TABLES

Table S1: Time- and cost-savings estimates for j5 compared with traditional cloning and DNA synthesis

Task		Traditional cloning[1] and robotics[4]	DNA Synthesis[2]	SLIC/Gibson/CPEC[3] with j5 and robotics[4]	Golden Gate with j5 and robotics[4]
A	Hands-on time[5]	5.7 hr design 6.7 hr lab	0.1 hr design[6] 0.1 hr order	1.2 hr design 4.4 hr lab	0.4 hr design 4.4 hr lab
	Materials cost	$212	$2345	$174	$174
	Total cost	$1452	$2365	$734	$654
	Total duration	2.5 weeks	2 months	2 weeks	2 weeks
B	Hands-on time	3.1 hr design 4.1 hr lab	0.1 hr design 0.1 hr order	0.5 hr design 4.2 hr lab	0.2 hr design 4.2 hr lab
	Materials cost	$73	$785	$64	$64
	Total cost	$793	$805	$534	$504
	Total duration	2 weeks	3 weeks	2 weeks	2 weeks
C	Hands-on time	176.3 hr design 767.0 hr lab	20.3 hr design 20.3 hr order	N/A	29.0 hr design[7] 82.9 hr lab[8]
	Materials cost	$27,813[9]	$533,535		$18,686[9]
	Total cost	**$122,143**	**$537,585**		**$29,876**
	Total duration	**11 months**	**2.3 months**		**1.5 months**
D	Hands-on time	67.9 hr design 229.9 hr lab	18.0 hr design 18.0 hr order	N/A	7.8 hr design[7] 74.9 hr lab[8]
	Materials cost	$8,102[9]	$137,310		$5,834[9]
	Total cost	**$37,882**	**$140,910**		**$14,104**
	Total duration	**3.5 months**	**1 month**		**1.3 months**

DNA construction tasks:

A) Metabolic pathway construction: vector backbone 4kb + 2 promoters (200 bp each) + 5 ORFs (with RBS, 1000 bp each) + 1 terminator (100 bp)
B) Protein chimera construction: vector backbone 4kb + 3 domains (500 bp each)
C) Combinatorial library (243 constructs = 3^5 [3 orthologs for each of the 5 ORFs]) of task A) above
D) Combinatorial library (216 constructs = 6^3 [6 orthologs for each of the 3 domains]) of task B) above

[1]Best-case scenario (viable restriction enzyme choices). Binary ligations with maximum intermediate re-use.
[2]Assumes perfect parallel process scaling, $0.39/bp, $50/construct for custom destination vector.
[3]SLIC/Gibson/CPEC can be used interchangeably to assemble the same j5-designed fragments (see Results).
[4]No automation of transformation/clonal isolation process
[5]Fully burdened-labor cost estimate: $100/hr. Design time includes sequence validation.
[6]Design time refers here to DNA sequence file manipulation (i.e., pasting each insert into the vector backbone)
[7]Almost exclusively sequence validation
[8]Almost exclusively transformation/clonal isolation processes
[9]Dominated by DNA sequencing service costs ($2.50 per 800-bp read)

METHODS

All DNA sequences and *E. coli* strains identified as JPUB_XXXXXX have been accessioned in the JBEI public registry (http://public-registry.jbei.org) and are available upon request. j5 design files, related oligo lists, and chromatogram sequence trace files are linked to the JPUB entries and can be downloaded or viewed there.

Plasmid pNJH00010 construction

Plasmid pNJH00010 (JPUB_000226) was constructed as designed by j5 (see the j5 design file pNJH00010.csv that is attached to the pNJH00010 registry entry). Briefly, plasmid pBbS8c-RFP (*1*) (JPUB_000041) was purified from *E. coli* strain JBEI-2566 (JPUB_000199) by Qiagen miniprep kit (per manufacturer's instructions), 3.5 ug was digested with 25 units each *EcoRI* and *XhoI* (Fermentas), and the vector backbone fragment gel purified (Qiagen). Insert parts were generated by PCR using Phusion polymerase (New England Biosciences, manufacturer's instructions) and primers RDR00130/132, RDR00134/136, and RDR00138/140 (see the pNJH00010.csv design file for the primer specifications). For the initial SLIC chew-back step, 360 ng of each assembly piece was digested with T4 DNA polymerase. Subsequently, 275 ng digested backbone was combined with equimolar parts 1 & 3, and 4-fold molar excess of part 2 due to its small size to mitigate the risk of excessive exonuclease degradation. The CPEC and Gibson assembly reactions used 100 ng vector, equimolar parts 1 and 3, and 4-fold molar excess part 2. Assembly reactions proceeded according to published methods (*2-4*) and 5 μL each reaction was transformed into 100 μL Keasling-1484 (*E. coli* DH10b Δ*araFGH* Δ*araE* P_{CP18}::araE (*1*)) chemically competent cells, yielding strain JBEI-2804 (JPUB_000235). Transformants were selected for on LB-agar plates with 30 μg/mL chloramphenicol, and screened by PCR (primers RDR00001 and RDR00142, see the DNA_oligo_file.csv attached to the pNJH00010 registry entry for the primer specifications) for the correct inserts. Plasmid DNA was isolated by miniprep (Qiagen kit) and the success of plasmid assembly verified by Sanger sequencing of plasmid DNA (sequencing trace files are attached to the Seq. Analysis tab of the pNJH00010 registry entry).

Construction of plasmids pRDR00001 – pRDR00008

Eight GFPuv-signal peptide plasmid variants (pRDR00001 – pRDR00008; JPUB_000227-234) were assembled by the Golden Gate method (*5, 6*) from linear PCR products amplified from pNJH00010. Specifications of all oligos, PCR reactions, and assembly combinations are provided in the j5 design files attached to the respective JBEI public registry entries. Briefly, the vector backbone was amplified with primers incorporating either a *sig1* or *sig2* signal peptide, and the *gfpuv* open reading frame was amplified with primers incorporating either a long or short linker sequence at the 5' end of the gene and either a regular or enhanced *ssrA* tag at the 3' end of the gene. PCR amplicons were purified by Qiagen column (manufacturer's instructions) and DNA concentrations determined by nanodrop. Golden Gate assembly reactions were set up in 15 μL containing 100 ng vector backbone and equimolar insert, $1/10^{th}$ volume 10X T4 DNA ligase buffer, 2,000 cohesive end units T4 DNA ligase (i.e. 1 μL high concentration) and 10 units *BsaI* (all enzymes, New England Biosciences). Reactions proceeded at 37˚C for 1 hour, followed by 5 minute incubations at 50˚C and 80˚C. Five μL of each reaction was transformed into *E. coli* strain Keasling-1484 (*1*), yielding strains JBEI-2747-9,2923,2751-3,2755 (JPUB_000236-43) and cells plated on LB-agar with 30 μg/mL chloramphenicol. Transformants were screened by colony PCR, plasmid DNA isolated by miniprep (Qiagen kit) and the success of plasmid assembly verified by Sanger sequencing of plasmid DNA (sequencing trace files are attached to the Seq. Analysis tab of the pRDR00001 – pRDR00008 registry entries).

Markerless deletion of clpX

In order to test the efficacy of the ssrA degradation tag on the GFPuv variants, the protease encoding gene *clpX* was removed from the Keasling-1484 (*1*) genome by a markerless deletion strategy (*7*) (see also Figure S8). Keasling-1484 was transformed with pREDI (*7*) (JPUB_000019), encoding the lambda red recombinase machinery under the control of P_{bad} and *I-SceI* homing endonuclease under the control of P_{rhaB} with a temperature sensitive origin of replication, resulting in strain JBEI-2948 (JPUB_000253). A j5-designed linear deletion cassette containing a kanamycin resistance marker, *sacB* sucrose counter-selection marker, I-SceI cleavage motif, and genomic sequence flanking *clpX* for targeted recombination regions was amplified using Phusion polymerase and primers RDR00044 and RDR00045 from template pSKI (*7*) (JPUB_000270), resulting in DNA part ΔclpX_cassette (JPUB_000255) (see the j5 design file DclpX_j5_design.csv that is attached to the ΔclpX_cassette registry entry). Because our aim was to generate a linear cassette rather than a circularized plasmid, these primers were manually modified from the j5 design file to remove the 5' overlapping regions

introduced by the software. Due to the length of the oligos and formation of inhibitory primer dimers, successful amplification required the addition of 1M betaine, 5% DMSO, and 50 μM 7deaza-GTP (*8*). Amplicons were incubated with 10 units *DpnI* for 1 hour at 37˚C, gel purified, and concentrated to ~ 100 ng/uL. Four μL (400 ng) of deletion cassette was transformed into electrocompetent JBEI-2948, cells were recovered at 30˚C, and plated on LB-agar with 100 μg/mL ampicillin, 50 μg/mL kanamycin, and 10 mM arabinose. Four recovered clones were screened by colony PCR with primers flanking (RDR00050/051) and specific to (RDR00050/052) the inserted deletion cassette (see the DNA_oligo_file.csv attached to the ΔclpX_cassette registry entry for the primer specifications). Those colonies yielding the expected PCR products were replica plated on LB-agar with ampicillin and kanamycin versus LB-agar with ampicillin and sucrose. One of the two clones demonstrating kanamycin resistance and sucrose sensitivity was archived as JBEI-3080 (JPUB_000269). The insertion cassette was then excised by growing JBEI-3080 to $O.D._{600}$ ~ 0.4 in LB plus ampicillin and 10 mM rhamnose over three 10% dilutions and plating on LB-agar plus ampicillin, rhamnose and 5% sucrose. Recovered colonies were replica plated on agar containing either sucrose or kanamycin, and strain JBEI-3083 (JPUB_000254) was selected for sucrose growth and kanamycin sensitivity. Markerless deletion was confirmed by colony PCR using flanking and insert specific primers, as above, and by Sanger sequencing of the resulting amplicons (sequencing trace files are attached to the Seq. Analysis tab of the JBEI-3083 registry entry).

GFP expression from plasmid variants

Plasmid pNJH00010 and each GFPuv plasmid variant were transformed by heat shock into chemically competent JBEI-3083, resulting in strains JBEI-3144,3133-40 (JPUB_000244-52). Recovered colonies were grown overnight at 42˚C to cure pREDI and restreaked on LB-agar plus chloramphenicol. Cells harboring the GFPuv plasmid but not pREDI were selected by colony PCR. Four colonies per plasmid variant for each host JBEI-3083 and Keasling-1484 (*1*) were grown overnight in deep well 96-well plates containing 1 mL LB + 30 μg/mL chloramphenicol per well. These were diluted 1:100, grown to an average O.D. ~ 0.2, induced with 5 mM arabinose, and grown an additional 6 hours. Cells were pelleted, washed twice in M9 minimal media, and GFP fluorescence and optical density measured in duplicate in a SpectroMax Plus384 (Molecular Devices) plate reader.

