

# OPTIMIZING KNITTING PROGRAMS

Laxman Dhulipala and Vidya Narayanan

15745 Compilers  
Carnegie Mellon University

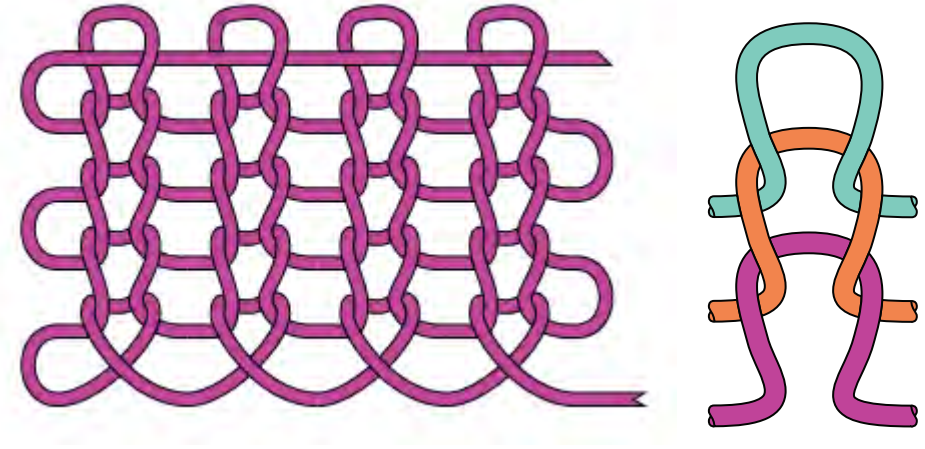



## Overview

Industrial knitting machines are programmable systems that can robustly fabricate soft objects. Most manufacturers offer custom solutions to design for and program these machines; these interfaces are traditionally built for the expert knit programmer and used to mass manufacture apparel. However, a modern V-bed knitting machine can be abstractly viewed as a hardware system that supports a small instruction set of standard operations and machine specific extensions. Having such a low-level abstraction layer allows one to build design and optimization tools for these fabrication processes without having to worry about specific hardware instances. Further, compiler techniques can be directly applied to low level knit IR to incorporate machine specific optimizations, particularly in scheduling knitting instructions to reduce fabrication times. In this work we designed a compiler optimization to improve the scheduling of a DAG representation of knitting programs and thereby reduce fabrication times. While we cannot prove that our algorithm is optimal (the underlying optimization problem is NP-hard), our optimization pass reduces the number of blocks for programs derived from a database of over 1200 lace patterns by 5.8x on average showing that it is effective in practice.

## Machine Knitting

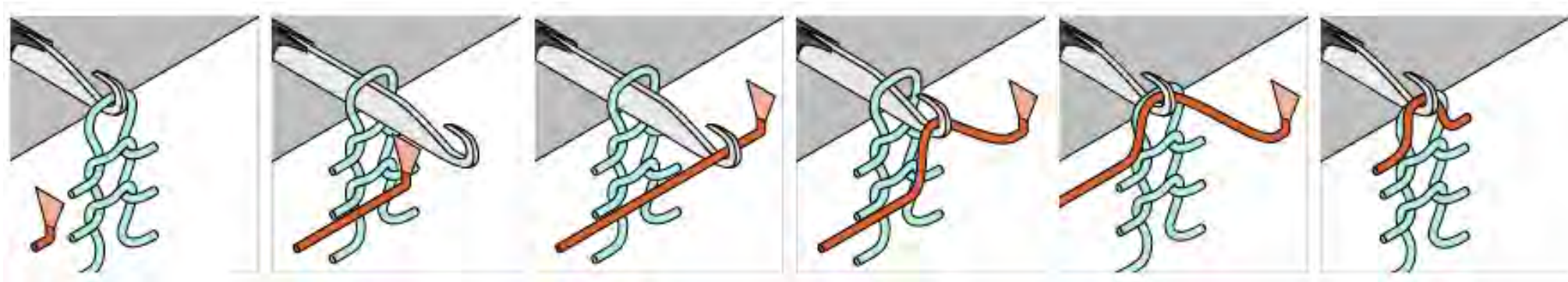
Knitting involves inter-meshing loops formed from a yarn to generate a stable fabric – each loop in the swatch is stabilized by the loop that is pulled through it:



