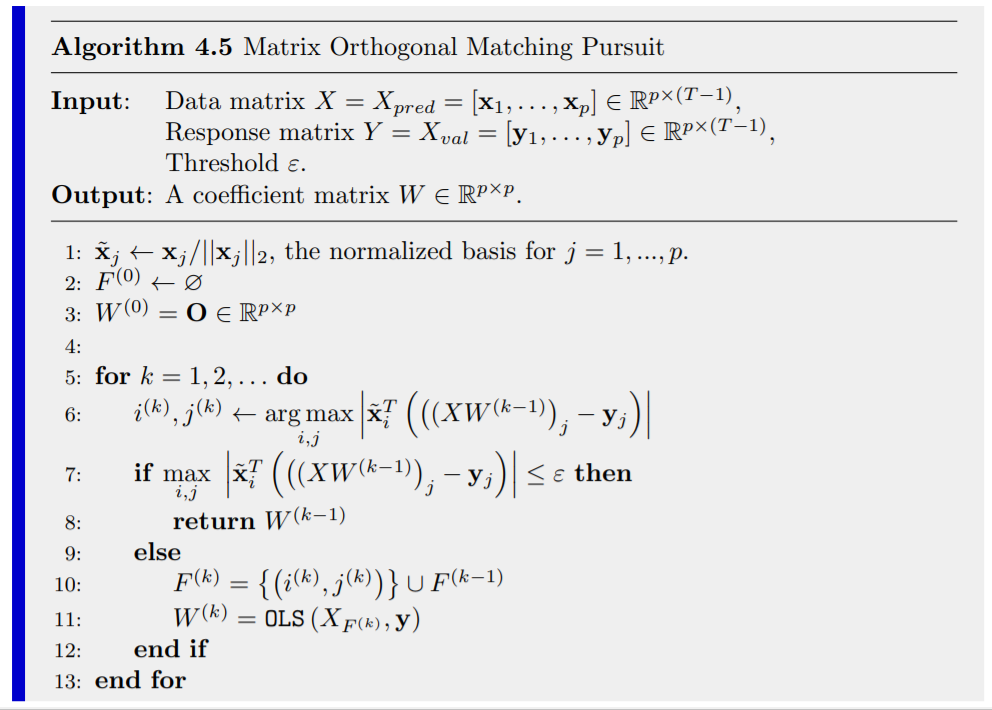
*Prep Meeting 17*

***Thesis***

*Wrote quite a bit, started from the beginning a bit, Exhaustive Search, Random Walk, OMP, quite some text, also formatting and overall structuring.*

***OMP***

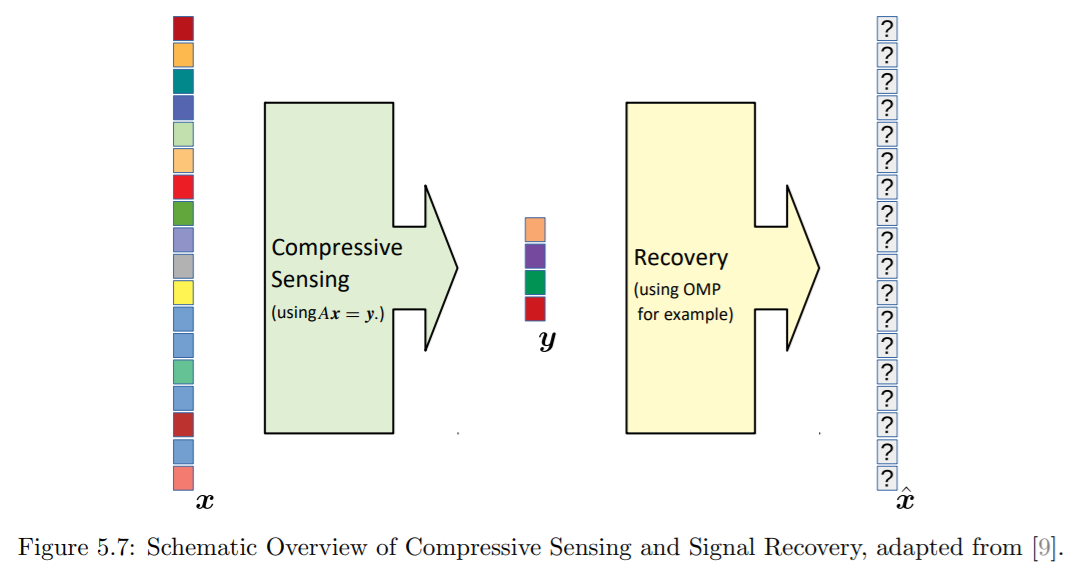
*Made the algorithm work in the Matrix Sense, without the unnecessary copying. Looks nicer for our setting, and is much faster.*