FIGURES

A

Part Name	Part Source (Sequence Display ID)	Reverse Compliment?	Start (bp)	End (bp)
pS8c-vector_backbone	pj5_00001	FALSE	2016	1238
BMC_nterm_sig_pep	BMC_nterm_sig_pe	FALSE	4	54
ccmN_nterm_sig_pep	ccmN_nterm_sig_p	FALSE	4	60
short_gly_ser_linker	BMC_nterm_sig_pe	FALSE	55	78
long_gly_ser_linker	BMC_nterm_sig_pe	FALSE	55	102
GFPuv	pj5_00001	FALSE	1242	1952
ssrA_tag_5prime	ssrA_tag_GFPuv	FALSE	1	15
ssrA_tag_enhanced_5prime	ssrA_tag_enhance	FALSE	1	21
ssrA_tag_3prime	ssrA_tag_GFPuv	FALSE	16	33

B

Sequence File Name	Format
pj5_00001.gb	Genbank
BMC_nterm_sig_pep_GFPuv.gb	Genbank
ccmN_nterm_sig_pep_GFPuv.gb	Genbank
ssrA_tag_GFPuv.gb	Genbank
ssrA_tag_enhanced_GFPuv.gb	Genbank

C

(>Bin) or Part Name	Direction	Forced Assembly Strategy?	Forced Relative Overhang Position?	Direct Synthesis Firewall?
pS8c-vector_backbone	forward		2	
BMC_nterm_sig_pep	forward			TRUE
long_gly_ser_linker	forward			
GFPuv	forward	unspecified	2	
ssrA_tag_5prime	forward	Embed_in_primer_reverse		TRUE
ssrA_tag_3prime	forward	Embed_in_primer_forward		

D

(>Bin) or Part Name	Direction	Forced Assembly Strategy?	Forced Relative Overhang Position?	Direct Synthesis Firewall?
>vector_backbone				
pS8c-vector_backbone	forward			
>nterm_sig_pep				TRUE
BMC_nterm_sig_pep	forward			
ccmN_nterm_sig_pep	forward			
>gly_ser_linker				
long_gly_ser_linker	forward			
short_gly_ser_linker	forward			
>GFPuv				
GFPuv	forward			
>ssrA_tag_5prime		Embed_in_primer_reverse		TRUE
ssrA_tag_5prime	forward	Embed_in_primer_reverse		
ssrA_tag_enhanced_5prime	forward	Embed_in_primer_reverse		
>ssrA_tag_3prime		Embed_in_primer_forward		
ssrA_tag_3prime	forward	Embed_in_primer_forward		

E

```
// Rules preventing multiple instances of the same gene

Rule r1(HMGS_SC_CO NOTMORETHAN 1);
Rule r2(HMGR_SC_CO NOTMORETHAN 1);
Rule r3(HMGS_SA NOTMORETHAN 1);
Rule r4(HMGR_SA NOTMORETHAN 1);
Rule r5(MK_SC_CO NOTMORETHAN 1);
Rule r6(PMK_SC_CO_f NOTMORETHAN 1);
Rule r7(ispA-O NOTMORETHAN 1);
Rule r8(idi-O NOTMORETHAN 1);
Rule r9(atoB NOTMORETHAN 1);

// Rules preventing multiple instances of gene orthologs

Rule r10(HMGS_SC_CO NOTWITH HMGS_SA);
Rule r11(HMGR_SC_CO NOTWITH HMGR_SA);

// Rules forcing use of one gene with another

Rule r12(HMGS_SC_CO WITH HMGR_SC_CO);
Rule r13(HMGS_SA WITH HMGR_SA);
Rule r14(HMGR_SC_CO WITH HMGS_SC_CO);
Rule r15(HMGR_SA WITH HMGS_SA);
```

F

Parameter Name	Value	Default Value	Description
MASTEROLIGONUMBEROFDIGITS	5	5	The default number of digits used to number an oligo; e.g. j5 00001 primer description uses 5 digits
MASTERPLASMIDNUMBEROFDIGITS	5	5	The default number of digits used to number a plasmid; e.g. pj5 00001 uses 5 digits
GIBSONOVERLAPBPS	26	26	The minimum number of bps for SLIC/Gibson/CPEC overlaps (should be an even number); this is also the starting design length for the annealing portion of primers
MAXIMUMOLIGOLENGTHBPS	110	110	The maximum oligo length to be ordered
MINIMUMFRAGMENTSIZEGIBSONBPS	250	250	The minimum fragment size for SLIC/Gibson assembly
GOLDENGATEOVERHANGBPS	4	4	The number of bps of the overhang resulting from the Golden-gate type IIs endonuclease digestion
GOLDENGATERECOGNITIONSEQ	GGTCTC	GGTCTC	The Golden-gate type IIs endonuclease recognition site sequence
GOLDENGATETERMINIEXTRASEQ	TTTTTTGGTCTCA	TTTTTTGGTCTCA	The extra 5' sequence required at each end of a Golden-gate assembly piece; e.g. NNNNNNGGTCTCN for BsaI (Eco31I)
MAXIMUM_IDENTITIES_GOLDEN_GATE_OVERHANGS_COMPATIBLE	2	2	The maximum number of tolerable non-gapped aligned identities between compatible overhang sequences for Golden-gate assembly
OLIGOSYNTHESISCOSTPERBPUSD	0.1	0.1	The oligo synthesis cost per bp ($US)
OLIGOPAGEPURIFICATIONCOSTPERPIECEUSD	40	40	The PAGE-purification cost per oligo ($US)
OLIGOMAXLENGTHNOPAGEPURIFICATIONREQUIREDBPS	60	60	The maximum oligo length that does not require PAGE-purification
MINIMUMPCRPRODUCTBPS	100	100	The minimum PCR product size
DIRECTSYNTHESISCOSTPERBPUSD	0.39	0.39	The cost per bp to do direct synthesis ($US)
DIRECTSYNTHESISMINIMUMCOSTPERPIECEUSD	159	159	The minimum cost of synthesis per piece ($US)
PRIMER_GC_CLAMP	2	2	Primer3 parameter: length of the desired GC clamp (Primer3 default is 0)
PRIMER_MIN_SIZE	18	18	Primer3 parameter: the minimum length of a primer (Primer3 default is 18)
PRIMER_MAX_SIZE	36	36	Primer3 parameter: the maximum length of a primer (Primer3 default is 27; maximum is 36)
PRIMER_MIN_TM	60	60	Primer3 parameter: the minimum primer Tm (Primer3 default is 57)
PRIMER_MAX_TM	70	70	Primer3 parameter: the mamimum primer Tm (Primer3 default is 63)
PRIMER_MAX_DIFF_TM	5	5	Primer3 parameter: the mamimum primer pair difference in Tm (Primer3 default is 100)
PRIMER_TM_SANTALUCIA	1	1	Primer3 parameter: use the Santalucia formula for calculating Tms (1 = TRUE; 0 = FALSE) (Primer3 default is 0 (FALSE))
PRIMER_SALT_CORRECTIONS	1	1	Primer3 parameter: use the salt correction formula for calculating Tms (1 = TRUE; 0 = FALSE) (Primer3 default is 0 (FALSE))
PRIMER_DNA_CONC	250	250	Primer3 parameter: DNA concentration to use when calculating Tms in micromolar (IDT uses 250; Primer3 default is 50)
MISPRIMING_3PRIME_BOUNDARY_BP_TO_WARN_IF_HIT	4	4	Only warn of mispriming if the BLAST hit between the primer and the template contains the 3' end of the primer (within this number of bp)
MISPRIMING_MIN_TM	45	45	The minimum approximate Tm to consider a significant mispriming event
MISPRIMING_SALT_CONC	0.05	0.05	The salt concentration used when estimating the mispriming Tm in Molar
MISPRIMING_OLIGO_CONC	0.00000025	0.00000025	The oligo concentration used when estimating the mispriming Tm in Molar
OUTPUT_SEQUENCE_FORMAT	Genbank	Genbank	The output sequence file format. Options are: "Genbank", "FASTA", or "jbei-seq".

G

Target Part Ordering/Selection/Strategy				
Part Order	ID Number	Name	Direction	Strategy
0	0	pNJH0010_vector_backbone	forward	PCR
1	1	BMC_nterm_sig_pep	forward	Embed_in_primer_reverse
2	4	long_gly_ser_linker	forward	Embed_in_primer_forward
3	5	GFPuv	forward	PCR
4	6	ssrA_tag_5prime	forward	Embed_in_primer_reverse
5	8	ssrA_tag_3prime	forward	Embed_in_primer_forward

Figure S1. Example j5 input. Zoom in with PDF display software as desired to improve legibility. **(A)** Example parts list CSV input file, stylized for clarity. The user must specify the name of each part (first column) to be included in the DNA assembly design process, the sequence source for each part (second column), if each part should be defined as the reverse compliment (i.e. bottom strand; third column) of the specified subsequence, and where each part starts (fourth column) and ends (fifth column) in its specified sequence source. **(B)** Example sequences list CSV input file, stylized for clarity. The user must specify the filename (first column) and format (second column) for each sequence file to be utilized in the DNA assembly design process. **(C-D)** Example target part order list CSV input files, stylized for clarity. **(C)** Single construct target part

order list example. The user must specify the sequential order (from top to bottom, first column) of the parts to be assembled together, the direction of each part (second column), and as well as optionally whether to force j5 to use a particular assembly strategy for each part (third column), whether to force j5 to use a particular relative overhang position (in bp, Golden Gate assembly only) following each part (fourth column), and whether to place a direct DNA synthesis firewall following each part (fifth column). **(D)** Combinatorial library target part order list example. The user must specify the sequential order (from top to bottom, first column) of the combinatorial part bins to be assembled together (each denoted by a leading '>' character, grey rows) and the parts within each bin (immediately following each bin name). Other columns are as in (C). **(E)** Eugene design specification rules example file. When designing assemblies with j5, it is possible to set design specification rules that limit the total number of times a given part appears in a given assembly (**`NOTMORETHAN`** statements, rules r1-r9), if two given parts should not appear together in the same assembly (**`NOTWITH`** statements, rules r10-r11), or if two given parts should only appear together in the same assembly (**`WITH`** statement, rules r12-r15). The design specification rules understood by j5 (shown as those shown here) are derived from (and are a strict subset of) the Eugene biological design specification computer language (*9, 10*). **(F)** Example j5 parameters CSV input file, stylized for clarity. The user may optionally change parameter values by modifying entries in the second column. Default values (third column) and descriptions (fourth column) are provided as a reference for each parameter. **(G)** Example Target Part Ordering/Selection/Strategy section of an assembly design CSV output file, stylized for clarity. The assembly order (top to bottom) and direction (fourth column), and the optimal assembly strategy (as determined by Algorithm S1, fifth column), are shown for each part to be assembled (third column).