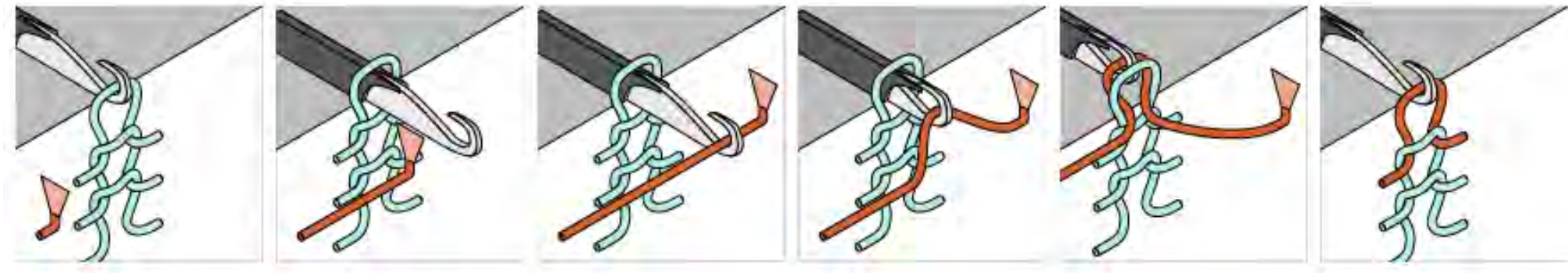
A knitting machine makes and holds a loop on a needle ; viewed from the top as . The machine is made of a two beds of needles. The needle beds can be offset with respect to each other and loops can be transferred across needles on opposite beds. Needles are actuated by the movement of a 'carriage' on rails over the length of the bed which also dictates the direction in which the yarn is laid out.

## Knitting Machine ISA

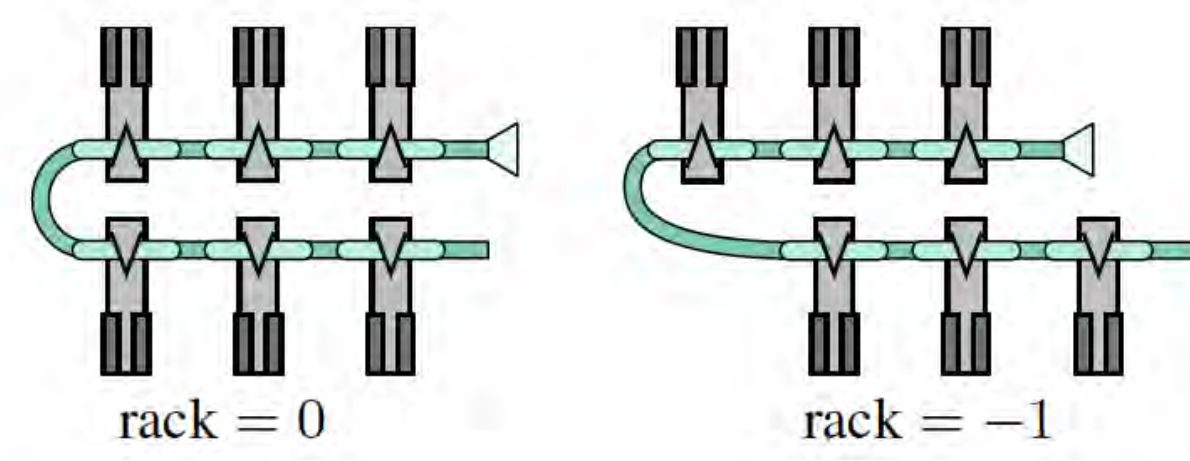
**Tuck operation** [tuck dir NB Y] : forms a loop in a specified direction(+ or -) at a needle-bed(NB) location with a specified yarn carrier Y



**Knit operation**[knit dir NB Y] : forms a loop through existing loops in a specified direction and needle-bed location



**Rack operation** [ rack x ]: translates the back(upper) bed by a specified amount(x) with respect to the front(lower) bed .



**Transfer operation** [xfer fromNB toNB]: moves all loops at a needle-bed location to the aligned needle on the opposing bed.