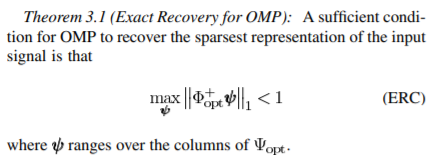
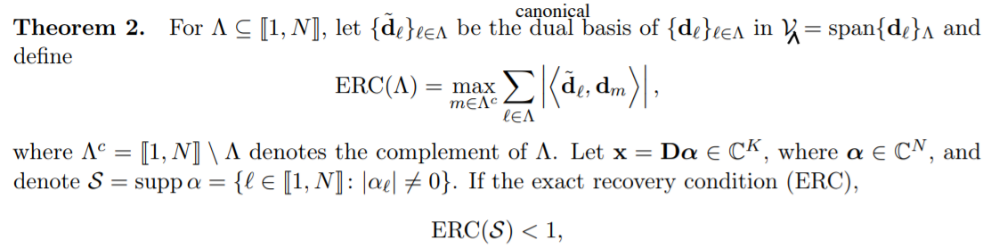
1. Line 6 + 10: In a greedily fashion, OMP picks the “atom” (in our case, coefficient of Wij) that is most correlated with the current residual, *while ensuring G(W)* *remains a DAG* (not in algorithm yet).
2. Line 11: Then, the residuals are re-estimated using ordinary least squares, restricted to all the selected atoms.
3. Line 7: The algorithm stops when we have a full DAG *G(W)* (or in the literature, whenever the largest gain of adding an atom is below a certain threshold \epsilon.

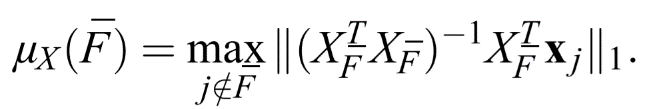
***Statistical Guarantees*** **in the setting of Ax = y**

Original problem: Compressive Sensing and Signal Recovery. Compress using Ax = y. Now, Given A and y, recover x. Figure:



Among different places, I found the following theorem (Tropp, 2004) for when this can be done perfectly (so without any noise):



Now, this quantity is of great interest for the analysis, but quite difficult to grasp. It also shows up in the most resembling paper that Alex sent:

Some explanations: We see the ordinary least squares representation of X\_F, where X\_F corresponds to all the atoms or coefficients that are in the optimal representation of the signal. So, we regress all “non-true” atoms on this optimal X\_F, and sum the absolute values of its coefficients.

Now, for the noiseless case, if this quantity is smaller than 1, then OMP provably recovers the optimal signal always. In the words of Tropp:

*“it guarantees that no spurious atom can masquerade as part of the signal well enough to fool OMP.”*

***Statistical Guarantees for Noisy Case***

Alex found a very interesting paper that is similar to our setting. However, it focuses on linear regression of data X on y, where each sample (X\_i, y\_i) is independent of all others. So, two differences:

1. Theirs is in vector form (y is a vector), ours is in matrix form (our y is *p-dimensional)*
2. Our (X\_i, y\_i) are heavily dependent on each other (between variables at same timestep, but also the same variable at different timesteps).

Nevertheless, their results were as follows:

“If some assumptions hold (normalized data, bounded and independent (X\_i, y\_i)), and this mu < 1, then there exists a threshold epsilon such that when we terminate, we have only found true coefficients.”

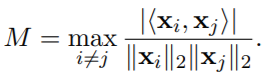
“Furthermore, if all coefficients have an absolute value larger than …, then we have exactly recovered the signal when we terminate.”