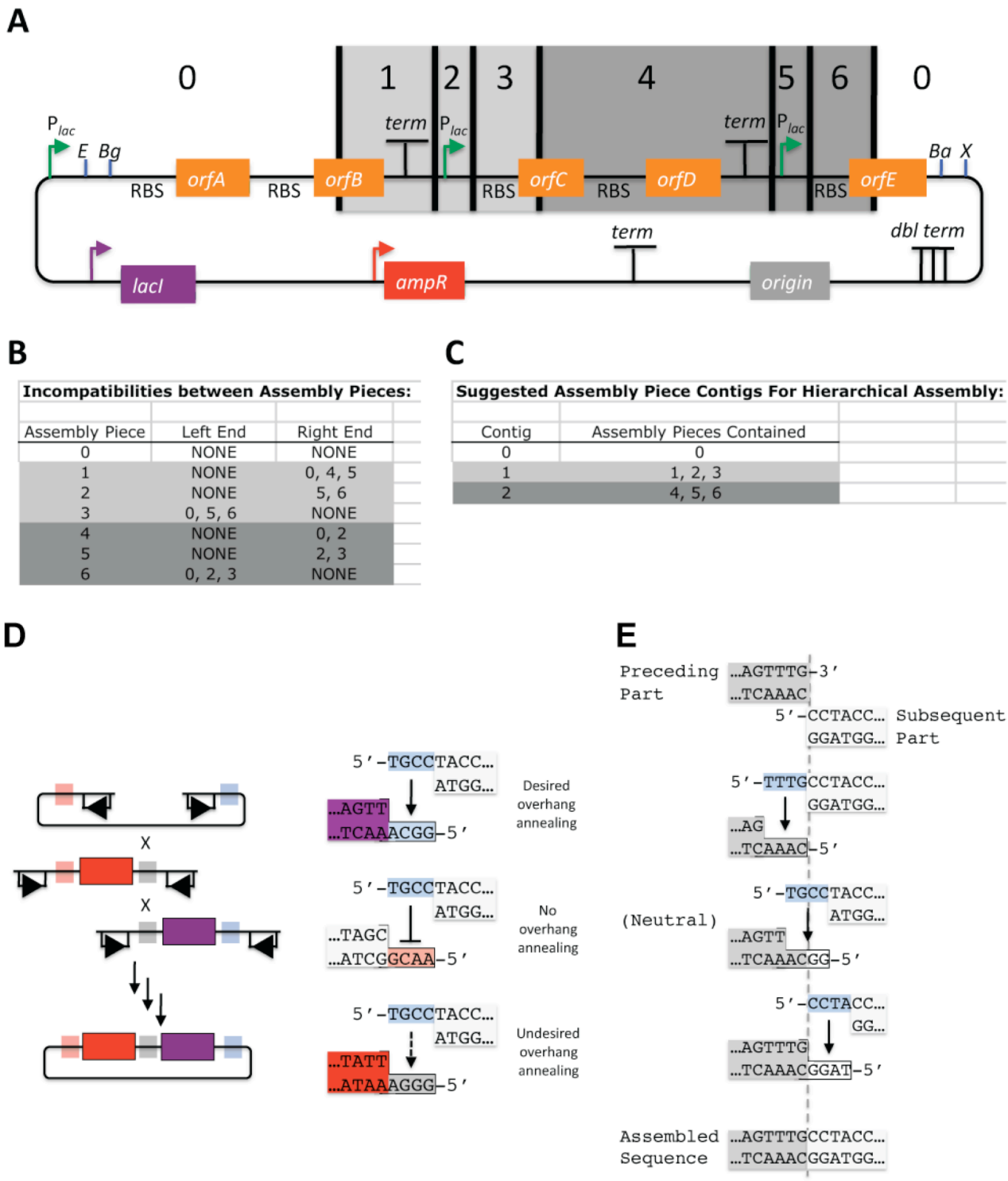


Figure S2. The search for compatible sets of homology and overhang sequences. **(A-C)** SLIC, Gibson, and CPEC assembly piece putative incompatibilities. **(A)** Schematic of a hypothetical plasmid to be assembled, demarcated at the boundaries of DNA assembly pieces “0” through “6”. Pieces “2” and “5” are identical, containing the P_{lac} promoter, which is also internally present in piece “0”. **(B)** j5-predicted assembly piece incompatibilities. The right end of assembly piece “4” (which contains 3’ flanking sequence homologous to the 5’ end of the P_{lac} in piece “5”) is incompatible with assembly pieces “0” and “2”. **(C)** j5 utilizes Algorithm S3 to design a hierarchical assembly strategy that circumvents these incompatibilities by first SOEing together pieces “1”, “2” and “3” (Contig 1), and separately “4”, “5”, “6” (Contig 2), and then assembling together Contig 0 (fragment “0”) with Contig 1 and Contig 2. **(D-E)** Search for optimal Golden Gate overhang sequences. j5 utilizes Algorithm S4 to search through all possible combinations of putative overhang sequences (constrained by the maximum oligo size and the extents of combinatorial sequence identity about each assembly junction) and selects the set of overhangs that 1) are compatible with themselves and each other, and that 2) are as

neutral as possible. **(D)** An example three-part (vector backbone, red part, purple part) Golden Gate assembly with red, grey, and blue overhang junctions. Directional type IIs endonuclease (e.g. *BsaI*) recognition sites are schematically indicated by rectangular boxes below the DNA strand, with arrowheads pointing to the adjacent cute site. Checking for overhang compatibility consists of making sure that (top right) each single stranded overhang sequence (e.g. “TGCC”, blue top overhang) is cohesive with its desired cognate partner (e.g. “GGCA”, blue bottom overhang), but not with itself nor with off-target sequences (e.g. “AACG”, red bottom overhang, middle right). If an overhang is sufficiently cohesive with an off-target sequence (e.g. “GGGA”, grey bottom overhang, bottom right, 3 cohesive base-pairs), the set of overhang sequences is declared incompatible and rejected. **(E)** Three possible Golden Gate overhang sequence options (blue sequences) are shown for a particular assembly junction that each result in the scar-less assembly of the preceding part (top left, grey sequence) and the subsequent part (top right, white sequence) into the desired assembled sequence (bottom). The first overhang sequence selection “TTTG” (top) draws all four base pairs from the preceding part (a negative 2-bp relative overhang position), the second selection “TGCC” (middle) draws two base pairs from the preceding part and two from the subsequent part (a neutral overhang), and the third selection “CCTA” (bottom) draws all four base pairs from the subsequent part (a positive 2-bp relative overhang position).

A

Oligo Name	Length	Tm	Tm (3' only)	Sequence
j5_00001_(gfpuv_sigpep_5prime)_pure_forward	34	60.208	60.208	ATGAGTAAAGGAGAAGAACTTTTCACTGGAGTTG
j5_00002_(RBS)_(gfpuv_sigpep_5prime)_forward	85	70.595	60.208	CTGTTTCTCCATACCCGTTTTTTTGGGAATTTTTAAGAAGGAGATATACATATGAGTAAAGGAGAAGAACTTTTCACTGGAGTTG
j5_00003_(gfpuv_sigpep_5prime)_pure_reverse	29	60.947	60.947	TCGAGTTTGTGTCCGAGAATGTTTCCATC
j5_00004_(gfpuv_sigpep_5prime)_(gfpuv_sigpep_XhoI_silent)_reverse	49	67.149	60.947	TGTGTGAGTTATAGTTGTATTCGAGTTTGTGTCCGAGAATGTTTCCATC
j5_00005_(gfpuv_sigpep_middle)_pure_forward	35	60.084	60.084	TACAACTATAACTCACACAATGTATACATCACGGC
j5_00006_(gfpuv_sigpep_middle)_forward	57	68.942	60.084	CATTCTCGGACACAAACTCGAATACAACTATAACTCACACAATGTATACATCACGGC
j5_00007_(gfpuv_sigpep_middle)_pure_reverse	23	56.6	56.6	GATCCATCTTCAATGTTGTGGCG
j5_00008_(gfpuv_sigpep_middle)_(gfpuv_sigpep_BamHI_silent)_revers	40	68.06	56.6	GGTCTGCTAGTTGAACAGATCCATCTTCAATGTTGTGGCG
j5_00009_(gfpuv_sigpep_3prime)_pure_forward	36	60.484	60.484	GTTCAACTAGCAGACCATTATCAACAAAATACTCCA
j5_00010_(gfpuv_sigpep_3prime)_forward	53	67.614	60.484	ACATTGAAGATGGATCTGTTCAACTAGCAGACCATTATCAACAAAATACTCCA
j5_00011_(gfpuv_sigpep_3prime)_pure_reverse	25	60.961	60.961	TTTAGCGATCGGGGAACATGCTCTG
j5_00012_(gfpuv_sigpep_3prime)_reverse	57	71.658	60.961	GTTTTATTTGATGCCTGGAGATCCTTACTCGATTTAGCGATCGGGGAACATGCTCTG
j5_00013_(pS8c-gfpuv_sig.pep_vector_backbone)_pure_forward	34	61.437	61.437	AGTAAAGGAGAAGAACTTTTCACTGGAGTTGTCC
j5_00014_(pS8c-gfpuv_sig.pep_vector_backbone)_forward	49	68.643	61.437	TTTTTTGGTCTCAGCAGTAAAGGAGAAGAACTTTTCACTGGAGTTGTCC
j5_00015_(pS8c-gfpuv_sig.pep_vector_backbone)_pure_reverse	36	57.245	57.245	ATGTATATCTCCTTCTTAAAAATTCCCAAAAAACG
j5_00016_(pS8c-gfpuv_sig.pep_vector_backbone)_reverse	50	63.16	57.245	TTTTTTGGTCTCATATGTATATCTCCTTCTTAAAAATTCCCAAAAAACG
j5_00017_(umuD_tag)_pure_forward	34	66.451	66.451	ATGTTGTTTATCAAGCCTGCGGATCTCCGCGAAA
j5_00018_(umuD_tag)_forward	50	70.629	66.451	TTTTTTGGTCTCACATATGTTGTTTATCAAGCCTGCGGATCTCCGCGAAA
j5_00019_(umuD_tag)_pure_reverse	20	61.777	61.777	GCCACCCAGATCGATGCGCT
j5_00020_(umuD_tag)_reverse	35	68.828	61.777	TTTTTTGGTCTCACTGCCACCCAGATCGATGCGCT
j5_00021_(pS8c-vector_backbone)_pure_forward	28	58.876	58.876	TAAATCGAGTAAGGATCTCCAGGCATCA
j5_00022_(ssrA_tag_3prime)_(pS8c-vector_backbone)_forward	59	74.575	58.876	TTTTTTGGTCTCAAACTATGCGCTGGCGGCGTAAATCGAGTAAGGATCTCCAGGCATCA
j5_00023_(pS8c-vector_backbone)_pure_reverse	32	54.025	54.025	CATATGTATATCTCCTTCTTAAAAATTCCCAA
j5_00024_(pS8c-vector_backbone)_(BMC_nterm_sig_pep)_reverse	100	68.717	54.025	TTTTTTGGTCTCACACCCATATTTTTTAAAACTTCATTGATTATTTGTTCTAATAAAGCGTTATTTTCCATATGTATATCTCCTTCTTAAAAATTCCCAA
j5_00025_(GFPuv)_pure_forward	31	60.539	60.539	AAAGGAGAAGAACTTTTCACTGGAGTTGTCC
j5_00026_(long_gly_ser_linker)_(GFPuv)_forward	92	80.194	60.539	TTTTTTGGTCTCAGGTGGCAGTGGTAGCGGGAGCTCGGGTGGCTCAGGCTCTGGTTCCAGTAAAGGAGAAGAACTTTTCACTGGAGTTGTCC

B

Direct Synthesis Name	Alias	Contents	Length	Sequence
dsj5_00001		(BMC_nterm_sig_pep)_(long_gly_ser_linker)_(GFPuv)_(ssrA_tag_5prime)	853	GGAGATATACAT

C

PCR Reactions			Forward Oligo		Reverse Oligo										
ID Number	Primary Template	Alternate Template	ID Number	Name	ID Number	Name	First Target Part	Last Target Part	Note	Mean Oligo Tm	Delta Oligo Tm	Mean Oligo Tm (3' only)	Delta Oligo Tm (3' only)	Length	Sequence
0	pS8c-gfpuv_sigpep		0	j5_00001	2	j5_00003	0	0	PCR	56.4505	4.851	56.4505	4.851	4522	TAAATCGAGTAA
1	pS8c-gfpuv_sigpep		4	j5_00005	6	j5_00007	3	3	PCR	62.919	4.76	62.919	4.76	711	AAAGGAGAAGAA
2	PCR/SOE Reaction 0	pS8c-gfpuv_sigpep	1	j5_00002	3	j5_00004	5	1	PCR	73.348	5.546	56.4505	4.851	4621	GGCGAACGATGA
3	PCR/SOE Reaction 1	pS8c-gfpuv_sigpep	5	j5_00006	7	j5_00008	2	4	PCR	78.8955	1.031	62.919	4.76	809	CAATGAAGTTTT

D

Assembly Pieces (SLIC/Gibson/CPEC)											
ID Number	Type	PCR ID Number	First Target Part	Core Target Part	Last Target Part	CPEC Tm Previous (3' only)	CPEC Tm Next (3' only)	Overlap with Previous (bps)	Overlap with Next (bps)	Length	Sequence
0	PCR	2	5	0	1	62.805	61.36	26	39	4621	GGCGAACGATGAAAA
1	PCR	3	2	3	4	60.083	62.805	39	26	809	CAATGAAGTTTTAAA