A knitting program is represented as a sequential list of these operations along with operations **In Y** and **Out Y** to bring in an active yarn and release it, respectively . (Figures from A compiler for 3D Machine Knitting, McCann et al SIGGRAPH 2016)

## Cost metric for knitting programs

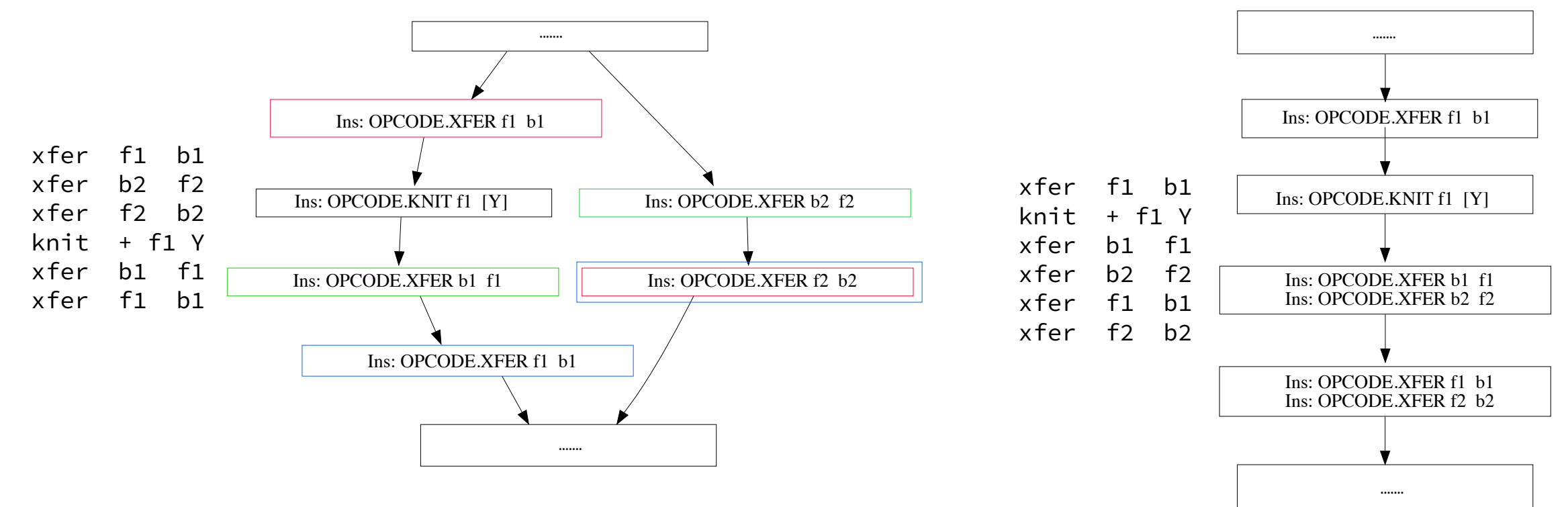
Knitting programs are decomposable into a set of *passes*. Each pass is a nearly constant time operation, so the objective is to minimize the total number of passes that are executed. A common hardware requirement is for **knit** & **tuck** operations that add loops to needles and **transfer** operations that move loops across needles to be executed in different passes. The **rack** instruction is executed *once* per pass and all operations in a pass are executed at the same racking value.

## DAG representation of Knitting Programs

We developed a DAG representation of a knitting program that captures choices in how a knitting program can be executed. One block dominates another in the DAG if there is a true needle or yarn dependency between the two blocks, so the ancestor must be executed first. Otherwise, the blocks are logically in parallel, which means that executing either one first produces a correct output. A *valid execution* is any topological ordering of the DAG.

## Optimizing a knitting DAG

The following example shows two reorderings of the same knitting code that require different number of passes to execute them. The DAG representation captures dependencies by edges and also highlights opportunities to reorder instructions – sibling nodes in the graph can execute in any order and merging instructions in such blocks may improve efficiency.



