DNA Weaver: optimal DNA assembly strategies via supply networks and shortest-path algorithms

Valentin Zulkower
valentin.zulkower@ed.ac.uk
Edinburgh Genome Foundry
School of Biological sciences
The University of Edinburgh, UK

The Edinburgh Genome Foundry Team
School of Biological Sciences
The University of Edinburgh, UK

Susan Rosser
Edinburgh Genome Foundry
SynthSys Centre
School of Biological Sciences
The University of Edinburgh, UK

1 MOTIVATION

Synthetic Biology applications often require the assembly of large DNA sequences from smaller fragments. This can be a long and expensive process, involving the careful design of many fragments sequences and multiple assembly steps. While software can help alleviate these challenges, existing solutions focus on specific scenarios, such as plasmid assembly from standardized genetic parts [1], and chromosome synthesis from commercially ordered DNA blocks [2] or oligonucleotides [6]. However, assembly projects may use different combinations of cloning methods (e.g. Gibson Assembly, Golden Gate assembly, recombination in yeast) and DNA sources (parts repositories, genomic DNA, commercial providers), depending on the desired DNA sequence and the researcher's preferences.

We present DNA Weaver, an open-source Python framework[1] and web application[2] to compute cost- and time-optimal assembly strategies from user-specified DNA sources and cloning methods. DNA Weaver combines supply-network models with shortest-path algorithms to return either perfect solutions or quick approximations to a variety of problems. This makes it suitable to plan routine cloning in a research laboratory, to automatically evaluate the costs and complexity of projects submitted online to a biofoundry, or to automatically optimize the design of novel sequences towards better manufacturability.

2 METHODS

Problem definition via supply networks

Users define an assembly problem either via Python scripts or the web interface, by providing the desired final sequence and designing a supply network representing all available cloning options (Figure 1A). Each node of the supply network represents a DNA source, e.g. a commercial provider, a parts repository, a PCR station which extracts DNA from existing constructs or genomes, or an assembly station which assembles supplied fragments into a larger one. Each source can be extensively parametrized to reflect its capabilities. For instance, DNA vendors can have custom sequence acceptance rules (see [5] for examples), pricing policies, and lead times. PCR stations can have a range of acceptable oligo primer lengths and annealing temperatures, as well a BLAST database of available constructs and genomes. Assembly stations can implement different cloning methods defined by parameters such as fragments overhangs, restriction sites used, acceptable size ranges of the fragments and final assembly, forbidden sequence patterns, etc.

Each edge of the supply network indicates a client-supplier relationship between two DNA sources. If a PCR station needs a primer oligonucleotide, or an assembly station needs a particular DNA fragment, they request the sequence from all of their direct suppliers. Each supplier indicates whether it can provide the sequence, at which price, and with what lead time. The client station then retains the best offer as the fragment price. This chain of competitive bidding between members of the network ensures that each DNA fragment of each assembly is obtained using the cheapest (and most adapted) vendors and assembly methods.

The variety of possible supply network layouts enables DNA Weaver to solve various cloning problems, from multi-step assembly (Figure 1A) to site-directed mutagenesis or the auto-completion of parts library assemblies with commercially ordered sequences (interactive examples of such scenarios are provided in the web application). In addition, the use of supply networks makes it easy to plug in new classes of DNA sources with different behaviors (which users can define using the Python language), for instance sources connected to remote sequence databases or vendor APIs.

Graph-based sequence decomposition

To provide a sequence at the best price, an assembly station must find the cheapest possible set of fragments which can be assembled into the desired sequence. This often amounts to identifying which regions of the sequence can be inexpensively obtained from DNA repositories or from the cheapest vendors. The sequence decomposition algorithm starts by building a costs graph (Figure 1B) where each edge represents a possible sequence fragment, and an edge's weight is

[1] https://github.com/Edinburgh-Genome-Foundry/DnaWeaver
[2] https://dnaweaver.genomefoundry.org