E

Assembly Pieces (Golden-gate)										
ID Number	Type	PCR ID Number	First Target Part	Core Target	Last Target	Overhang with Previous	Overhang with Next	Relative Overhang Position	Length	Sequence
0	PCR	2	5	0	1	AACT	GGTG	2	4621	TTTTTTGGTCTC
1	PCR	3	2	3	4	GGTG	AACT	2	804	TTTTTTGGTCTC

F

Combinations of Assembly Pieces						
Variant			Bin 0		Bin 1	
Number	Name	Assembly Method	Part(s)	Assembly Piece ID Number	Part(s)	Assembly Piece ID Number
0	pj5_00003	Golden-gate	(ssrA_tag_3prime)_(pNJH0010_vector_backbone)_(BMC_nterm_sig_pep)	0	(long_gly_ser_linker)_(GFPuv)_(ssrA_tag_5prime)	1
1	pj5_00004	Golden-gate	(ssrA_tag_3prime)_(pNJH0010_vector_backbone)_(BMC_nterm_sig_pep)	0	(long_gly_ser_linker)_(GFPuv)_(ssrA_tag_enhanced_5prime)	2
2	pj5_00005	Golden-gate	(ssrA_tag_3prime)_(pNJH0010_vector_backbone)_(BMC_nterm_sig_pep)	0	(short_gly_ser_linker)_(GFPuv)_(ssrA_tag_5prime)	3
3	pj5_00006	Golden-gate	(ssrA_tag_3prime)_(pNJH0010_vector_backbone)_(BMC_nterm_sig_pep)	0	(short_gly_ser_linker)_(GFPuv)_(ssrA_tag_enhanced_5prime)	4
4	pj5_00007	Golden-gate	(ssrA_tag_3prime)_(pNJH0010_vector_backbone)_(ccmN_nterm_sig_pep)	5	(long_gly_ser_linker)_(GFPuv)_(ssrA_tag_5prime)	1
5	pj5_00008	Golden-gate	(ssrA_tag_3prime)_(pNJH0010_vector_backbone)_(ccmN_nterm_sig_pep)	5	(long_gly_ser_linker)_(GFPuv)_(ssrA_tag_enhanced_5prime)	2
6	pj5_00009	Golden-gate	(ssrA_tag_3prime)_(pNJH0010_vector_backbone)_(ccmN_nterm_sig_pep)	5	(short_gly_ser_linker)_(GFPuv)_(ssrA_tag_5prime)	3
7	pj5_00010	Golden-gate	(ssrA_tag_3prime)_(pNJH0010_vector_backbone)_(ccmN_nterm_sig_pep)	5	(short_gly_ser_linker)_(GFPuv)_(ssrA_tag_enhanced_5prime)	4

G

Plasmid Name	Alias	Contents	Length	Sequence
pj5_00001		(pBbS8c-RFP_EcoRI_XhoI_vector_backbone)_(RBS)_(gfpuv_sigpep_5prime)_(gfpuv_sigpep_XhoI_silent)_(gfpuv_sigpep_middle)_(gfpuv_sigpep_BamHI_silent)_(gfpuv_sigpep_3prime)	5299	GACGTCTTATGA
pj5_00002		(pS8c-gfpuv_sig.pep_vector_backbone)_(umuD_tag)	5422	GACGTCTTATGA
pj5_00003		(pS8c-vector_backbone)_(BMC_nterm_sig_pep)_(long_gly_ser_linker)_(GFPuv)_(ssrA_tag_5prime)_(ssrA_tag_3prime)	5365	TCATTTTCGCCA
pj5_00004		(pS8c-vector_backbone)_(BMC_nterm_sig_pep)_(long_gly_ser_linker)_(GFPuv)_(ssrA_tag_enhanced_5prime)_(ssrA_tag_3prime)	5371	TCATTTTCGCCA
pj5_00005		(pS8c-vector_backbone)_(BMC_nterm_sig_pep)_(short_gly_ser_linker)_(GFPuv)_(ssrA_tag_5prime)_(ssrA_tag_3prime)	5341	TCATTTTCGCCA
pj5_00006		(pS8c-vector_backbone)_(BMC_nterm_sig_pep)_(short_gly_ser_linker)_(GFPuv)_(ssrA_tag_enhanced_5prime)_(ssrA_tag_3prime)	5347	TCATTTTCGCCA
pj5_00007		(pS8c-vector_backbone)_(ccmN_nterm_sig_pep)_(long_gly_ser_linker)_(GFPuv)_(ssrA_tag_5prime)_(ssrA_tag_3prime)	5371	TCATTTTCGCCA
pj5_00008		(pS8c-vector_backbone)_(ccmN_nterm_sig_pep)_(long_gly_ser_linker)_(GFPuv)_(ssrA_tag_enhanced_5prime)_(ssrA_tag_3prime)	5377	TCATTTTCGCCA
pj5_00009		(pS8c-vector_backbone)_(ccmN_nterm_sig_pep)_(short_gly_ser_linker)_(GFPuv)_(ssrA_tag_5prime)_(ssrA_tag_3prime)	5347	TCATTTTCGCCA
pj5_00010		(pS8c-vector_backbone)_(ccmN_nterm_sig_pep)_(short_gly_ser_linker)_(GFPuv)_(ssrA_tag_enhanced_5prime)_(ssrA_tag_3prime)	5353	TCATTTTCGCCA

Figure S3. Example j5 output. Zoom in with PDF display software as desired to improve legibility. **(A)** Example master oligos list CSV input file, stylized for clarity. The user may specify the names (first column), lengths (in bp, second column), full-length (third column) and template-annealing 3' end (fourth column) melting temperatures, and DNA sequences (fifth column) of oligos in the user's collection. Subsequent to the design process, j5 appends to this list the new oligo(s) to be ordered, following the naming and numbering convention the user specifies (first column). **(B)** Example master direct DNA syntheses list CSV input file, stylized for clarity. The user may specify the name (first column), alias (second column), contents (part names enclosed in parentheses separated by underscores; third column), length (in bp, fourth column), and DNA sequence (fifth column) of directly synthesized DNA sequences in the user's collection. Subsequent to the design process, j5 appends to this list the new direct synthesis sequence(s) to be ordered, following the naming and numbering convention used in the first column. **(C)** Example PCR Reactions section of an assembly design CSV output file, stylized for clarity. The primary (second column) and alternative (third column) templates, forward (fifth column) and reverse (seventh column) primers (as determined by Algorithm S2), full-length (mean, eleventh

column; delta, twelfth column) and template-annealing 3' end (mean, thirteenth column; delta, fourteenth column) primer melting temperatures, product length (in bp, fifteenth column) and sequence (sixteenth column), are shown for each PCR reaction. The parts contained within each PCR product (from first part, eighth column, to the last part, ninth column, corresponding to that shown in Figure S1F), and a note (tenth column) indicating whether the PCR product should be SOE'd together with adjacent assembly pieces prior to the DNA assembly process, are also shown for each PCR reaction. **(D-E)** Example Assembly Pieces section of an assembly design CSV output file, stylized for clarity. **(D)** SLIC/Gibson/CPEC assembly. The upstream (seventh column) and downstream (eighth column) flanking homology sequence melting temperatures, corresponding upstream (ninth column) and downstream (tenth column) flanking homology sequence overlap lengths, length (in bp, eleventh column) and sequence (twelveth column), are shown for each assembly piece. The parts contained (from first part, fourth column, to the last part, sixth column, corresponding to that shown in Figure S1F), derivation (e.g. PCR or digest; second column), and corresponding PCR reaction number (if applicable, third column, corresponding to that shown in (C) are also shown for each assembly piece. **(E)** Golden Gate assembly. The upstream (seventh column) and downstream (eighth column) top strand overhang sequences (as determined by Algorithm S4), and the downstream relative overhang position (in bp, ninth column; see Figure S2E), are shown for each assembly piece. Other columns are as in (D). **(F)** Example Combinations of Assembly Pieces section of an assembly design CSV output file, stylized for clarity. The assembly method (third column) and the assembly piece in each combinatorial bin corresponding to the variant (fourth and columns thereafter) is shown for each plasmid variant to be constructed (second column). **(G)** Example master plasmids list CSV input file, stylized for clarity. The user may optionally specify the names (first column), aliases (second column), contents (part names enclosed in parentheses separated by underscores, third column), lengths (in bp, fourth column), and DNA sequences (fifth column) of plasmids in the user's collection. Subsequent to the design process, j5 appends to this list the new plasmid(s) to be constructed, following the naming and numbering convention the user specifies (first column).

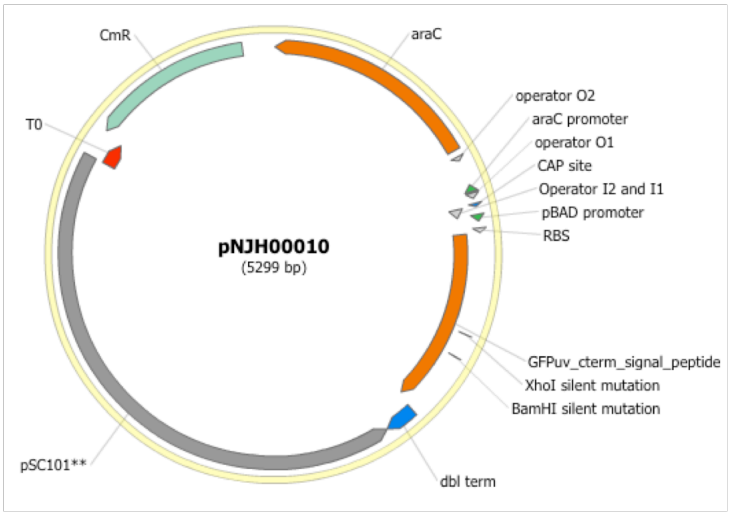


Figure S4. Plasmid map of pNJH00010 derived from the Genbank-format sequence file resulting from j5-designed SLIC/Gibson/CPEC assembly (Figure 2).

A

PLATE ID	PLATE WELL	LIQUID TYPE	VOLUME (uL)
oligos_plate	A01	j5_00001_(gfpuv_sigpep_5prime)_pure_forward	100
oligos_plate	B01	j5_00002_(RBS)_(gfpuv_sigpep_5prime)_forward	100
oligos_plate	C01	j5_00003_(gfpuv_sigpep_5prime)_pure_reverse	100
oligos_plate	D01	j5_00004_(gfpuv_sigpep_5prime)_(gfpuv_sigpep_XhoI_silent)_reverse	100
oligos_plate	E01	j5_00005_(gfpuv_sigpep_middle)_pure_forward	100
oligos_plate	F01	j5_00006_(gfpuv_sigpep_middle)_forward	100
oligos_plate	G01	j5_00007_(gfpuv_sigpep_middle)_pure_reverse	100

B

Parameter Name	Value	Default Value	Description
MAXDELTATEMPERATUREADJACENTZONES	5	5	The maximum difference in temperature (in C) between adjacent zones on the thermocycler block
MAXDELTATEMPERATUREREACTIONOPTIMUMGIVENACCEPTABLE	5	5	The maximum acceptable difference in temperature (in C) between the optimal annealing temperature of a PCR reaction, and the annealing temperature of the thermocycler block zone it is sitting in
MAXMCSTEPSPERSCONE	1000	1000	The maximum number of Monte-Carlo steps attempted per thermocycler block zone
MAXWELLVOLUMEMULTIWELLPLATE	100	100	The maximum liquid volume (in uL) that a well in the multi-well plate can hold
MCTEMPERATUREFINAL	0.0001	0.0001	The final temperature at the end of the Monte-Carlo simulated annealing run (in arbitrary reduced units)
MCTEMPERATUREINITIAL	0.1	0.1	The initial temperature in the beginning of the Monte-Carlo simulated annealing run (in arbitrary reduced units)
MINPIPETTINGVOLUME	5	5	The minimum pipetting volume (e.g. for a robotics platform) (in uL)
NCOLUMNSMULTIWELLPLATE	12	12	The number of columns in the multi-well plate
NROWSMULTIWELLPLATE	8	8	The number of rows in the multi-well plate
TRIALDELTATEMPERATURE	0.1	0.1	The Monte-Carlo step trial change in temperature for a thermocycler block zone
WELLSPERTHERMOCYCLERZONE	16	16	The number of wells per thermocycler block zone
ZONESPERTHERMOCYCLERBLOCK	6	6	The number of zones per thermocycler block