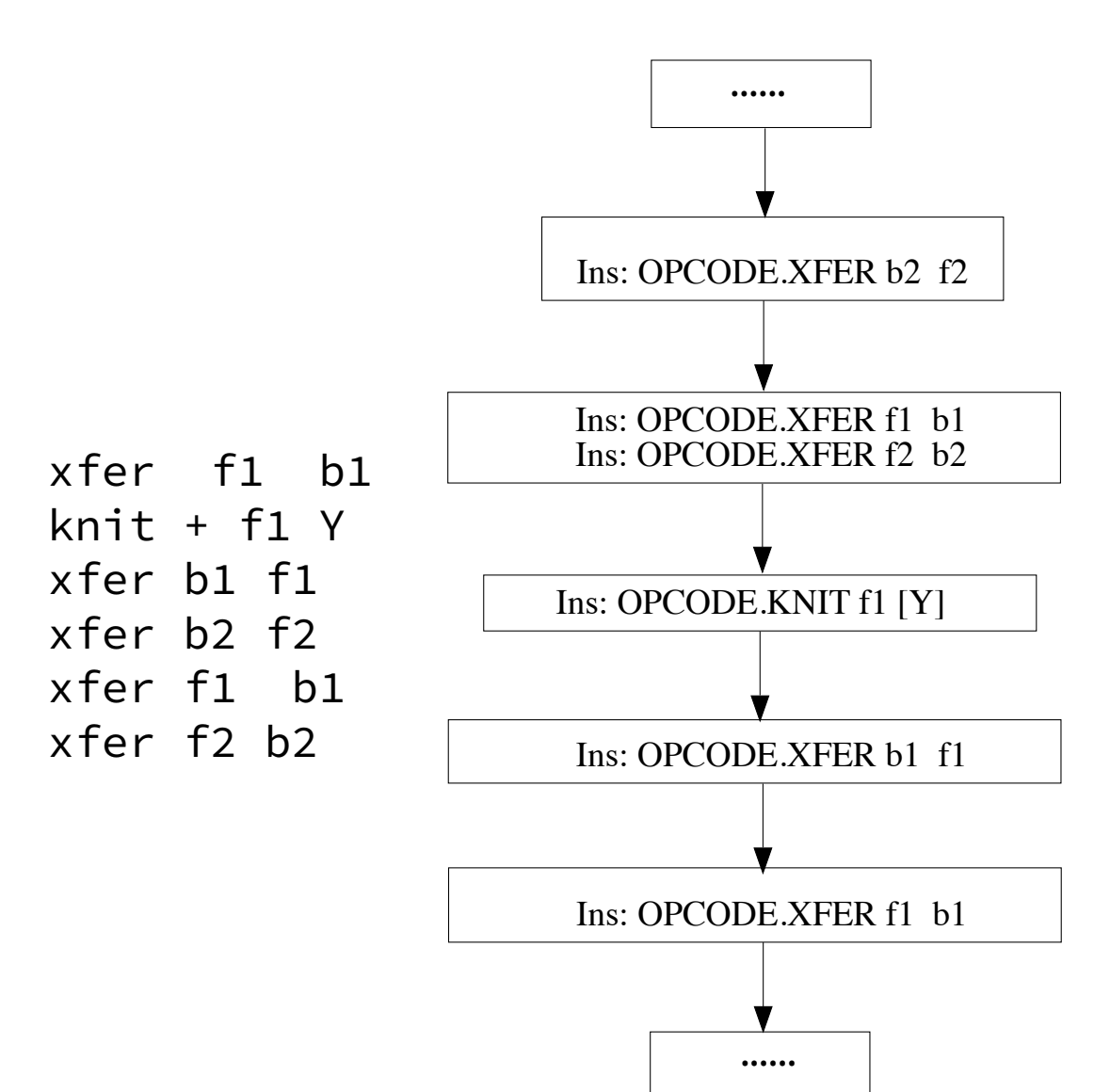
**The knitting reordering problem.** Given a DAG representing a knitout program, where blocks in sequence represent a needle or yarn dependency and blocks in parallel represent that the blocks can be executed in any order, find an equivalent DAG respecting the dependencies that minimizes the number of blocks(passes) used.

We showed that this problem is NP-hard by reducing from 3-SAT. The reduction had to make use of the fact that there are at least two *types* of passes which restrict blocks that are in parallel from merging (only blocks of the same type can be merged).

