Interactive assembly algorithms for molecular cloning

Evan Appleton[1,2], Jenhan Tao[3], Traci Haddock[2] & Douglas Densmore[1,2,4]

Molecular biologists routinely clone genetic constructs from DNA segments and formulate plans to assemble them. However, manual assembly planning is complex, error prone and not scalable. We address this problem with an algorithm-driven DNA assembly planning software tool suite called Raven (http://www.ravencad.org/) that produces optimized assembly plans and allows users to apply experimental outcomes to redesign assembly plans interactively. We used Raven to calculate assembly plans for thousands of variants of five types of genetic constructs, as well as hundreds of constructs of variable size and complexity from the literature. Finally, we experimentally validated a subset of these assembly plans by reconstructing four recombinase-based 'genetic counter' constructs and two 'repressilator' constructs. We demonstrate that Raven's solutions are significantly better than unoptimized solutions at small and large scales and that Raven's assembly instructions are experimentally valid.

Genetic engineering technology has increased in scale and complexity since it was invented 40 years ago. In the past decade, a number of DNA assembly methods have emerged[1–6] that include restriction-ligation–based and homologous recombination–based cloning systems, and many follow the design model of assembling genetic 'constructs' from genetic 'parts'[7]. While the precise definitions of these terms have been debated[8], the consensus is that parts are DNA segments and constructs are ordered sets of these parts. Given this design model, two fundamental questions arise: First, how should we identify and select parts to create the desired functional genetic constructs? Second, once these constructs have been selected, how do we physically assemble them? This work formalizes this second question and provides algorithms that address experimental realities to improve the speed, modularity and experimental efficiency of this process for state-of-the-art DNA cloning techniques.

Cloning-based assembly approaches can be broadly classified into binary assembly techniques, where two DNA parts are assembled in one cloning step, and multi-way (one-pot) assembly techniques, where two or more parts are assembled in one step. Generally speaking, multi-way assembly methods are faster because they can minimize cloning steps and can be exploited to leave no assembly artifacts in a completed construct, while binary assembly methods typically require more cloning steps, but in some cases utilize specific cloning sites to allow a simpler standardization of part composition and modularity.

Previous work[9,10] detailed hierarchical assembly algorithms for binary assembly[11] but lacked formulation to address more modern multi-way assembly techniques. There are few automated tools to exploit the high degrees of modularity and reuse for multi-way techniques, and no tools for producing complete assembly plans. Some approaches[12] detail the process of automatically selecting oligonucleotides and analyze trade-offs between cloning and gene synthesis for multi-way assembly; however, they do not optimize cloning steps and stages.

This article describes a method for performing optimizations on intermediate cloning step selection and part junction selection for any number of target constructs while considering a library of existing parts for reuse. We show that for sets of thousands of variants of multiple types of contemporary genetic constructs and a large set of constructs from the literature[13–20], our program outperforms unoptimized solutions ($P(z) < 0.001$), and we then experimentally verify a small subset of these optimized solutions by reconstructing 'genetic counter' and 'repressilator' constructs. This work also details, to our knowledge, the first automated cloning workflow in which experimental outcomes may be directly fed back into the software to recalculate an alternative assembly plan.

The algorithms presented are housed in an online web application called Raven that produces full assembly plans in human- and computer-readable instructions and graphical Synthetic Biology Open Language (SBOL)-compatible images for each of the supported assembly methods[1–5,11].

RESULTS

Algorithm overview

In this work, we break the problem of assembling a set of DNA constructs with a selected cloning technique into three main sub-problems. First, we determine an optimized hierarchical cloning plan for assembling a set of constructs. For this part, Raven uses dynamic programming to reduce the computational time it takes to solve the large problem of selecting intermediate cloning steps for a set of target constructs into smaller sub-problems. Because the heuristic scores for assembling a specific intermediate are

[1]Graduate Program in Bioinformatics, Boston University, Boston, Massachusetts, USA. [2]Center of Synthetic Biology, Boston University, Boston, Massachusetts, USA. [3]Bioinformatics and Systems Biology Graduate Program, University of California, San Diego, La Jolla, California, USA. [4]Department of Electrical and Computer Engineering, Boston University, Boston, Massachusetts, USA. Correspondence should be addressed to D.D. (dougd@bu.edu).