C

PCR Reactions:

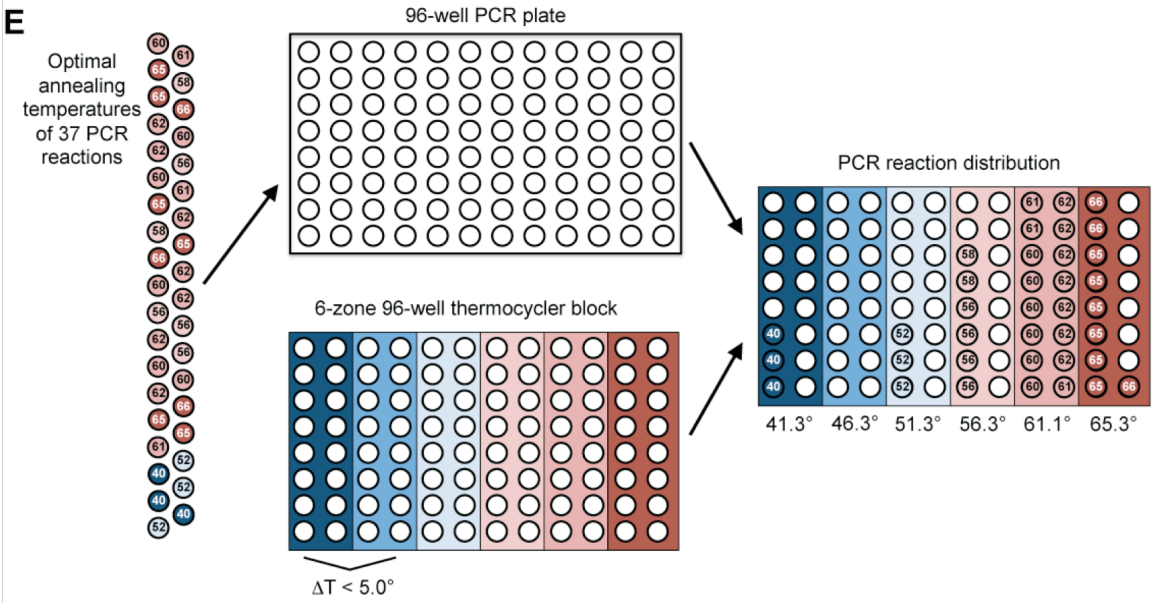
Index	TEMPLATE_PLATE	TEMPLATE_WELL	TEMPLATE_VOLUME	PRIMER1_PLATE	PRIMER1_WELL	PRIMER1_VOLUME	PRIMER2_PLATE	PRIMER2_WELL	PRIMER2_VOLUME	WATER_VOLUME	OUTPUT_PLATE	OUTPUT_WELL	REACTION_NUMBER	OPTIMAL_ANNEALING_TEMP	THERMOCYCLER_BLOCK	THERMOCYCLER_ZONE	THERMOCYCLER_ZONE_ANNEALING_TEMP	EXPECTED_SIZE
1	templates_plate	B07	5	oligos_plate	F05	5	oligos_plate	H03	5	0	mysassemblyfilelist_condensed_PCR_PLATE_1	A01	10	58	0	0	56	4522
2	templates_plate	B01	5	oligos_plate	F03	5	oligos_plate	H05	5	0	mysassemblyfilelist_condensed_PCR_PLATE_1	B01	12	59	0	0	56	4521
3	templates_plate	A01	5	oligos_plate	F01	5	oligos_plate	H07	5	0	mysassemblyfilelist_condensed_PCR_PLATE_1	C01	11	58.6	0	0	56.6	4520
4	templates_plate	A01	5	oligos_plate	G01	5	oligos_plate	G01	5	0	mysassemblyfilelist_condensed_PCR_PLATE_1	A05	1	58.6	0	2	56.6	66
5	templates_plate	A01	5	oligos_plate	F01	5	oligos_plate	H01	5	0	mysassemblyfilelist_condensed_PCR_PLATE_1	B05	4	58.6	0	2	56.6	137
6	templates_plate	B01	5	oligos_plate	G02	5	oligos_plate	G02	5	0	mysassemblyfilelist_condensed_PCR_PLATE_1	A07	6	59.2	0	3	59.2	5196
7	templates_plate	B01	5	oligos_plate	F02	5	oligos_plate	H02	5	0	mysassemblyfilelist_condensed_PCR_PLATE_1	B07	8	59.2	0	3	59.2	5325
8	templates_plate	A01	5	oligos_plate	A01	5	oligos_plate	C01	5	0	mysassemblyfilelist_condensed_PCR_PLATE_1	A09	0	62.2	0	4	62.5	425
9	templates_plate	A01	5	oligos_plate	B01	5	oligos_plate	D01	5	0	mysassemblyfilelist_condensed_PCR_PLATE_1	B09	3	62.2	0	4	62.5	496
10	templates_plate	A01	5	oligos_plate	A02	5	oligos_plate	C02	5	0	mysassemblyfilelist_condensed_PCR_PLATE_1	C09	2	62.5	0	4	62.5	299
11	templates_plate	A01	5	oligos_plate	B02	5	oligos_plate	D02	5	0	mysassemblyfilelist_condensed_PCR_PLATE_1	D09	5	62.5	0	4	62.5	308
12	templates_plate	B01	5	oligos_plate	A04	5	oligos_plate	C03	5	0	mysassemblyfilelist_condensed_PCR_PLATE_1	E09	11	62.5	0	4	62.5	711
13	templates_plate	B01	5	oligos_plate	B04	5	oligos_plate	D04	5	0	mysassemblyfilelist_condensed_PCR_PLATE_1	F09	13	62.5	0	4	62.5	804
14	templates_plate	B01	5	oligos_plate	B04	5	oligos_plate	D04	5	0	mysassemblyfilelist_condensed_PCR_PLATE_1	G09	14	62.5	0	4	62.5	815
15	templates_plate	B01	5	oligos_plate	F04	5	oligos_plate	D04	5	0	mysassemblyfilelist_condensed_PCR_PLATE_1	H09	15	62.5	0	4	62.5	790
16	templates_plate	B01	5	oligos_plate	A05	5	oligos_plate	D05	5	0	mysassemblyfilelist_condensed_PCR_PLATE_1	A10	7	63.8	0	5	63.8	136
17	templates_plate	C01	5	oligos_plate	A05	5	oligos_plate	D05	5	0	mysassemblyfilelist_condensed_PCR_PLATE_1	A10	17	63.8	0	5	63.8	136
18	templates_plate	C01	5	oligos_plate	B05	5	oligos_plate	D05	5	0	mysassemblyfilelist_condensed_PCR_PLATE_1	B11	9	63.8	0	5	63.8	197

D

Thermocycler Blocks:

BLOCK ID	BLOCK ZONE	BLOCK_ZONE_ANNEALING_TEMP
0	0	56
0	1	57.5
0	2	58.6
0	3	59.2
0	4	62.5
0	5	63.8

E



F

Index	control1TEMPLATE_VOLUME	control1PRIMER1_VOLUME	control1PRIMER2_VOLUME	TEMPLATE_PLATE	TEMPLATE_WELL	TEMPLATE_VOLUME	PRIMER1_PLATE	PRIMER1_WELL	PRIMER1_VOLUME	PRIMER2_PLATE	PRIMER2_WELL	PRIMER2_VOLUME	WATER_VOLUME	OUTPUT_PLATE	OUTPUT_WELL
1	Normal	Normal	Normal	templates_plate	2	5	oligos_plate	21	5	oligos_plate	23	5	0	mysassemblyfilelist_condensed_PCR_PLATE_1	1
2	Normal	Normal	Normal	templates_plate	2	5	oligos_plate	22	5	oligos_plate	24	5	0	mysassemblyfilelist_condensed_PCR_PLATE_1	2
3	Normal	Normal	Normal	templates_plate	2	5	oligos_plate	23	5	oligos_plate	31	5	0	mysassemblyfilelist_condensed_PCR_PLATE_1	3
4	Normal	Normal	Normal	templates_plate	1	5	oligos_plate	5	5	oligos_plate	7	5	0	mysassemblyfilelist_condensed_PCR_PLATE_1	17
5	Normal	Normal	Normal	templates_plate	1	5	oligos_plate	6	5	oligos_plate	8	5	0	mysassemblyfilelist_condensed_PCR_PLATE_1	18
6	Normal	Normal	Normal	templates_plate	2	5	oligos_plate	13	5	oligos_plate	15	5	0	mysassemblyfilelist_condensed_PCR_PLATE_1	33
7	Normal	Normal	Normal	templates_plate	2	5	oligos_plate	14	5	oligos_plate	16	5	0	mysassemblyfilelist_condensed_PCR_PLATE_1	34
8	Normal	Normal	Normal	templates_plate	1	5	oligos_plate	1	5	oligos_plate	3	5	0	mysassemblyfilelist_condensed_PCR_PLATE_1	65
9	Normal	Normal	Normal	templates_plate	1	5	oligos_plate	2	5	oligos_plate	4	5	0	mysassemblyfilelist_condensed_PCR_PLATE_1	66
10	Normal	Normal	Normal	templates_plate	1	5	oligos_plate	9	5	oligos_plate	11	5	0	mysassemblyfilelist_condensed_PCR_PLATE_1	67
11	Normal	Normal	Normal	templates_plate	1	5	oligos_plate	10	5	oligos_plate	12	5	0	mysassemblyfilelist_condensed_PCR_PLATE_1	68
12	Normal	Normal	Normal	templates_plate	2	5	oligos_plate	25	5	oligos_plate	27	5	0	mysassemblyfilelist_condensed_PCR_PLATE_1	69
13	Normal	Normal	Normal	templates_plate	2	5	oligos_plate	26	5	oligos_plate	28	5	0	mysassemblyfilelist_condensed_PCR_PLATE_1	70
14	Normal	Normal	Normal	templates_plate	2	5	oligos_plate	24	5	oligos_plate	29	5	0	mysassemblyfilelist_condensed_PCR_PLATE_1	71
15	Normal	Normal	Normal	templates_plate	2	5	oligos_plate	30	5	oligos_plate	28	5	0	mysassemblyfilelist_condensed_PCR_PLATE_1	72
16	Normal	Normal	Normal	templates_plate	2	5	oligos_plate	30	5	oligos_plate	29	5	0	mysassemblyfilelist_condensed_PCR_PLATE_1	73
17	Normal	Normal	Normal	templates_plate	3	5	oligos_plate	17	5	oligos_plate	19	5	0	mysassemblyfilelist_condensed_PCR_PLATE_1	81
18	Normal	Normal	Normal	templates_plate	3	5	oligos_plate	18	5	oligos_plate	20	5	0	mysassemblyfilelist_condensed_PCR_PLATE_1	82

