity distribution for case `id` 200 and case `id` 8744

It is pretty easy to see that case `id` 200 and case `id` 8744 are really similar. The interpolated result of case `id` 344 based on case `id` 200 and case `id` 8744, together with the actual result simulated by the fluid solver, are shown in Figure 5.

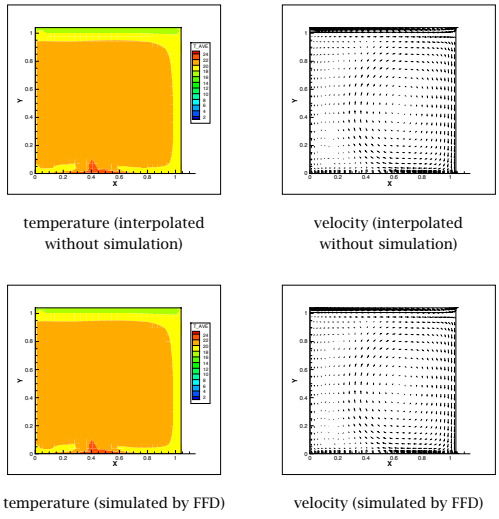

Figure 5: Temperature and velocity distribution for case `id` 344

The interpolated and real results are highly identical. The average difference between the real simulation result and the interpolation result is lower than 1 percent, which can be neglected. This suggests in such scenarios a fluid simulation is not required for case 344; direct interpolation result will suffice.

Simulate certain cases after interpolation

The previous experiment does not conclude any two cases can be interpolated. Below is an example case using three sets of simulation results, which cannot be directly interpolated, and the algorithm needs to use fluid solver to converge the interpolated result.

Case `id` 8430 and case `id` 20 are found by the algorithm linked by case `id` 188. Their outdoor temperature, inflow air speed and ventilation temperature are shown in Table 3.

Table 3: Input parameters for case `id` 8430, 20 and 188

case `id`	outdoor temperature	inflow speed	inflow temperature
8430	10.6C	0.466m/s	21.26164C
20	14.4C	0.465m/s	18.15218C
188	12.8C	0.465m/s	19.72866C

Figure 6 shows the temperature and velocity distribution result for case `id` 8430 and case `id` 20.

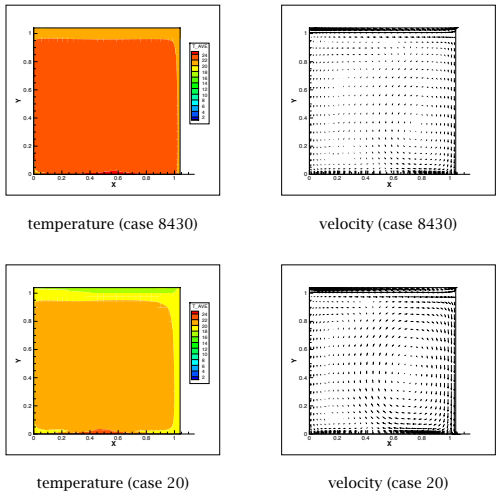

Figure 6: Temperature and velocity distribution for case `id` 8430 and case `id` 20

It is evident that case `id` 20 and case `id` 7430 are not similar. The temperature distribution and velocity field are different as can be seen on the figures. The interpolated result of case `id` 188 based on case `id` 200 and case `id` 8744, together with the actual result simulated by the fluid solver, are shown in Figure 7.

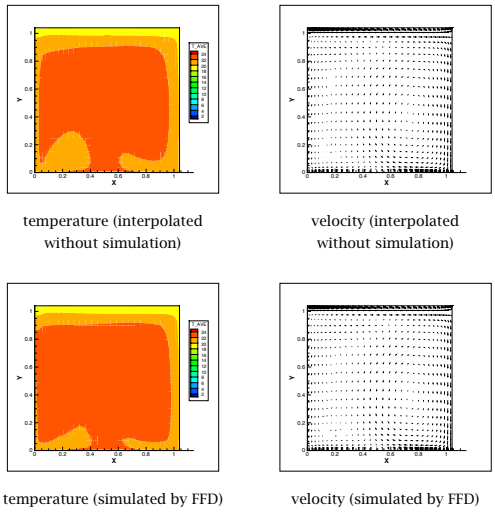

Figure 7: Temperature and velocity distribution for case `id` 188

The algorithm will perform the fluid solver to converge the interpolated result to exact the solution. The averaged difference between the simulated result and the interpolated result is about 13 percent. Converging from the interpolated result takes much fewer iterations (about 3 times faster) than doing from case 20 or case 8430.