RECEIVED 27 NOVEMBER 2013; ACCEPTED 7 APRIL 2014; PUBLISHED ONLINE 28 APRIL 2014; DOI:10.1038/NMETH.2939

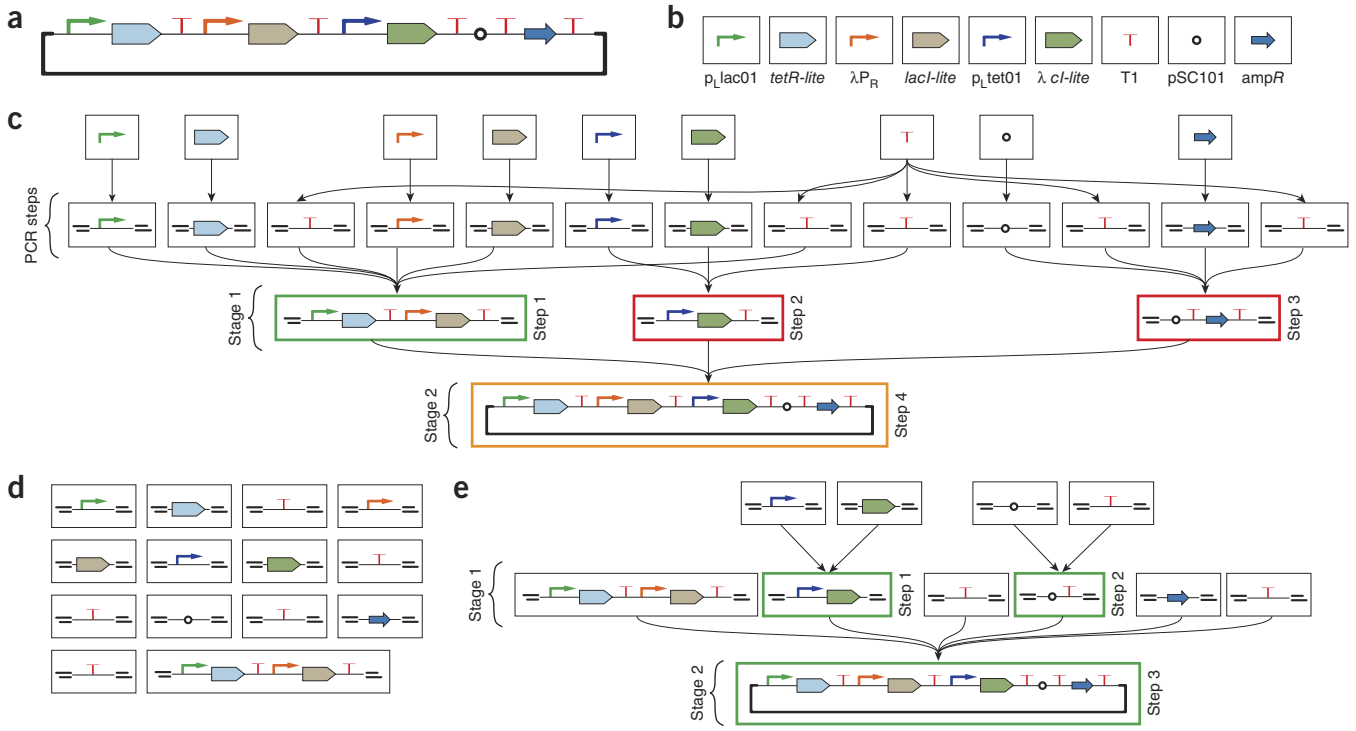

Figure 1 | Example assembly of the repressilator[21] with Gibson assembly. The graphic symbols are composed using Pigeon[22] (http://www.pigeoncad.org/) from SBOL[23] (http://www.sbolstandard.org/) visual images to denote part types. (**a**) The repressilator. (**b**) Starting library consists only of template DNA. (**c**) A plan for assembling the repressilator given **b** requires 13 PCRs, 4 steps and 2 stages. Two steps fail (steps 2 and 3; red boxes), one step succeeds (step 1; green box) and the dependent step in the second stage cannot be attempted (orange box). (**d**) The updated library contains basic parts and intermediate parts with specific overhangs from **c**. (**e**) An optimized plan, in which all steps succeed (green boxes), is generated with no PCRs, three steps and two stages.

assumed to remain constant regardless of previous or future cloning steps, once an optimized heuristic solution for an intermediate is found, the solution is reused if the candidate intermediate step is encountered again. For constructs that share parts internally or share parts with other target constructs, sharing of assembly intermediates can reduce the total step count considerably, and, if many steps can be done in parallel, the number of cloning stages can also be minimized. The cloning step solution comprising all the stages and steps necessary to build the target constructs under consideration constitutes a hierarchical 'assembly graph'.

Second, we determine an optimized set of part junctions (hereafter referred to as 'overhangs') required to perform the selected cloning steps. Overhang assignment, which aims to minimize overhang generation cost (hereafter 'PCR steps'), is determined in three steps. First, the requirements for overhang uniqueness based on the selected assembly method are determined (for example, all parts must have unique pairs in each individual cloning step). Second, for modular overhang assignment, overhang pair sharing is maximized for all cloning steps where sharing can eliminate extra steps and the total number of unique overhang sequences is minimized. The second step is skipped for some assembly methods because it is assumed that all overhangs are either the same or all are unique. Third, after all overhang pairs have been determined, the existing parts library is used to map abstract overhangs to DNA sequences, maximizing library reuse with a constrained Cartesian product (**Supplementary Software** and **Supplementary Figs. 1–3**).

