Using Constraint Programming to Enumerate Permutations avoiding Mesh Patterns

Ruth Hoffmann University of St Andrews

This talk is based on joint work with Özgür Akgün, Chris Jefferson

Constraint programming is a proven technology for solving complex combinatorial decision, optimisation or enumeration problems. Constraints are a natural, powerful means of representing and reasoning about complex problems that impact all of our lives. Constraint programming offers a means by which solutions to such problems can be found automatically, and proceeds in two phases. First, the problem is modelled as a set of decision variables, and a set of constraints on those variables that a solution must satisfy. A decision variable represents a choice that must be made in order to solve the problem. The domain of potential values associated with each decision variable corresponds to the options for that choice.

Enumerating permutations avoiding mesh (or other) patterns lends itself perfectly to constraint programming. We have created a model which represents the definition of a mesh pattern in `Essence` . We then use `Conjure` to solve a range of permutation pattern problems. We can find if a pattern is present at all, or count the number of occurrences of the pattern.

The main benefit of modelling permutation patterns as a set of constraints is that constraint models are composable. This means it is easy to find the permutations which satisfy a set of patterns, or find permutations which contain one mesh pattern, but do not contain a second mesh pattern.

We will share the model, and give insights into how it works, as well as present results to show how competitive constraint programming can be with fast algorithms for NP-complete problems.

Modelling Mesh Patterns

One of the most difficult part of modelling mesh patterns is edge conditions – in the mesh the cells around the edge represent "all values up to the end of the permutation", which need to be handled differently.

To avoid having to special case the edges of the mesh, we extend our permutation from searching for a permutation p on $\{1,\dots,n\}$ to a permutation on $\{0,\dots,n+1\}$, where $0^p = 0$ and $(n+1)^p = n+1$. Similarly when searching for the pattern in a permutation, we add an extra fixed value to the start and end of the pattern, which map to 0 and $n+1$ respectively.