Perform simulation on the remaining cases

Certain cases are inherently far away from other cases. For instance, this can be seen in the previously mentioned case whose `id` is 20. Case 1664 is directly linked with case 20. Their outdoor temperature, inflow air speed and ventilation temperature are shown in Table 4.

Table 4: Input parameters for case `id` 20 and 1664

case `id`	outdoor temperature	inflow speed	inflow temperature
20	14.4C	0.465m/s	18.15218C
1664	14.4C	0.65947m/s	18C

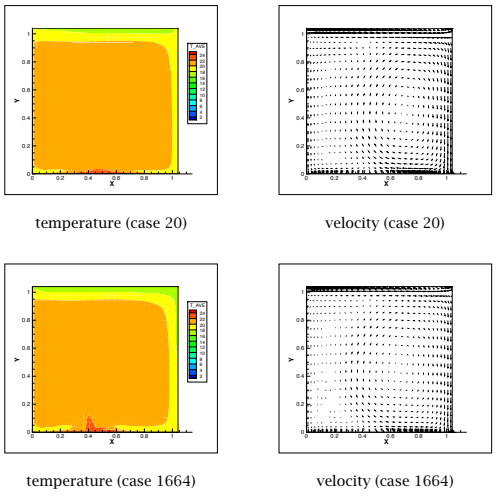

Figure 8: Temperature and velocity distribution for case `id` 20 and case `id` 1664

Since such cases are not similar by prediction, the only thing the solver can do is to converge from case 20 to case 1664. Figure 8 shows the temperature and velocity distribution result for case `id` 1664 and `id` 20 simulated by the FFD solver.

Case 1664 and case 20 were predicted to differ by 34 percent using the calculation method described in Appendix A. By reusing case 20 rather than starting from zero vectors as the initial state, the simulation only saves about 1/6 of the iterations, which is a quite limited speed-up.

CONCLUSION

Figure 9 shows the minimum spanning tree of 8760 cases.

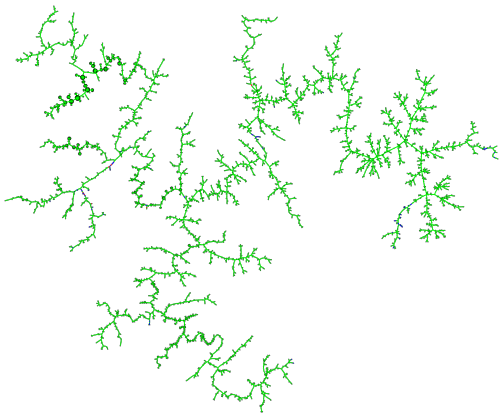

Figure 9: Minimal Spanning Tree for 8760 cases

Figure 10 is a small part from Figure 9. The entire minimum spanning tree shows that most of the points are able to be interpolated; thus, the simulation time is saved greatly. The statistics of the simulation are provided.