Third, based on the cloning steps and their overhangs, oligonucleotides for PCR are automatically designed (**Fig. 1a–c**). For all PCR steps, primers are designed on the basis of part, overhang sequence, direction (forward or reverse strand) and assembly method. The primer designs are optimized for length and melting temperature, but other complex optimizations are not considered. For more sophisticated primer designs, we provide outputs compatible with existing state-of-the-art primer design software[12]. The summary of all cloning steps, PCR steps and oligonucleotide designs constitutes a complete 'assembly plan'.

Following Raven's assembly instructions, a user might encounter some assembly steps that fail. The user can then mark each step in the plan as successful, failed or not attempted (as a result of step failures in an earlier stage) (**Fig. 1c**) to recalculate an alternative assembly plan. The parts from the successful steps are added to the library and failed steps are forbidden from appearing in a new plan (**Fig. 1d,e**). The interactive refinement of an assembly plan is meant to be independent of specific protocols and reaction conditions and can complement troubleshooting specific reactions in a preliminary plan. This process continues iteratively until all target parts are assembled. As this algorithm relies on heuristics in many locations, we cannot make any claims that it is optimal. However, we can prove that the solutions are correct in linear time as a function of the number of intermediates.

In silico assembly of thousands of constructs

To determine the quality of Raven's assembly plans, we compared our solutions against unoptimized solutions for each data set by randomly sampling the assembly plan space for each set of constructs under consideration (**Supplementary Note**). For both unoptimized solutions and Raven solutions, we assumed no

Table 1 | Raven-optimized and average unoptimized assembly scores for constructing plasmids from the literature

Construct source	No.	Unoptimized solutions									Raven solution								
		RFC10			RFC94			Gibson			RFC10			RFC94			Gibson		
Bonnet *et al.*[20]	6	6.09	45.7	21	3.83	47.7	58	2.67	12.3	34	5	43	21	3	39	54	2	8	34
		0.80	1.24	0	0.56	2.07	0	0.59	2.05	0	0.08	0.02	0.5	0.07	<0.01	0	0.13	0.02	0.5
Bonnet *et al.*[19]	13	6.81	80.9	23	4.41	116	127	3.27	32.9	55	5	73	23	3	75	104	2	24	55
		0.64	2.61	0	0.52	4.50	0	0.50	2.73	0	<0.01	<0.01	0.5	<0.01	<0.01	0	<0.01	<0.01	0.5
Friedland *et al.*[14]	5	8.97	89.7	27	4.99	101	125	4.99	33.3	50	6	74	27	3	49	62	2	19	50
		0.87	3.49	0	0.59	4.05	0	0.59	2.91	0	<0.01	<0.01	0.5	<0.01	<0.01	0	<0.01	<0.01	0.5
Lou *et al.*[13]	191	7.81	482	90	6.15	869	420	4.13	477	449	5	624	90	3	401	266	2	381	449
		0.59	23.2	0	0.54	34.4	0	0.36	10.1	0	<0.01	1	0.5	<0.01	<0.01	0	<0.01	<0.01	0.5
Moon *et al.*[17]	15	8.70	106	37	4.82	135	140	3.66	38.0	84	6	102	37	3	71	90	2	22	84
		0.99	3.03	0	0.58	4.87	0	0.60	3.62	0	<0.01	0.09	0.5	<0.01	<0.01	0	<0.01	<0.01	0.5
Suiti *et al.*[18]	23	7.39	125	19	4.87	224	255	3.60	71.4	97	5	121	19	3	171	234	2	55	97
		0.63	4.38	0	0.48	7.51	0	0.59	3.99	0	<0.01	0.18	0.5	<0.01	<0.01	0	<0.01	<0.01	0.5
Tabor *et al.*[15]	6	7.76	49.2	12	4.61	76.9	98	3.49	23.1	26	5	39	12	3	39	51	2	14	26
		0.82	3.56	0	0.58	3.47	0	0.56	2.36	0	<0.01	<0.01	0.5	<0.01	<0.01	0	<0.01	<0.01	0.5
Tamsir *et al.*[16]	14	6.30	60.6	15	4.24	84.1	93	2.89	28.8	53	4	56	15	3	59	69	2	21	53
		0.53	2.27	0	0.50	4.24	0	0.52	2.71	0	<0.01	0.02	0.5	<0.01	<0.01	0	0.04	<0.01	0.5
All	273	9.28	969	204	6.17	1,566	907	4.36	717	844	6	1,092	204	3	879	797	2	544	844
		0.85	23.8	0	0.48	24.7	0	0.51	13.2	0	<0.01	0.99	0.5	<0.01	<0.01	0	<0.01	<0.01	0.5

Number of constructs (no.) considered in each set is shown. Numbers in each set of columns refer to cloning stages, cloning steps and PCR steps, respectively. Unoptimized solutions are represented by averages (top of each row) and s.d. (bottom of each row). Raven solutions are reported (top), along with the probability, $P(z)$, of selecting this solution randomly using a statistical z-test (bottom). P values are calculated assuming a normal distribution of assembly outcomes.

preexisting library of parts except template DNA and constrained our assembly calculations such that a maximum of six parts could be assembled per reaction for one-pot assemblies, as reactions with more parts show low efficiency.