Figure S5. Example multi-well plate j5 input and output. Zoom in with PDF display software as desired to improve legibility. **(A)** Example multi-well plate CSV input file, stylized for clarity. The user must specify the volume (in μL; fourth column) for each liquid component (third column) for each well (second column) for each plate that will be utilized in the DNA assembly process. **(B)** Example downstream automation parameters CSV input file, stylized for clarity. Default values (third column) and descriptions (fourth column) are provided as a reference for each parameter name (first column). The user may change parameter values by modifying the entries in the second column. **(C)** PCR Reactions section of a distribute PCR reactions CSV output file, stylized for clarity. The plate (second column), well (third column), and volume (in μL; fourth column) of each template; the plate (fifth column), well (sixth column), and volume (in μL; seventh column) of each forward primer; the plate (eighth column), well (ninth column), and volume (in μL; tenth column) of each reverse primer; the

volume of water to be added (eleventh column); the destination plate (twelfth column) and well (thirteenth column); the optimal annealing temperature (fifteenth column); the destination thermocyler block (sixteenth column), zone (seventeenth column), and annealing temperature (eighteenth column); and the expected product size (in bp, nineteenth column) are shown for each PCR reaction (fourteenth column, corresponding to that shown in Figure S3C). **(D)** Thermocycler Blocks section of a distribute PCR reactions CSV output file, stylized for clarity. The annealing temperature (third column) for each zone (second column) in each thermocycle block (first column). **(E)** Optimal distribution of PCR reactions across thermo-cycler annealing temperature gradients. Given the optimal annealing temperatures of each of the PCR reactions required for an assembly process and the thermocycler block gradient constraints (e.g. each neighboring zone must differ by less than 5 °C, as shown, see (B), j5 optimizes the thermocycler block annealing temperature gradient(s), and distributes the PCR reactions across multi-well plate(s) that will be placed in these optimized gradient(s). **(F)** NextGen (eXeTek) Expression workstation control CSV output file, stylized for clarity. The NextGen-specific template (second column), forward primer (third column), and reverse primer (fourth column) control parameters; the plate (fifth column), well index (sixth column), and volume (in μL; seventh column) of each template; the plate (eighth column), well index (ninth column), and volume (in μL; tenth column) of each forward primer; the plate (eleventh column), well index (twelfth column), and volume (in μL; thirteenth column column) of each reverse primer; the volume of water to be added (fourteenth column); and the destination plate (fifteenth column) and well index (sixteenth column) are shown for each PCR reaction.

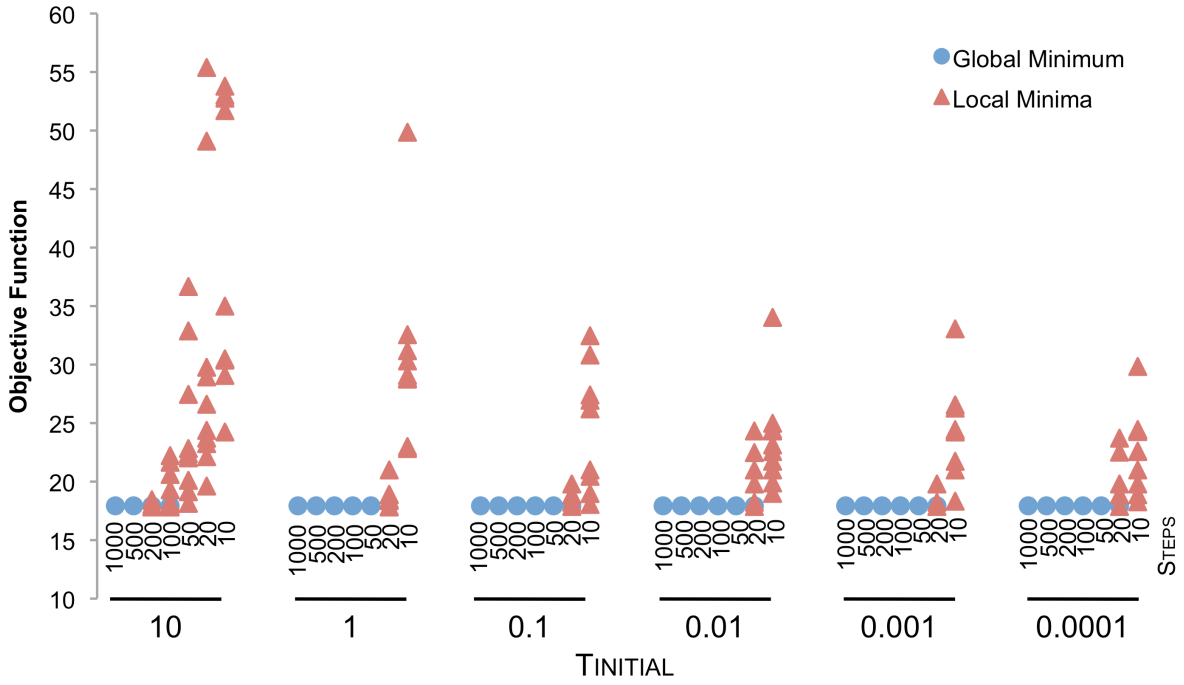


Figure S6. Algorithm S5 convergence as a function of MAXMCSTEPSPERZONE and MCTEMPERATUREINITIAL parameters for the PCR reactions shown in Figure S5E. Algorithm S5 was run 10 times for each choice of MAXMCSTEPSPERZONE (here "STEPS") and MCTEMPERATUREINITIAL (here "TINITIAL"), with all other parameters set to their respective j5 default values. The best objective function encountered for each run is plotted with either a red triangle indicating a non-global minima, or a blue circle indicating the global minimum. All runs with 50 or more MAXMCSTEPSPERZONE (with the exception of MCTEMPERATUREINITIAL = 10, which required 500 or more MAXMCSTEPSPERZONE) identified the global minimum. The default parameters for j5 are MCTEMPERATUREINITIAL = 0.1 and MAXMCSTEPSPERZONE = 1000.

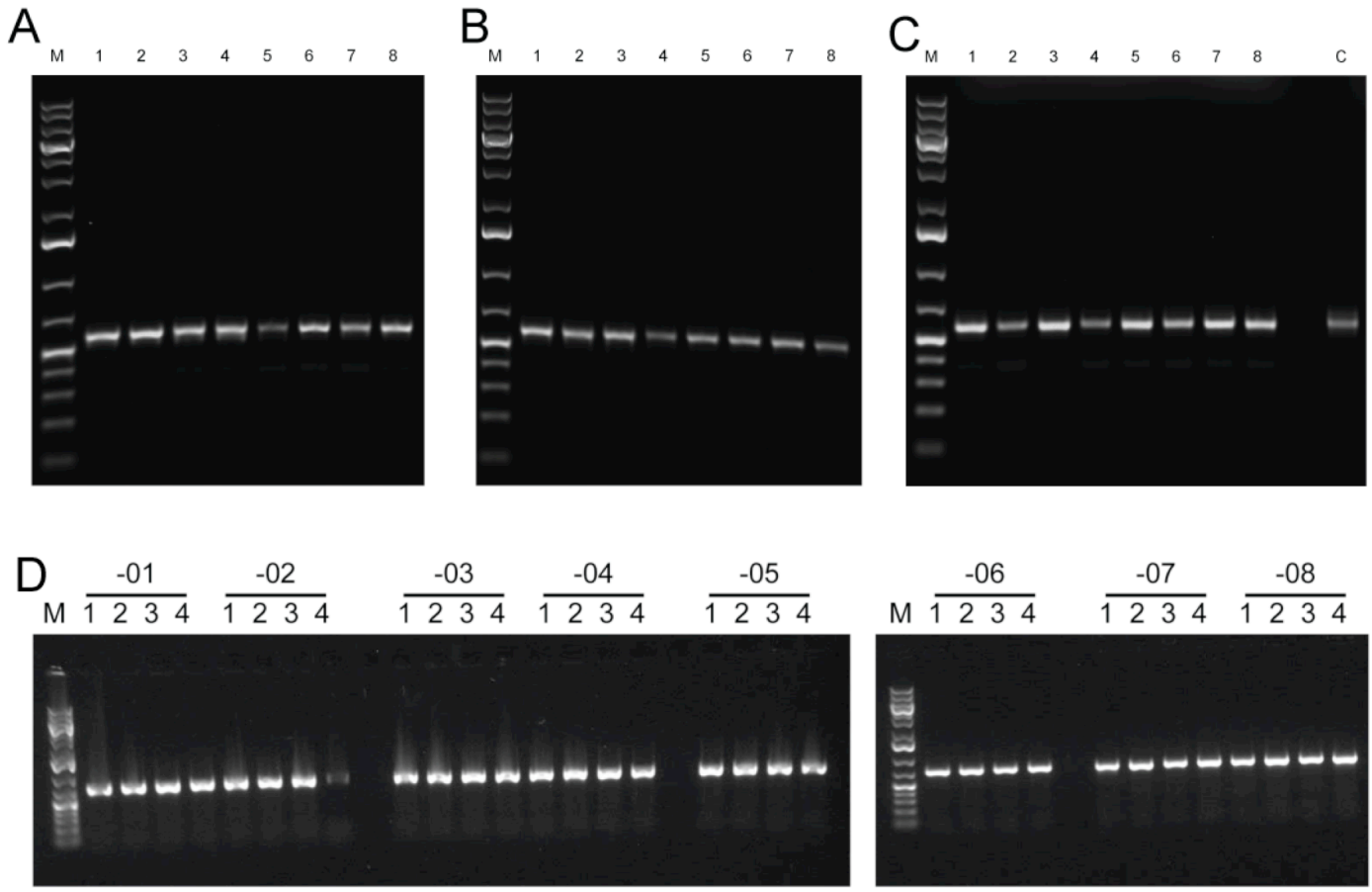
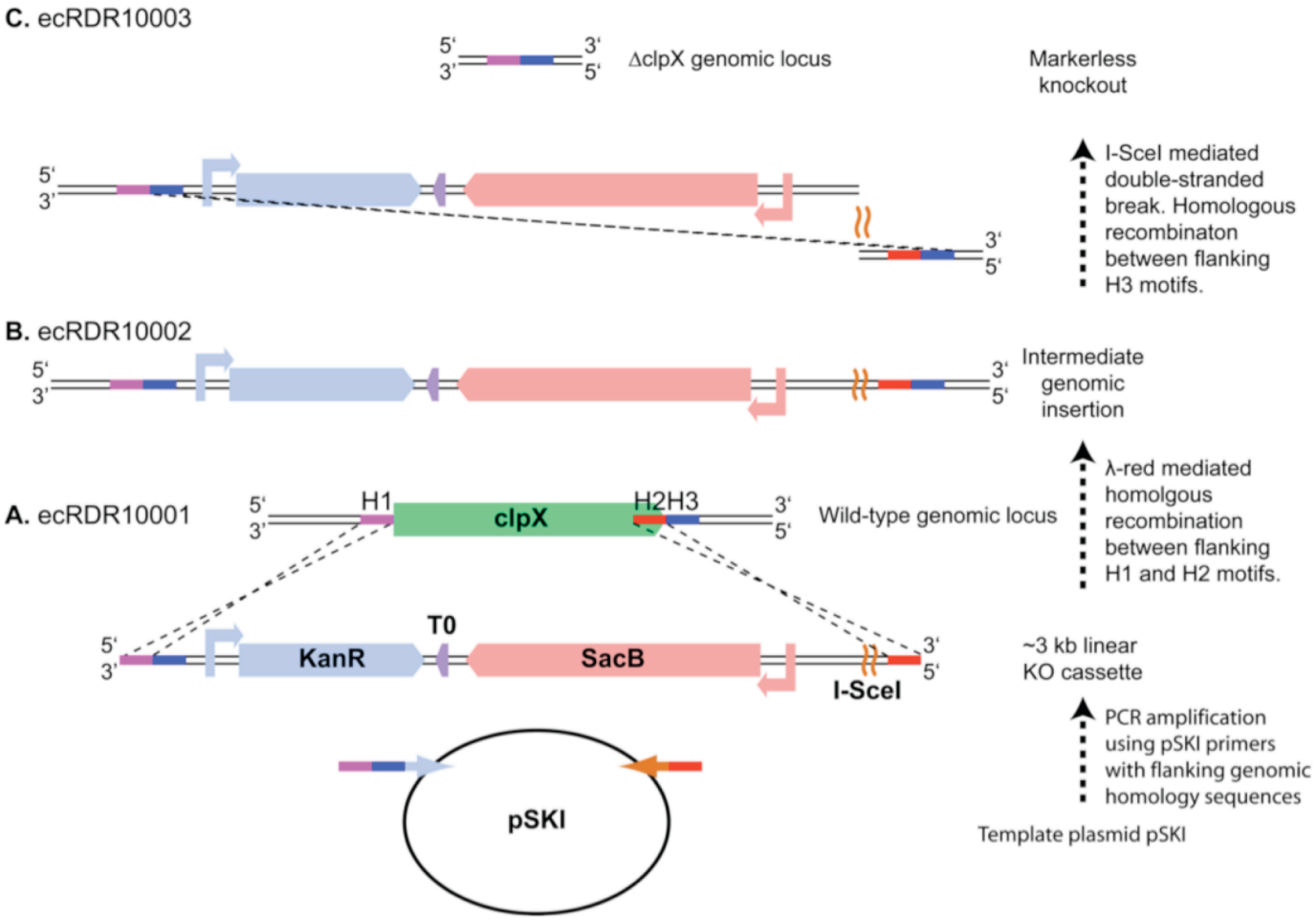
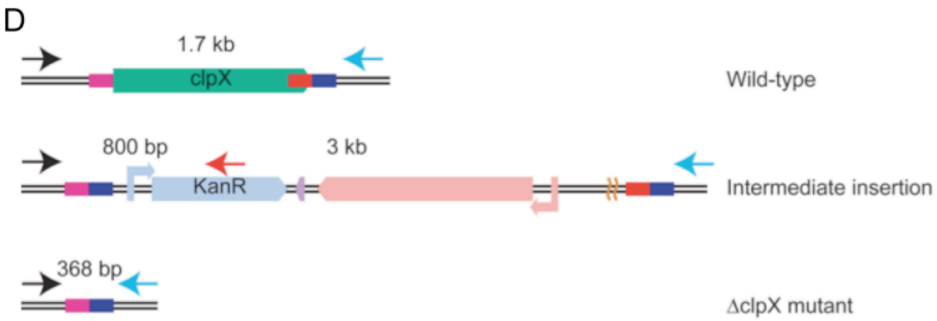