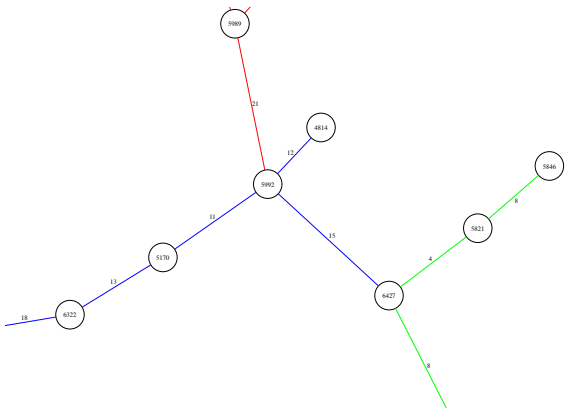

Figure 10: Magified small fraction of the minimal spanning tree

We assign a unique `id` number to each of the simulated cases, based on their hour index in the year. For example, if the `id` equals to 5, this means the case is for Jan 1 at 5AM in the morning. The simulation

uses realistic outdoor temperature, simulated indoor air flow speed and temperature simulated by energy calculation.

The simulation is then performed according to the calculated strategy. A simple breath first search (BFS) or depth first search (DFS) walk can traverse the whole tree efficiently; interpolating all the small edges directly, converging the interpolated middle weighted edges, and then using the brutal force method to do the large weighted edges.

Figure 9 in the beginning of this section shows interesting behaviors. The seperated cases are now connected together into a connected component, with each edge, where each edge gives the closest relationship. The outcome illustrates the following:

First, it shows that it is important to conduct annual simulations rather than only extreme cases. The spanning tree shows that there are at least 100 cases that are extreme cases. The 100 cases differ from each other by greater than 16 percent. Thus, when using the extreme case's results to represent a problem, greater errors might be expected.

Also, there are 1000 cases differing from each other by greater than 10 percent. Thus, simple interpolation will not be accurate enough.

Finally, it is shows that more than half of the cases can be directly interpolated. Thus, tremendous speed-up can be achieved by interpolation which eliminates the need to simulate similar cases.

For instance, when zoomed in, Figure 11 shows that there are many points forming some clusters. We assume the points within a cluster perform very similar results and the minor differences between these points could be neglected. Therefore, the clusters indicate that many cases can be predicted by simply using interpolation, and no fluid computation is needed.

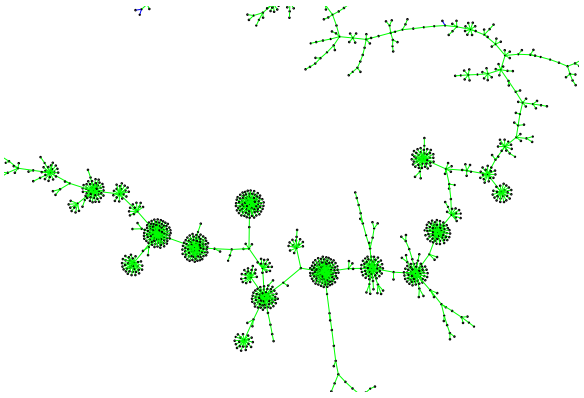

Figure 11: Similar cases forming clusters in the graph

If 8760 cases are calculated separately using the traditional way, the average difference would be 100 percent. Randomly or sequentially choosing a starting point from the simulated points would result a average difference of 35 percent. In the planning strategy mentioned, as there are 8760 cases, there are $8760-1$ paths when the minimum spanning tree is computed. For all the 8759 paths, 5423 paths have a distance below 1 percent, 7032 paths have a distance below 2 percent, 8340 paths have a distance below 3 percent, and only 81 paths have a distance greater than 5 percent. The average difference is much smaller than the traditional method or the random/sequential reusing method. See Figure 12.

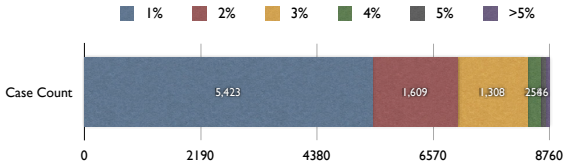

Figure 12: Distance statistics

In order to estimate the overall speed-up, this research uses a fluid solver to perform the tests to estimate the converging speed in relation to case difference.

When simulating without using any previous case results, a typical simulation takes 428 iterations for this simple geometry using an 100×100 grid. On average, 15 iterations will be used for cases that differ by 1%, 53 iterations for 3%, 97 iterations for 5%, 185 iterations for 15%, and 368 iterations for 35%.