First we considered several published sets of complex genetic constructs covering a variety of sizes, types and architectures[13–20]. For each of these sets, we determined optimized and unoptimized solutions for BioBricks (BioBricks Foundation request for comments (BBF RFC) 10), MoClo (BBF RFC 94) and Gibson assembly methods.

Assembly solutions were scored in terms of cloning stages, cloning steps and PCR steps, and the Raven solutions are compared to average unoptimized solutions (**Table 1**) and the best unoptimized solutions (**Supplementary Table 1**). Raven's solutions were significantly better than unoptimized solutions for assembly stages ($P(z) < 0.01$) for all three assembly methods for nearly all construct sets. Raven's MoClo solutions were significantly better for both cloning steps and PCR steps ($P(z) < 0.01$). Raven's solutions had significantly fewer cloning step solutions for Gibson ($P(z) < 0.01$) for all construct sets, and in only one BioBricks solution did the unoptimized plans result in fewer steps. However, as Raven's strongest scoring heuristic is cloning stages, when selecting the best assembly plan, Raven allows additional steps in favor of fewer stages. Similarly, the summary of all sets has a better cloning-step solution for BioBricks because of the inclusion of the aforementioned set, which contains by far the greatest number of constructs of the considered construct sets. For BioBricks and Gibson cloning, the number of PCRs is not optimized, so all Raven answers are equivalent to those of the unoptimized solution.

Next, to demonstrate the power of Raven solutions on an even larger scale, we used Eugene[24] to generate a set of 1,000 or more variant constructs for five separate types of constructs: DNA invertase cascade (DIC) counters, toggle switches, repressilators, transcriptional NOR gates and invertase NOR gates (**Fig. 2a** and **Supplementary Note**). Because it is common for large constructs to need tuning to achieve function, these sets contain variants to represent a spectrum of possible function and provide many opportunities to share intermediates.

To determine one unoptimized solution for each of these sets, we randomly selected 500 constructs and calculated an unoptimized, one-pot hierarchical assembly graph. We repeated this experiment 1,000 times for each of the five designs to get a distribution for each type of design and found that Raven's algorithms were able to select assembly graphs that require significantly fewer cloning steps than the average unoptimized graphs for all five designs ($P(z) < 0.001$) (**Fig. 2b**). Because these data sets were made from combinatorial part substitutions, there exist many opportunities to share cloning intermediates and assembly vectors using modular overhangs. We observed that our modular overhang assignment solutions required significantly fewer PCRs than unoptimized solutions for overhang assignment for each of the five design types ($P(z) < 0.001$) (**Fig. 2c**).

Finally, we determined how Raven's solutions performed as a function of the number of constructs under consideration. We repeated the *in silico* experiments for the five design types for variable numbers of constructs. We found that Raven's solutions significantly outperformed the unoptimized solution spaces for both cloning steps and PCR steps (**Fig. 2d**) at a small scale of 5 constructs ($P(z) < 0.001$) as well as at a larger scale of 500 constructs ($P(z) < 0.001$). As the number of constructs under consideration increases exponentially, Raven's solutions for both hierarchical assembly and overhang assignment also improve exponentially compared to unoptimized solutions (**Fig. 2d**).

Interactive assembly of genetic constructs

To highlight Raven's ability to utilize an existing library of constructs, we used it to calculate an assembly plan for six repressilator[21] constructs using an existing library with an existing overhang schema. The constructs were designed based on previously published schema using the CIDAR (http://www.cidarlab.org/) MoClo library as a resource. Design constraints