It would be very cool to also be able to have such a result for DAG-OMP. However, some hurdles:

* Some assumptions fail (primarily independence, how can we circumvent this?).
* This mu value is unreasonably large in our setting, which implies that all coefficients must have an absolute value larger than e.g. 5, which is of course not possible in our setting.

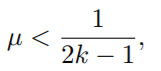
If we can overcome these hurdles, then I think we have some cool results, also from a statistics / mathematics perspective.

***Sparse Noiseless Signaling Guarantees***

Found some extra sources on sparse signal recovery (so the noiseless case), which state somewhat the same results.

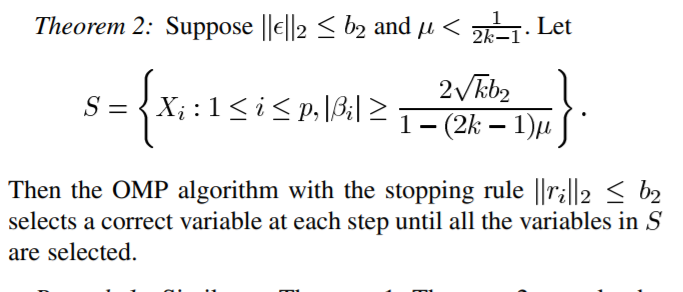
Define the mutual incoherence as

Then, if the signal is *k*-sparse (i.e., only *k* non-zero coefficients in W), then OMP recovers W if

, where mu = M.

Unfortunately, the value for M is in our scenario very high, as there is a large coherence between the time series. Our mu is sometimes close to 1, but almost always larger than 0.5, so this is useless until *k = 1*, which is not a useful scenario. Nevertheless, interesting result. However, it showcases that we use OMP in a very different scenario than it was originally used.

For the noisy case, I found this theorem which builds upon it, but it also used this mutual incoherence which is unsuitable for us unfortunately.

******

Our case seems to be quite different than the conventional ones. Although we have a large mutual coherence, OMP seems to always work well. This seems to be because when there is a large mutual coherence, a true and untrue coefficient are very close in terms of gain. However, the true one will always we slightly larger and when we add that, the gain of the untrue coefficient decreases drastically.

***Mu***

*Investigated the mu, and I hoped to find a way to circumvent the issues, but it seems that is not possible. Even for well-defined matrices W, we can let the mu get arbitrarily close to 1. However, when running OMP, there seems to be no problem. A large mu seems to imply that it is more difficult to distinguish a true coefficient from a false coefficient, but in the end, OMP seems to always pick the right one, so there does not seem to be such an issue here. Nevertheless, for guarantees, it seems to be an issue.*

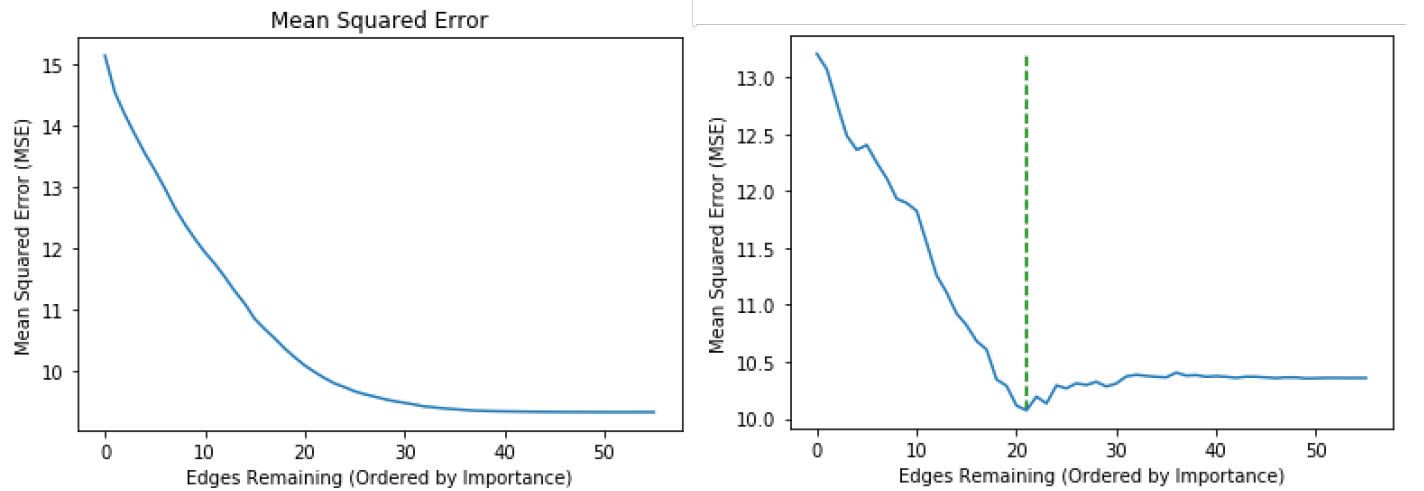
Recall: Mu quantifies how well aspurious atom can masquerade as part of the true signal.