Thus, even without any interpolation, the minimal spanning tree based algorithm is 12.55 times faster than simulating without case reuse, or 10.79 times faster than simulating by reusing sequential or random case results.

By incorporating interpolation, the iteration required can be further limited — more than half of the 1% weighted edge can be done without fiuid simulation, and for those cases simulated by fluid solver after interpolation (2% – 5%), much fewer iterations (about 2/3 of them) are also saved after interpolation. Thus, the overall speed-up is estimated to be 20 times faster than the traditional method.

The simulation algorithm will also be able to adjust itself during the run time to predict the actual distance (edge value) between each pair of the cases. When the simulation algorithm detect during actual simulation that two cases are not close or far away as predicted, it will be able to adjust the parameters in its simulation algorithms to correctly predict future cases. Online learning method, may also help when adjusting the parameter settings. When a few edges in the graph are changed, a very efficient linear time algorithm exists to update the minimal spanning tree. But this is beyond the scope of this research.

CONCLUSION

With the algorithms introduced in this researches, it is possible to form an optimized strategy to perform simulation by reusing previous cases instead of simulating similar cases over and over again. The method can correctly choose similar cases as baseline, either using interpolation method to predict the case without any simulation, or using nearest neighboring case as

starting point to do the simulation, thus dramatically reduce the simulation time.
This simulation strategy is self-adjustable. It will automatically adjust its own parameters based on existing cases, and thus make future predictions more efficient.
The algorithm can also be intergrated with GPGPU optimization and FFD algorithm, and each optimization can be standalone. This gives great portability for the three-layered framework.
With the scheduling algorithm added to the three layered framework, this research is able to simulate annual hourly conditions as fast as doing two extreme cases using traditional approaches by multiplying the time factors of the three. Thus, annual approximate simulation can be possible within a short period of time, usually within an hour or several hours.
There are some potential shortcomings of this research, which can be improved in the future.
This strategy planning algorithm requires the input parameters for all hours in annual simulation to be present. In some applications, these data might be computed just in time, whereas a more sophisticated planning strategy might be needed.
Linear regression is sufficient for the simple case study presented, where there is a close-to-linear relationship among output difference, input difference and converging speed. For more complicated cases, such linear relationship may not hold, and a more advanced machine learning method might be needed.
The strategy planning acceleration requires a confined input space, such as simulations of a fully mechanically controlled building enclosure. Only when there are many similar hourly solutions for the annual simulation will this method work. Notably, much work needs to be done to apply this strategy planning algorithm to simulation cases with sub-hourly variations or with large input space, such as those with natural ventilation or sun radiation. Thus, for very complicated scenarios as these, this strategy planning algorithm may fail to achieve expected speed-up.
Interpolating a new outcome on previous results effectively assumes that there is a steady-state solution to the fluid state and that there are no historical effects from previous fluid states or from influencing conditions such as thermal mass. While for hourly simulations this may be acceptable, the application of this approach to sub-hourly simulations is quite limited, and requires more work.

REFERENCES

D Blay, S. M. and Niculae, C. 1992. Confined turbulent mixed convection in the presence of horizontal buoyant wall jet. *Fundamentals of Mixed Convection*, 213:65–72.

Djunaedy, E. 2005. External coupling between building energy simulation and computational fluid dynamics. *PhD thesis, Technische Universiteit Eindhoven*.

Wang, Y. and Malkawi, A. 2013. The foundation of an optimal strategy planning algorithm approach to accelerate annual hourly cfd simulation. *CLIMA2013 11th REHVA World Congress and 8th International Conference on IAQVEC*.

Yue Wang, Ali Malkawi, Y. K. Y. and Feng, N. 2012a. A three layered framework for annual indoor airflow cfd simulation (part i). *The 1st Asia conference of International Building Performance Simulation Association*.

Yue Wang, Ali Malkawi, Y. K. Y. and Feng, N. 2012b. A three layered framework for annual indoor airflow cfd simulation (part ii). *The 1st Asia conference of International Building Performance Simulation Association*.