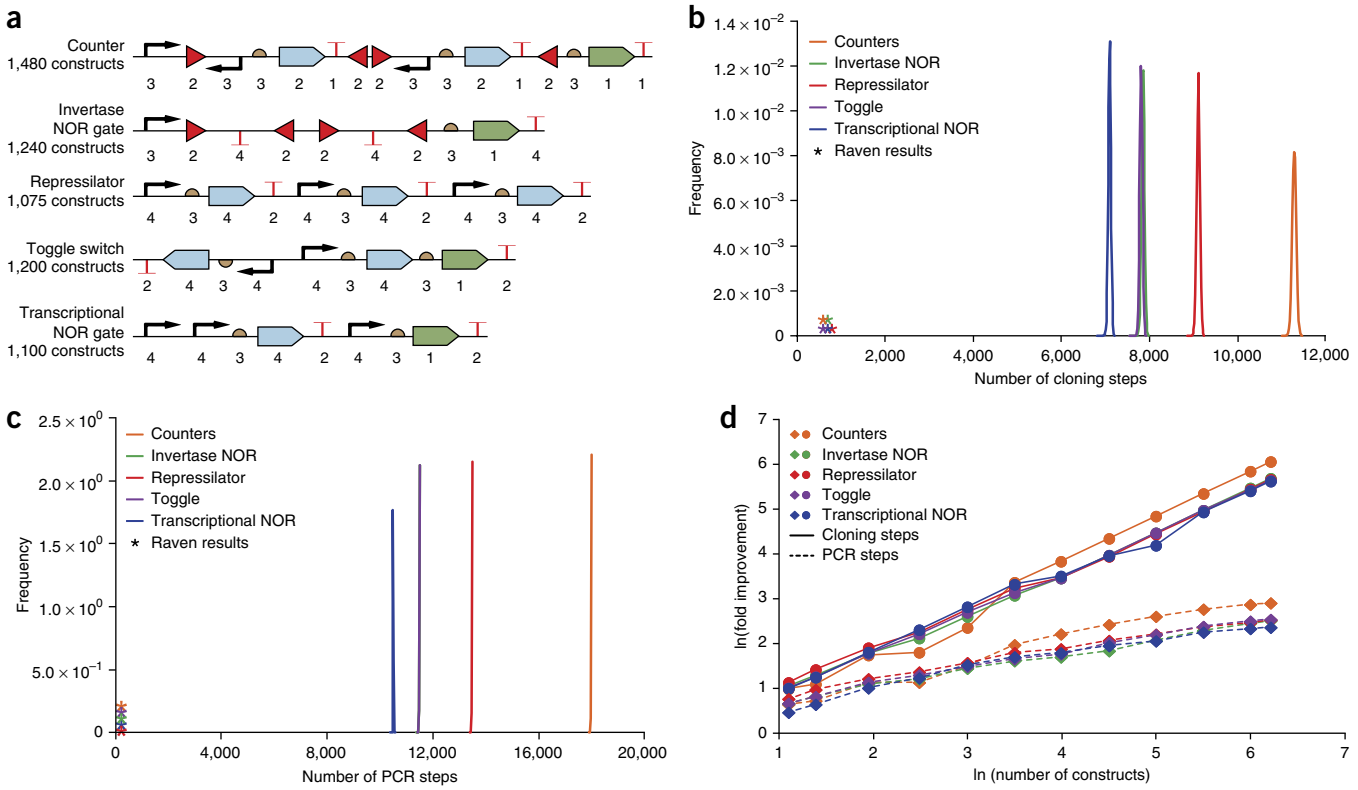

Figure 2 | *In silico* assembly with Raven. (**a**) SBOL visual representations of the DIC counter, invertase NOR gate, repressilator, toggle switch and transcriptional NOR gate constructs, indicating the number of parts we sampled at each position and the total possible construct variants after application of Eugene rules. (**b**) Cloning steps required for MoClo assembly of a 500-construct subset of each set of 1,000 or more constructs from **a**. Asterisks represent the Raven solution; other points represent unoptimized cloning step solutions. (**c**) PCR steps required for MoClo assembly given the cloning step solution in **b**. Asterisks represent the Raven solution; other points represent random PCR step solutions. (**d**) The fold improvement of Raven's solution compared to unoptimized solutions in **b**,**c** as a function of construct quantity. Raven's solutions improve as the number of constructs per assembly plan increases. PCR steps (dashed) and cloning steps (solid) are shown separately.

allowed only up to four parts per cloning step, as opposed to six. The assembly plan for these six constructs required 17 assembly steps, 2 assembly stages, 0 PCR reactions and 23 shared parts, and we successfully constructed two constructs without modification to this plan (**Supplementary Fig. 4**).

We then selected a subset of the constructs from Friedland *et al.*[14] (representing some of the largest and most complex constructs in the sets) and constructed them using Raven. We used a MoClo assembly plan (BBF RFC 94) for the DIC counter constructs, assuming a library of only template DNA and cloning vectors. The Raven-designed oligonucleotides from the assembly plan were used to amplify parts using the original constructs as template (**Supplementary Fig. 5** and Online Methods), overhang sites were chosen from a preselected set of 4-bp modular scars (**Supplementary Table 2**) and it was assumed that all cloning steps would have equivalent cloning efficiency. The assembly plan for all four constructs required 29 steps, 3 stages and 34 PCR steps (**Fig. 3a** and **Supplementary Fig. 6**). We implemented this preliminary assembly plan as specified by the human-readable instructions that Raven generated, using standard reaction conditions (Online Methods and **Fig. 3a**).

This initial plan was not successful. However, Raven has four primary mechanisms for interactively modifying assembly plans to circumvent unsuccessful cloning steps. First, Raven can detect undesirable restriction sites that can be removed with PCR. Second, intermediate clones flagged for expressing undesirable genes (such as the flp_e recombinase) or other traits (**Fig. 3b**) can be biased for or against appearing in an assembly plan (**Supplementary Fig. 7**). Third, default cloning vectors assigned to each assembly stage based on each assembly method may be substituted. Finally, cloning efficiency as a function of number of parts assembled per cloning reaction may be modified from default equivalent-efficiency values (**Supplementary Figs. 8** and **9**).