I.o.w. Mu quantifies how well a non-true coefficient can masquerade as a true coefficient.

See notebook for demo.

***LOOCV OMP***

We were working on using bootstrapping approaches and crossvalidation approaches to find the datagenerating matrix W. If we do not use any of these approaches that DAG-OMP will always yield a dense DAG G(W). However, we can expect that a large portion of these coefficients are actually untrue, but are simply overfitting on correlated noise. We can also see this in the plot of the MSE.



Left: No cross-validation, we overfit on the noise.

We see after adding ~25 edges, the MSE does not decrease that much anymore. It seems these edges are not that important, but since we only stop at a dense DAG, we still continue. This means that we achieve a **low** MSE, but a **high** false negative rate (we estimate coefficients whereas it is just noise), resulting in **low** accuracy.

Right: Cross-validation. We clearly see a minimum at around 20 edges (which was true, in fact!). We achieve a **slightly worse MSE**, but we achieve an **almost perfect** accuracy (99%).

Interesting behaviour: \ **\_,** three groups.

\/

**Cross-Validation Approaches**

Bootstrapping and Cross-Validation must be done with **caution**, as we have **highly dependent** data in the form of time series. What works:

Issue with this dependent data: Suppose we want to do LOOCV. We leave one time step *t* of *p* variables out, and train on the remaining data. This will not work, as there is an unreasonably large gap between X\_{t-1} and X\_{t+1}, which results in an incorrect fit. The effect of this becomes smaller as *T* increases, but it is mathematically not just.

Testing one sample, on the other hand, is okay. We can simply test our matrix *W\_LOOCV(t)* on X\_t and Y\_t = X\_{t – 1}W.

So, the issue is the training set. We must somehow not train on the gap between X\_{t – 1} and X\_{t + 1}, but only on the two blocks X\_[1:t-1] and X\_[t+1:T]. For selecting, we can simply exclude the *t*th component when calculating the correlation. However, for the OLS then, how can we exclude X\_t from the calculation?

*So, it seems difficult to remove the dependency of X\_t in the data with LOOCV.*

What does work? When we split the data X into two blocks, train and test, then it is quite simple. We train on X\_[1:k] and validate on X\_[k+1:T].

**Extra Stuff**

**Kernel OMP:** Paper from Li. et al. From Australia showed that all OMP calculations can be done by inner procucts, hence allowing for Kernel-Trickes. They show Kernel-OMP on some data, and apparently it is very fast. Perhaps an interesting direction to try in the future?

**Random Walk Puzzle:** Quite unrelated, but I briefly discussed using a random walk on the set of permutation matrices in the thesis, and I wanted to show that the initial permutation is important. E.g., if we start at P = [[p]] = [1, 2, …, p], it may take a long time to reach its reverse -P := [p, p – 1, …, 1].

Now, the question: Assume we can do random transpositions of two integers, what is the expected number fo steps to go from P to -P? It seems to be approximately p!.

We have for p = 2, 3, … that the answer is 1, 5, 27, 128, …, however I have not found a closed form solution and it irks me to not have an answer.

**Misc**

Other things in the last two weeks were put in the two PDF documents, which can be found on the GitHub.