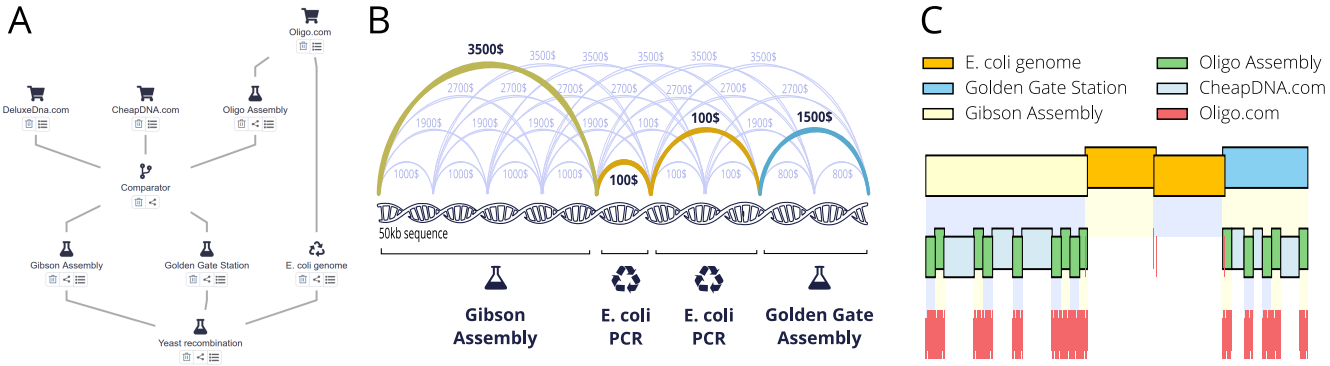

Figure 1: DNA Weaver's web interface, sequence decomposition algorithm, and output. (A) Web interface screenshot of a user-defined supply network modeling a multi-step assembly protocol. The final sequence is assembled via recombination in yeast (bottom station) from large fragments obtained either via Gibson assembly, Golden Gate assembly, or via PCR of the *Escherischia coli* (*E. coli*) chromosome. The Golden Gate and Gibson stations assemble fragments originating either from two commercial providers (cart icons), or from oligonucleotide assembly. (B) Simplified representation of the costs graph used by the yeast recombination station in panel A to decompose a 50kb sequence. The highlighted shortest path indicates the cheapest solution, where lateral fragments are obtained from assembly stations, and the central region's homology to *E. coli* is exploited to obtain cheap DNA fragments via genomic PCR. (C) Assembly plan for the 50kb sequence in panel B, from oligo assembly (bottom level) to the final assembly in yeast (top level), returned in under 10 seconds by the web application.

the fragment's cost (accounting for the addition of assembly overhangs). The shortest graph path from the first nucleotide to the last can be computed using the Dijkstra algorithm [3] and gives the cost-optimal sequence decomposition.

The computational complexity to cost all edges is $O(L^2)$ (where L is the sequence length). DNA Weaver also implements approximate resolution approaches such as the A* path-finding algorithm [3] or nucleotide-skipping, which can reduce the number of edges to compute by orders of magnitude, allowing quasi-real-time sequence editing with manufacturability feedback.

Further graph operations allow to model the limitations of a cloning method, e.g. by limiting the number of parts in an assembly, constraining fragments sizes, forbidding incompatible overhangs to be used in a same cloning step, or preventing high-GC regions to be used as overhangs. It is also possible, by filtering graph edges based on supplier lead time, to obtain the cheapest cloning strategy under a time constraint, or the fastest assembly plan within a given budget.

Output and use for sequence optimization

DNA Weaver generates comprehensive assembly planning reports featuring plots of the final assembly plan (Figure 1C), PDF documents summarizing all sequence ordering assembly steps, and spreadsheets listing all sequences to order.

For design optimization purposes, DNA Weaver can return an analysis of the cost graph pinpointing the sequence locations susceptible to impact assembly cost. This information can then be used to iteratively re-design the desired sequence towards lower costs and manufacturing complexity, possibly taking into account other application-specific objectives and constraints, via existing DNA optimization software such as DNA Chisel [4].

Acknowledgments

The Edinburgh Genome Foundry is supported by the BBSRC (BB/M025659/1, BB/M025640/1, and BB/M00029X/1 to YC) and the BBSRC/MRC/EPSRC funded UK Centre for Mammalian Synthetic Biology (BB/M0101804/1) as part of the RCUK's Synthetic Biology for Growth programme.

REFERENCES

[1] Appleton, E., Tao, J. H., Haddock, T., and Densmore, D. Interactive assembly algorithms for molecular cloning. *Nat Methods 11*, 6 (2014), 657–+.

[2] Christen, M., Del Medico, L., Christen, H., and Christen, B. Genome Partitioner: A web tool for multi-level partitioning of large-scale DNA constructs for synthetic biology applications. *PLOS ONE 12*, 5 (2017), 1–19.

[3] Delling, D., Sanders, P., Schultes, D., and Wagner, D. Engineering route planning algorithms. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* (2009).

[4] Edinburgh Genome Foundry. DNA Chisel, a generic sequence optimizer. https://github.com/Edinburgh-Genome-Foundry/DnaChisel.

[5] Oberortner, E., Cheng, J. F., Hillson, N. J., and Deutsch, S. Streamlining the Design-to-Build Transition with Build-Optimization Software Tools. *ACS Synthetic Biology* (2017).

[6] Richardson, S. M., Mitchell, L. A., Stracquadanio, G., Yang, K., Dymond, J. S., DiCarlo, J. E., Lee, D., Huang, C. L. V., Chandrasegaran, S., Cai, Y., Boeke, J. D., and Bader, J. S. Design of a synthetic yeast genome. *Science* (2017).