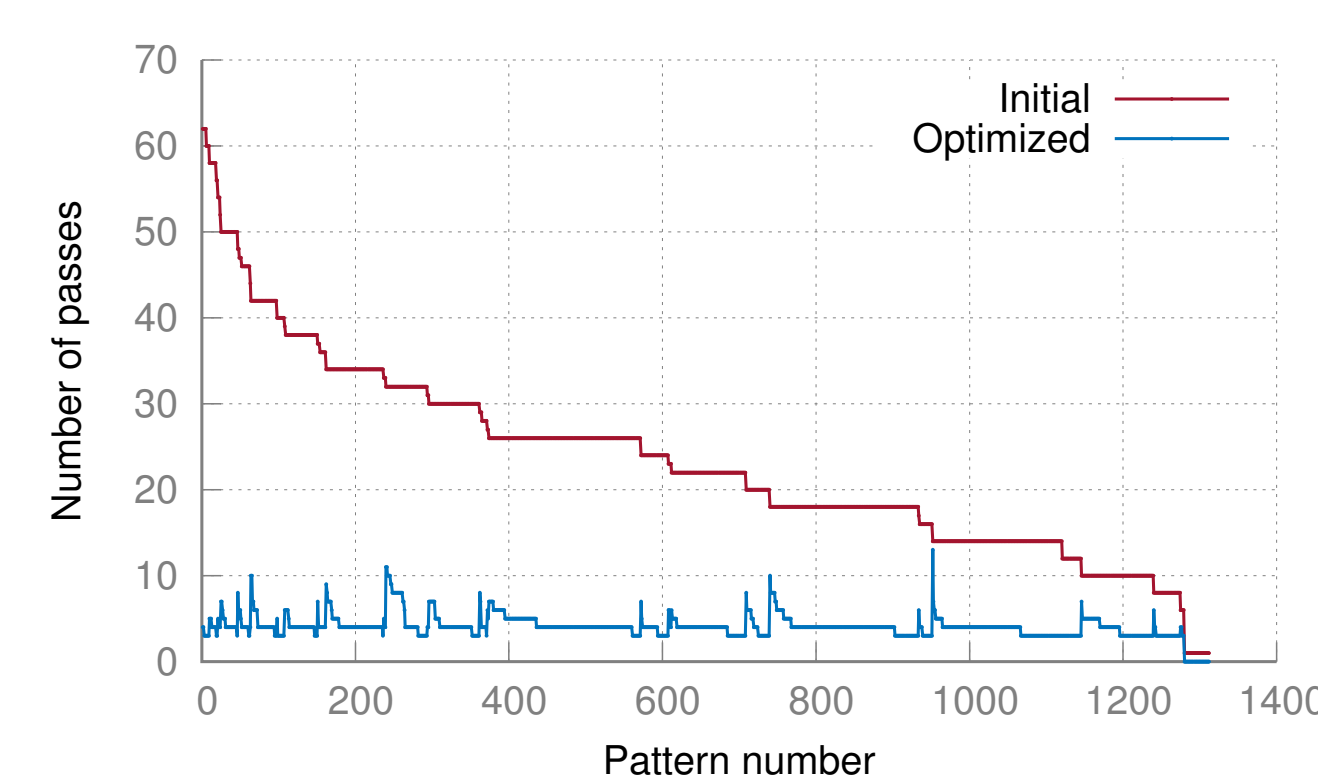
## Heuristic solution for code movement

Our heuristic algorithm operates in two passes. The first pass attempts to move each instruction to the earliest block it can be merged with, repeating until no improvements are found. The second pass attempts to move each instruction downward merging it with latest block it can be merged with, repeating until no improvements are found.

Our heuristic may indeed miss finding an optimal solution because it greedily merges instructions when it finds the opportunity to do so. A different order of merging blocks that merged the red blocks(in the graph above) first, would have generated the solution shown in the inset – which is sub-optimal. However, in general our heuristic does manage to reduce the number of passes to significant extent (see graph).



## Optimizing the Lace Pattern Database



(Left) We ran our optimizer on transfer planning knitting instructions generated for over 1200 patterns from the 'Essential Stitch Collection'. On average, the optimization provides a 5.8x reduction in the number of passes. (Right) An example of a larger lace pattern