Figure S7. Colony PCR screening. **(A-C)** Colony PCR screening of pNJH00010 transformants of **(A)** CPEC, **(B)** Gibson, and **(C)** SLIC assemblies. (**A-C**) Lane 1 – 1 kb+ DNA ladder "M"; lanes 2 through 9 – amplicons from colonies "1" through "8" (respectively). (**C**) Lane 11 – plasmid DNA positive control "C". **(D)** Agarose gel electrophoresis of colony PCRs of combinatorial Golden Gate assembly transformants. Four colonies were screened for each transformation of plasmids pRDR00001 - pRDR00008 (indicated as -01, -02, -03, … -08). Lane 1 – 1 kb+ DNA ladder "M".


C. ecRDR10003
5'
3'
3'
5'
ΔclpX genomic locus
Markerless knockout
I-SceI mediated double-stranded break. Homologous recombinaton between flanking H3 motifs.
B. ecRDR10002
Intermediate genomic insertion
A. ecRDR10001
H1
clpX
H2H3
Wild-type genomic locus
λ-red mediated homolgous recombination between flanking H1 and H2 motifs.
T0
KanR
SacB
I-SceI
~3 kb linear KO cassette
PCR amplification using pSKI primers with flanking genomic homology sequences
pSKI
Template plasmid pSKI


D
1.7 kb
clpX
Wild-type
800 bp
KanR
3 kb
Intermediate insertion
368 bp
ΔclpX mutant

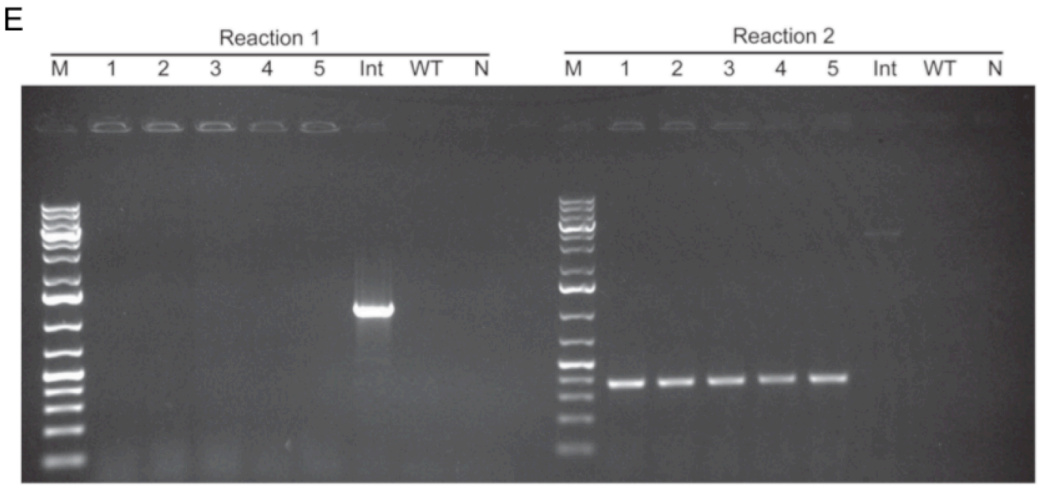

E
Reaction 1
M 1 2 3 4 5 Int WT N
Reaction 2
M 1 2 3 4 5 Int WT N

Figure S8. *clpX* deletion. **(A-B)** *clpX* deletion cassette design. **(A)** Schematic of the linear *clpX* deletion cassette (JPUB_000255) assembly task. The deletion cassette region from plasmid pSKI (*7*), spanning from the promoter region upstream of *kanR* through the I-SceI homing-endonuclease recognition sequence is PCR amplified, with the forward primer introducing a sequence homologous to the *E. coli* genome immediately upstream of the *clpX* coding sequence (H1) and a sequence homologous to the genome immediately downstream of the *clpX* coding sequence (H3), and the reverse primer introducing a sequence homologous to a portion of *clpX* coding sequence (H2). **(B)** Schematic of the marker-less deletion of the genomic copy of *clpX* utilizing the linear deletion cassette. The deletion cassette depicted in (A) is transformed into *E. coli* (strain ecRDR10001/JBEI-2948) expressing the λ-red recombinase system from plasmid pREDI (*7*). Following λ -red mediated double-homologous recombination (at the H1 and H2 loci), replacing *clpX* in situ with the deletion cassette, transformant colonies are selected from kanamycin agar plates (strain ecRDR10002/JBEI-3080). Following the expression of the I-SceI homing-endonuclease from the pREDI plasmid, double stranded break at the I-SceI recognition site within the deletion cassette, and homologous recombination at the H3 locus, colonies are selected from sucrose (*sacB* counter-selection) agar plates, and counter-screened for kanamycin sensitivity, indicating the markerless deletion of *clpX* (strain ecRDR10003/JBEI-3083). **(D-E)** Colony PCR *clpX* protease deletion validations. **(D)** Schematic of diagnostic colony PCR reactions. Reaction 1: forward primer (black arrows) anneals to sequence flanking the 5' end of the *clpX* coding sequence (CDS), reverse primer (red arrow) anneals within the *kanR* CDS. Reaction 1 should result in an 800 bp product for the *clpX* deletion cassette integration intermediate, but in no product for Δ*clpX* mutant nor wildtype. Reaction 2: forward primer (black arrows) anneals to sequence flanking the 5' end of the *clpX* CDS, reverse primer (blue arrows) anneals to sequence flanking the 3' end of the *clpX* CDS. Reaction 2 should result in a 368 bp product for a Δ*clpX* mutant, a 3 kb product for the *clpX* deletion cassette integration intermediate, or a 1.7 kb product for WT. **(E)** Colony PCR validations of *clpX* markerless deletion (JBEI-3083). For each reaction 1 and reaction 2: Lane 1 – 1kb DNA ladder "M", lanes 2 through 6 – Δ*clpX* mutants 1 through 5 (respectively), lane 7 – *clpX* deletion cassette integration intermediate (strain JBEI-3080), lane 8 – WT control (JBEI-2948), lane 9 – no DNA template control. All bands were observed at the expected size. In reaction 2, the integration intermediate band is faint but present, while the expected wildtype band was not detected.