In cases where users have already started a large assembly but get stuck on unforeseen challenges, they can use the Raven redesign feature (**Fig. 3c**) to calculate a new plan. When using this feature, Raven automatically adds the successful parts into the library and forbids failed intermediates from appearing in the alternative solution. When we got stuck on the first plan for the counters, we used a redesigned solution, which required seven steps, two stages and two PCR steps (**Supplementary Fig. 10**). This plan reused the four successful intermediates from the initial plan and split up the two unsuccessful intermediates into smaller intermediates. One of these intermediates was also unsuccessful, so a second redesign with the same cost was implemented that succeeded in the cloning of all intermediates (**Supplementary Fig. 11**).

This third plan, although successful for creating all intermediates, was not successful for cloning the final constructs. This was

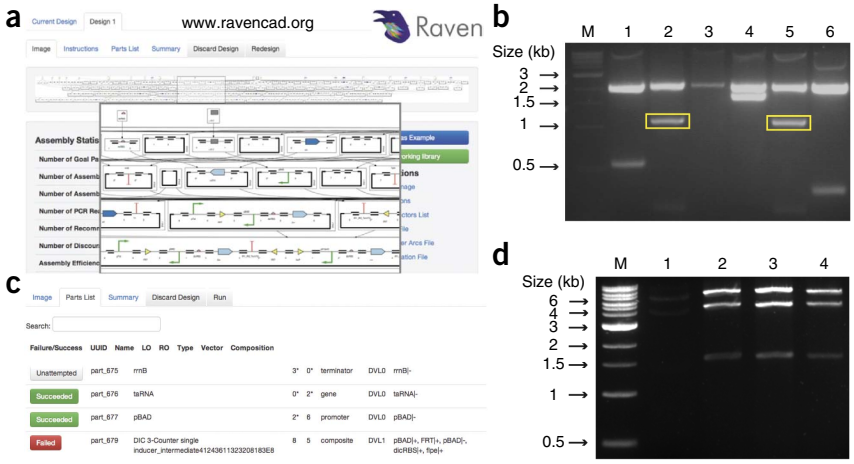

Figure 3 | Interactive assembly. (**a**) An initial assembly plan in the Raven user interface. (**b**) A SpeI restriction analysis for initial cloning intermediates that had expected bands at 0.5 kb (lane 1), 1.7 kb and 0.2 kb (lane 2), 2.1 kb (lane 3), 1.7 kb (lane 4), 1.5 kb and 2 kb (lane 5), and 3.4 kb (lane 6), with a 2.1-kb vector band (lanes 1–6). Incorrect bands (boxed in yellow) are seen for lanes 2 and 5. (**c**) The Raven user interface redesign tab records failure of these intermediates and success of all other intermediates, and generates a new plan. (**d**) A PstI restriction analysis for counter 1 (lane 1), counter 2 (lane 2), counter 4 (lane 3) and counter 6 (lane 4). Expected bands at 7 kb, 4 kb and 1.6 kb (lanes 1–4).

because BBF RFC 94 assumes the use of high-copy plasmids for all cloning steps. In this case, since it is critical that recombinases are not expressed, it was problematic to clone the final counter constructs into high-copy plasmids owing to leaky promoter behavior. To address this, we forced an extra cloning stage by requiring the construction of larger intermediates and assigned a pBAC[25] for the final cloning stage (**Supplementary Fig. 12**).

Using this plan, all cloning intermediates were constructed and used to build the final constructs successfully (**Fig. 3d**). Several of the intermediates incurred mutations as a result of cloning artifacts, but these were located at internal part junctions and the flanking junctions needed for future steps remained unaltered. Therefore, Raven cannot guarantee the production of an exact target sequence; *in vivo* recombination events are difficult to predict and outside the scope of the assembly plan. Moreover, as long as the necessary restriction sites and part junctions remain intact, the Raven plan remains valid.

DISCUSSION

Because it is not feasible for a human to design hundreds or thousands of assembly plans manually and even more difficult to produce efficient and low-cost solutions for such sets, a computational tool to automatically determine these solutions is needed. And because assembly planning instructions are necessary for liquid-handling robots and microfluidics to perform high throughput cloning and other automation techniques, the absence of an automated method to inform a robot which steps to take to assemble genetic constructs would severely limit the automation power of a larger tool pipeline.

Raven generates experimentally valid assembly plans, and, although it cannot guarantee success of any one plan or complete target sequence, it can generate new plans on the basis of some specific step failures and efficiency data. While these algorithms have the ability to incorporate feedback of reaction failures and successes to produce better solutions, they do not provide any methodology for predicting the success or failure of specific assembly steps or the construct's function. It is important to note that some standardized cloning protocols cannot be rigidly implemented to clone all constructs owing to inherent complexity of function of the constructs under consideration: some cloning challenges still must be solved by amending standard protocols and thus fall outside the purview of a protocol-agnostic assembly plan.

Finally, formal assembly files can be used to capture assembly information from previously attempted assemblies. The documentation of cloning reaction success and failure and of the path to successful assembly can be accumulated and allow easier reproduction of published work. This is particularly important because this information is often poorly documented, which hinders the ability to build on previous work. Formally documented assembly planning provides a better avenue for tracking this information, and previously attempted assemblies could be studied to develop new heuristics and bring further insight to popular molecular cloning methods.

