Cassowary Development Report

Athan Clark

September 22, 2015

Abstract

The development of a pure implementation of the Cassowary constraint solver has been a difficult one indeed; I've found there to be crucial details overlooked in the specification of the algorithm - mainly regarding the concept of *weighted* coefficients.

I've further refined the concept of the algorithm down to a cleaner specification, with a few issues still remaining before a fast and efficient implementation can be achieved.

Main Issues

Weighted Coefficients as Lists

Initally, I was under the impression that weighted coefficients could be tangibly regarded as "objects" - things that we can compare to each other, combine, etc. This is actually not the case for Cassowary; rather, we should imagine each weighted coefficient as a stream of values. Now when we need to find the next pivot, instead of having a one-shot computation to find the output we're looking for, we need to fold over all the streams at once. This took me a while to understand, and all my code is based on my previous idea.

The Java implementation of Cassowary actually goes against this paradigm, and naively tries the monolithic appoach I initially imagined - they crudely multiply each weight by some large factor like 1000 I think it was. This is incorrect according to the specification, and shouldn't be called a *weighted* Cassowary implementation.

The real perspective we should have is that the weights add a dimension to the entire solving process, and we can't factor the entirity of that dimension in each pivot step, but rather we would need to pivot multiple times along the length of that dimension.

Luckily this won't be too difficult to engineer around - most of my code is very generic and reusable, the main hurdle I see is retaining the generic nature of the existing code while embedding this particular behavior. I still need to work this out.

Oriented Matroid

The oriented matroid embedding is still very elusive - I did a lot of work making generic set-oriented optimization utilities for Haskell, and they are very straightforward for abstract code. However, I still don't completely understand how Richard Bland proved that his pivoting algorithm (the one that's used in Cassowary) satisfies an oriented matroid.

Without this information, it would be hard to claim that my implementation would be perfectly "correct". I can infer where I *ought* to be coding, and how this all ought to work, but I can't formally verify that my implementation is an oriented matroid without **excessive** work.

I think I can get by without needing this, however in the long run it would be wise to prove these formalisms for the library's integrity and stability. For now thought, this is secondary to getting the system running.

Conclusion

After I have the top point complete, then we will have (*fast*) primal and dual simplex optimization algorithms for *weighted* constraint sets, which is 90% of what the Cassowary solver actually is under-the-hood.

After this, the rest of the engine's development will involve edge-case scenarios - short circuiting unnecessary computations, small optimization tricks for edit constraints, etc. Even this isn't that important - we could start working on the Haskell → PureScript port immediately if the Haskell version proves to be performant enough.

From my most recent benchmarks, we are working in the microsecond (μs) range for single pivots on constraint sets *without* weights, up to 1000 variables. I haven't viewed the asymptotics of multiple pivots, nor how weights affect the scheme, but I have high hopes.