Raven currently supports only six highly used, well-defined cloning methods, but additional systematic biases and constraints outside the tool's core heuristics can be applied to Raven's solutions by specifying forced, forbidden, recommended and discouraged intermediates and specific cloning vectors. Moreover, the principles of this approach could be expanded and further generalized to nearly to any cloning method, provided common sub-problem scoring required by dynamic programming. The generality of the algorithmic solutions and the breadth of the permitted inputs allow assembly solutions to be adapted to potentially any DNA assembly method because Raven broadly suggests how to reuse DNA libraries to build a set of genetic constructs.

METHODS

Methods and any associated references are available in the online version of the paper.

Note: Any Supplementary Information and Source Data files are available in the online version of the paper.

ACKNOWLEDGMENTS

The authors would like to thank S. Bhatia, N. Hillson, E. Oberortner and V. Vasilev for conversations regarding the algorithm development. We also thank M. Smanski (Massachusetts Institute of Technology), S. Iverson (Boston University) and the Boston University iGEM team for providing samples and for conversations regarding MoClo cloning experiments. We would like to thank the authors of work from which this work was extended and all alpha-testers of the Raven software. Finally, we would like to thank T.K. Lu (Massachusetts Institute of Technology), C. Voigt (Massachusetts Institute of Technology) and D. Endy (Stanford University) for providing samples of the genetic constructs that were used to implement assembly plans. This work has been funded by the Office of Naval Research under grant no. N00014-11-1-0725.

AUTHOR CONRIBUTIONS

E.A., J.T. and D.D. developed the algorithms. E.A. and J.T. implemented the algorithms and user interface. E.A. and T.H. designed and performed experiments.

T.H. developed standard MoClo protocols and provided materials. E.A., J.T., T.H. and D.D. wrote the paper.

COMPETING FINANCIAL INTERESTS

The authors declare competing financial interests: details are available in the online version of the paper.

1. Gibson, D.G. *et al.* Enzymatic assembly of DNA molecules up to several hundred kilobases. *Nat. Methods* **6**, 343–345 (2009).
2. Weber, E., Engler, C., Gruetzner, R., Werner, S. & Marillonet, S. A modular cloning system for standardized assembly of multigene constructs. *PLoS ONE* **6**, e16765 (2011).
3. Li, M.Z. & Elledge, S.J. Harnessing homologous recombination *in vitro* to generate recombinant DNA via SLIC. *Nat. Methods* **4**, 251–256 (2007).
4. Quan, J. & Tian, J. Circular polymerase extension cloning of complex gene libraries and pathways. *PLoS ONE* **4**, e6441 (2009).
5. Engler, C., Kandzia, R. & Marillonnet, S. A one pot, one step, precision cloning method with high throughput capability. *PLoS ONE* **3**, e3647 (2008).
6. Sarrion-Perdigones, A. *et al.* GoldenBraid: an iterative cloning system for standardized assembly of reusable genetic modules. *PLoS ONE* **6**, e21622 (2011).
7. Endy, D. Foundations for engineering biology. *Nature* **438**, 449–453 (2005).
8. Arkin, A. Setting the standard in synthetic biology. *Nat. Biotechnol.* **26**, 771–774 (2008).
9. Densmore, D. *et al.* Algorithms for automated DNA assembly. *Nucleic Acids Res.* **38**, 2607–2616 (2010).
10. Blakes, J. *et al.* A heuristic for maximizing DNA reuse in synthetic DNA library assembly. *ACS Synth. Biol.* doi:10.1021/sb400161v (20 February 2014).
11. Shetty, R.P., Endy, D. & Knight, T.F. Engineering BioBrick vectors from BioBrick parts. *J. Biol. Eng.* **2**, 5 (2008).
12. Hillson, N.J., Rosengarten, R.D. & Keasling, J. j5 DNA assembly design automation software. *ACS Synth. Biol.* **1**, 14–21 (2012).
13. Lou, C., Stanton, B., Chen, Y.-J., Munsky, B. & Voigt, C.A. Ribozyme-based insulator parts buffer synthetic circuits from genetic context. *Nat. Biotechnol.* **30**, 1137–1142 (2012).
14. Friedland, A.E. *et al.* Synthetic gene networks that count. *Science* **324**, 1199–1202 (2009).
15. Tabor, J.J. *et al.* A synthetic genetic edge detection program. *Cell* **137**, 1272–1281 (2009).
16. Tamsir, A., Tabor, J.J. & Voigt, C.A. Robust multicellular computing using genetically encoded NOR gates and chemical 'wires'. *Nature* **469**, 212–215 (2011).
17. Moon, T.S., Lou, C., Tamsir, A., Stanton, B.C. & Voigt, C.A. Genetic programs constructed from layered logic gates in single cells. *Nature* **491**, 249–253 (2012).
18. Siuti, P., Yazbek, J. & Lu, T.K. Synthetic circuits integrating logic and memory in living cells. *Nat. Biotechnol.* **31**, 448–452 (2013).
19. Bonnet, J., Yin, P., Ortiz, M.E., Subsoontorn, P. & Endy, D. Amplifying genetic logic gates. *Science* **340**, 599–603 (2013).
20. Bonnet, J., Subsoontorn, P. & Endy, D. Rewritable digital storage in live cells via engineered control or recombination directionality. *Proc. Natl. Acad. Sci. USA* **109**, 8884–8889 (2012).
21. Elowitz, M.B. & Leibler, S. A synthetic oscillatory network of transcriptional regulators. *Nature* **403**, 335–338 (2000).
22. Bhatia, S. & Densmore, D. Pigeon: a design visualizer for synthetic biology. *ACS Synth. Biol.* **2**, 348–350 (2013).
23. Peccoud, J. *et al.* Essential information for synthetic DNA sequences. *Nat. Biotechnol.* **29**, 22 (2011).
24. Bilitchenko, L. *et al.* Eugene: a domain specific language for specifying and constraining synthetic biological parts, devices, and systems. *PLoS ONE* **6**, e18882 (2011).
25. Wright, D.A. *et al.* Standardized reagents and protocols for engineering zinc finger nucleases by modular assembly. *Nat. Protoc.* **1**, 1637–1652 (2006).

ONLINE METHODS

Cloning destination vectors. The *lacZα* fragment was PCR amplified from a *lacZα*-containing cloning vector (pMJS2AF, donated by M. Smanski) and subsequently cloned into three backbones, depending on the MoClo level: level 0 used pSB1A2, level 1 used pSB1K3, and level 2 used pSB1A2. DNA containing the *lacZα* fragment was used as template for PCR reactions. PCR reactions with 5× Phusion HF buffer, 100 μM dNTPs, Phusion DNA polymerase, 5% DMSO, 1 mM $MgCl_2$ (New England BioLabs, Ipswich, MA, USA) and sterile diH2O. Reactions were performed using the following parameters: one denaturation step at 95 °C for 5 min, followed by 30 extension cycles (95 °C 20 s., 61 °C 20 s., 72 °C 15 s.), a final 5 min extension step at 72 °C and then incubation at 4 °C. PCR products over 100 bp were purified using either the QIAquick PCR Purification Kit (Qiagen Inc., Valencia, CA, USA) or GenCatch PCR Purification Kit (Epoch Life Sciences, Sugar Land, TX, USA) according to the manufacturer's protocol. PCR products and pSB1K3 and pSB1A2 vectors were digested with SpeI enzyme (NEB) according to the manufacturer's protocol using up to 500 ng DNA. Restriction digestions were purified using the QIAquick PCR Purification Kit (Qiagen) following the manufacturer's protocol. Ligation reactions were performed with T4 DNA ligase (NEB) following the manufacturer's protocol with a 3:1 insert part to vector backbone ratio.

MoClo (BBF RFC 94) cloning protocol. Each MoClo reaction had the following contents: 40 fmol of each DNA component (DNA PCR product or previously made MoClo DNA parts, and the appropriate destination vector), BsaI or BbsI (BsaI for level 1, BbsI for level 0 and level 2; NEB), high concentration T4 DNA ligase (C M1794, Promega, Madison, WI, USA), T4 DNA ligase buffer (Promega) and sterile, diH2O. Reactions performed using the following parameters: 25–35 cycles (37 °C 1.5 min, 16 °C 3 min), followed by 50 °C for 5 min and 80 °C for 10 min and then a hold at 4 °C until transformed. Level 0 reactions were done for 25 cycles, while level 1 and 2 reactions were done for 25–30 cycles. Transformations were done into Alpha Select Gold Efficiency (Bioline USA Inc., Taunton, MA, USA), DH5α-Z1 and epi300 competent *Escherichia coli* cells. Transformations were heat-shocked at 42 °C for 45 s and recovered in SOC medium for 1 h at 37 °C, 300 r.p.m.

Primer design. Primers for MoClo (BBF RFC 94) assembly were designed in the following format for parts larger than 24 bp: NN-[BpiI recognition site]-NN-1234-part-5768-NN-[BpiI recognition site]-NN. Forward primers: 5′-NN-GAAGAC-NN-[overhang sequence]-[first 24 bp of part]-3′. Reverse primers: 5′-[last 24 bp of gene]-[overhang sequence]-NN-GTCTTC-NN-3′. For parts smaller than 24 bp, annealing primers were designed that adhere to the preceding format.

Assembly algorithms. All algorithms in Raven are implemented in Java. The Raven user interface is implemented in JavaScript using jQuery and Bootstrap libraries. Scripts for recommended, discouraged, forbidden and required parts are implemented in Eugene. Automatic graphical assembly pictures are generated using graphviz libraries and automatically generated construct glyphs from http://pigeoncad.org/. Pseudocode for Raven algorithms detailed in the **Supplementary Software**.