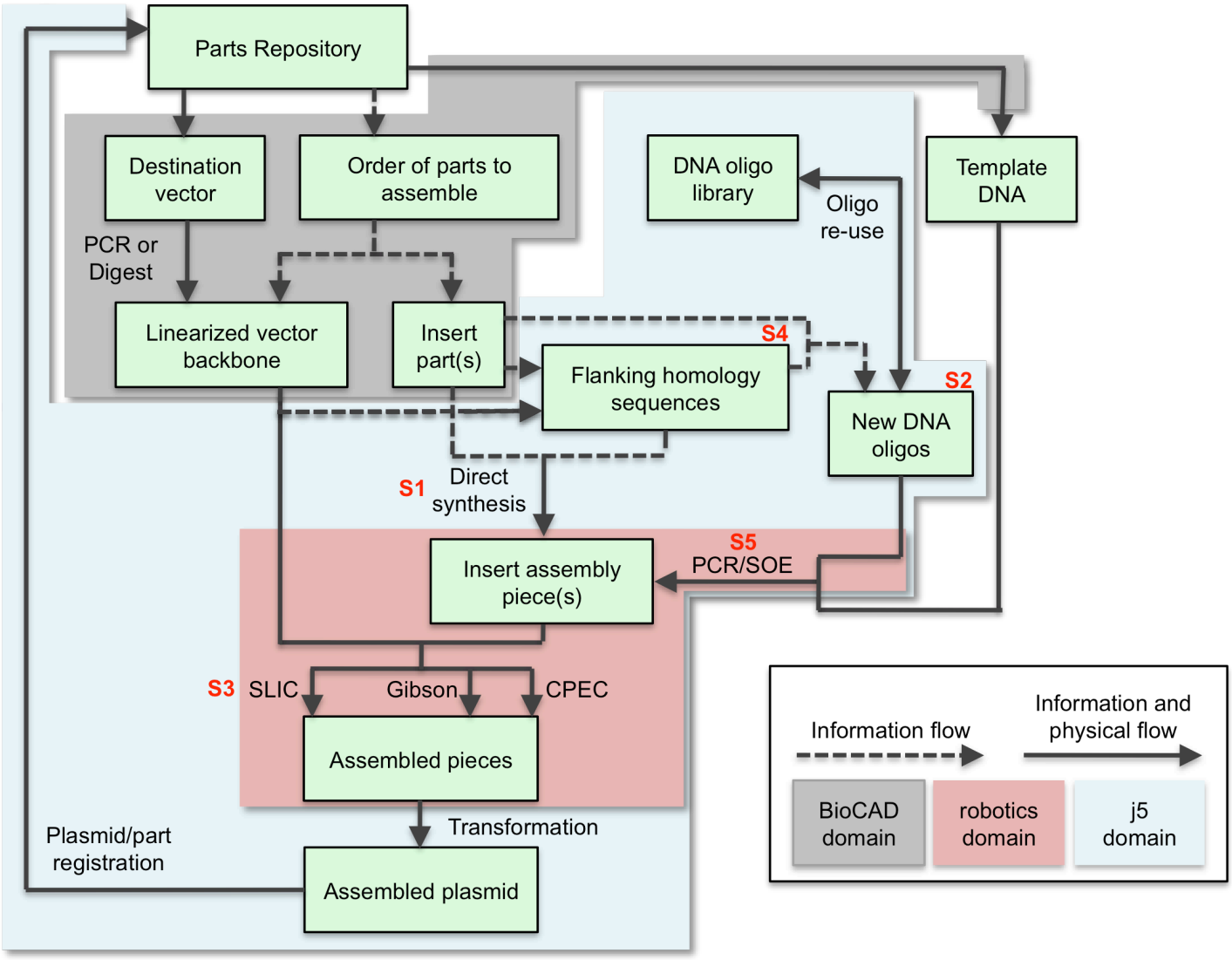


Figure S9. Detailed workflow for SLIC/Gibson/CPEC assembly. A researcher begins the DNA assembly process by selecting parts to assemble from a registry of biological parts (such as the JBEI-ICE repository) or a local collection of DNA sequences; biological computer-aided design (BioCAD) tools may assist this process. The parts to assemble are categorized into either the linearized destination vector, or insert parts. The linearized destination vector is physically achieved by digesting the destination vector with restriction enzymes (as in Figure 2) or by PCR-amplifying the vector backbone (as in Figure 3). Given the sequences of the linearized destination vector and the insert parts, j5 designs flanking homology sequences for each assembly piece, and performs an analysis to determine for which (if any) portions of the assembly direct synthesis would be more cost-effective than either PCR/SOE or oligo embedding. j5 then designs DNA oligos for synthesis, and/or suggests re-use of existing oligos where possible, to amplify the desired assembly pieces. The parts to be assembled do not need to physically exist before using j5 to design the assembly, since it is possible to specify a direct synthesis strategy for any assembly fragment. Liquid handling robotics or other devices may assist the execution of PCR/SOE to generate the assembly pieces, as well as their subsequent SLIC/Gibson/CPEC assembly. j5 facilitates this process by condensing/aggregating designs for multiple independent assemblies into 96-well plate format including optimally distributing reactions across a thermo-cycler annealing temperature gradient (as in Figure S5E). After transforming a competent cloning strain with the assembly reaction, a clonal isolate of the assembled plasmid is sequence verified, assayed for function as desired, and then deposited into the parts registry or local collection for subsequent re-use. The Golden Gate process is analogous that shown, with the design of overhang sequences substituting for the design of flanking homology sequences. S1-S5 (bold red lettering) refer to locations within the workflow where Algorithms S1-S5 are utilized. Algorithm S1 determines when DNA synthesis is cost-effective, S2 assists the design of new DNA oligos, S3 determines if a hierarchical assembly strategy is required, S4 optimizes the design of Golden Gate overhang sequences, and S5 optimizes the placement of PCR reactions across thermocycler gradients.

ALGORITHMS

Algorithm S1: After the user has selected an assembly methodology (SLIC/Gibson/CPEC or Golden Gate), heuristically determine the most cost-effective strategy to incorporate each part into an assembly fragment prior to executing the full assembly design process.

```
1:  for all part ∈ part_list do
2:      if not defined part.strategy then
3:          if part.length < minimum_PCR_length then
4:              if CanEmbedInPrimer(part) then
5:                  part.strategy ← embed_in_primer
6:              else
7:                  part.strategy ← synthesis
8:              end if
9:          else
10:             part.strategy ← PCR
11:         end if
12:     end if
13: end for
14: for all part ∈ part_list do
15:     if part.strategy ≠ synthesis then
16:             if MarginalPCRCost(part) > SynthesisCost(part) then
17:             part.strategy ← synthesis
18:         end if
19:     end if
20: end for
21: for all part ∈ part_list do
22:     if part.strategy = synthesis then
23:         if part.next.strategy ≠ synthesis then
24:             if MarginalPCRCost(part.next) > MarginalSynthesisCost(part.next) then
25:                 part.next.strategy ← synthesis
26:             end if
27:         end if
28:     end if
29: end for
30: for all part ∈ part_list do
31:     if part.strategy = synthesis then
32:         if part.previous.strategy ≠ synthesis then
33:             if MarginalPCRCost(part.previous) > MarginalSynthesisCost(part.previous) then
34:                 part.previous.strategy ← synthesis
35:             end if
36:         end if
37:     end if
38: end for
```

where MarginalPCRCost() returns the (in context) marginal cost of adding the part to its designated PCR reaction;

where MarginalSynthesisCost() returns the (in context) marginal cost of adding the part to the adjacent direct synthesis fragment.

Direct synthesis orders often have minimum charges per synthesized sequence (e.g. $0.39/bp and a $159 minimum per sequence), so the marginal cost of adding a part to an adjacent direct synthesis fragment might be significantly less expensive than directly synthesizing the part by itself (since an additional minimum charge will not be incurred). This is the justification for the third step of Algorithm S1. Algorithm S1 assumes that the

most likely primer lengths, and flanking sequence lengths (SLIC, Gibson or CPEC) or relative overhang positions (Golden Gate) will be used throughout the assembly process. Since primer, flanking sequence, and relative overhang positions are optimized during the design process and thereby differ from the most likely values, this heuristic may fail at non-continuities in the MarginalPCRCost() function. For example, extending the length of a primer from 60 to 61 bp may result in an abrupt additional DNA oligo PAGE-purification cost which may be as much as an additional $60/primer. For this reason, further development will focus on integrating the determination of the most cost-effective assembly strategy into the full design process.

Algorithm S1 is currently utilized after the user has already selected an assembly methodology. Further development of Algorithm S1 could assist the user in deciding which assembly method to select by comparing the cost and time requirements for the various assembly methods. This could include a refined distinction between SLIC, Gibson, and CPEC assembly from a method cost-perspective, associating differential anticipated failure rate risks as costs embodied in extra time, labor and DNA sequencing requirements. Furthermore, Algorithm S1 provides a reasonable heuristic for determining the most cost-effective assembly strategy for a single construct, but does not properly account for part re-use across a combinatorial library. For example, it may be less expensive to directly synthesize two parts in a single contiguous fragment (due to a minimum per sequence charge as described above). However, if each of the two parts can be repeatedly re-used across a combinatorial library, but the concatenation of the two parts is only used in one of the combinations, synthesizing the two parts separately can be effectively amortized over multiple combinations and provide the most cost-effective strategy. Further development will target combinatorial amortization accounting. In the meantime, a manual software control mechanism (direct synthesis firewalling) is in place that allows the user to prevent directly synthesizing adjacent parts together across combinatorial assembly junctions. Algorithm S1 does not account for the costs of enzymatic reagents, competent cells, sequencing reactions, nor labor charges (which may dominate in industry). Further development will target a more sophisticated cost function that includes these factors. Finally, j5 outputs only the Algorithm S1-calculated cost-optimal strategy, but could be further developed to provide a set of comparable alternatives when the difference in cost falls within a user-specifiable threshold.

Algorithm S2: Progressively relieve violated constraints during primer (or flanking sequence) design.

Existing programs such as Primer3 (*11*) can be successfully leveraged to optimize the design of primers or flanking homology sequences (effectively primers for adjacent assembly pieces during Gibson and CPEC assembly). One drawback to these existing software packages is that they provide primer pair designs only if a given set of design criteria is met. For example, if all considered primers fall outside of a desired melting temperature range, an error message is issued, but no primer designs are returned. While it may be possible to force the software to design at least one (if sub-optimal) primer pair per desired PCR reaction, this may result in many undesirable design constraint violations, even if primer pairs with fewer constraint violations (but perhaps with lower overall design scores, constraint violations aside) are accessible. Algorithm S2 first attempts to design optimal primers that meet all design constraints; if unable to do so, constraints are progressively relieved until an acceptable primer pair has been achieved. In addition to the primers (or flanking homology sequences) designed, warning messages are issued if any design constraints were violated/relieved during the design process and/or if any putative template mis-priming events with above threshold melting temperatures are identified via BLAST (*12*).

```
1:  constraints ← target
2:  repeat
3:      primers ← DesignPrimers(constraints)
4:      constraints.gc_clamp ← constraints.gc_clamp - 1
5:  until defined primers or constraints.gc_clamp < 0
6:  constraints.gc_clamp ← constraints.gc_clamp + 1
7:  if not defined primers then
8:      repeat
9:          EliminateFirstViolatedConstraint(constraints)
10:         primers ← DesignPrimers(constraints)
11:     until defined primers
12: end if
13: while defined primers and constraints.gc_clamp < target.gc_clamp do
```

```
14:     constraints.gc_clamp ← constraints.gc_clamp + 1
15:     primers ← DesignPrimers(constraints)
16: end while
17: if not defined primers then
18:     constraints.gc_clamp ← constraints.gc_clamp – 1
19:     primers ← DesignPrimers(constraints)
20: end if
```

where DesignPrimers() returns the optimal primer pair if the design constraints can be met;

where EliminateFirstViolatedConstraint() identifies (via a rank-ordered triage process) the next violated constraint to relieve; the constraint rank-ordering (first eliminated to last) is as follows: too many Ns, too many poly-X, GC content, minimum Tm, maximum Tm, maximum difference in Tm, self-complementarity, and pair-complementarity.

For the SLIC/Gibson/CPEC design shown in Figure 2, of the 6 primers (required for the 3 PCR reactions) and the 6 PCR-derived assembly junction termini, only the 4 primers for PCR reactions “1” and “3” could be successfully designed by Primer3 without Algorithm S2 constraint relief. For this design, the particular rank-ordering of constraint relief had no impact on the total number or type of constraints relieved. For the combinatorial Golden Gate design shown in Figure 3, of the 12 primers required for the 6 PCR reactions, the 8 primers for PCR reactions “1”, “2”, “3”, and “4” could be successfully designed by Primer3 without Algorithm S2 constraint relief. Here too, the particular rank-ordering of constraint relief had no impact on the total number or type of constraints relieved. For other designs, the particular rank-ordering of constraint relief may have a more significant impact. Algorithm S2’s constraint rank-ordering is currently subjective. Over time, given an accumulated data set of PCR successes and failures, it would be possible to objectively analyze the relationship between relaxed constraint type and PCR or SLIC/Gibson/CPEC assembly failure rate.

It should be pointed out that (at least for Primer3), GC clamp length is associated only with a constraint, unlike primer melting temperature, for example, for which there are constraints (e.g. maximum and minimum acceptable temperature) in addition to a scoring function (distance from the target melting temperature) that rank-orders multiple putative primers that fall within constraint tolerances. As a consequence, no GC clamp is considered equivalent to a one or two-bp GC clamp if they are all shorter than the design constraint. For this reason, Algorithm S2 treats the GC clamp separately from all other constraints that have associated scoring functions; other constraint-only parameters could be similarly treated.

Algorithm S3: Identify SLIC/Gibson/CPEC assembly piece incompatibilities; if found, design a hierarchical assembly strategy

The SLIC (*3*), Gibson (*2*), and CPEC (*4*) assembly methodologies utilize sequence homology at assembly piece termini to direct the assembly process. If two or more assembly pieces have sufficiently identical sequence at their respective termini (e.g. fragments “2” and “5” in Figure S2A), there is an ambiguity in the assembly process, which can lead to undesirable products (e.g. pieces assembled in the incorrect order or sections missing altogether). These assembly pieces are said to be incompatible with one another, since placing them into the same assembly reaction can lead to undesired products. For the CPEC method in particular, and potentially for the Gibson method, there is an additional concern that the terminus of an assembly piece will mis-prime an internal portion of itself or another assembly piece (e.g. the 3’ end of fragment “4” could mis-prime the P_{lac} subsequence in fragment “0” in Figure S2A), which can also lead to undesired assembly products. Algorithm S3 first identifies any putative assembly piece incompatibilities, and then attempts to design a hierarchical assembly strategy that mitigates the risk of incorrect assembly products. If no such hierarchical assembly strategy is possible, a warning message is issued.

```
1:  for all start_piece ∈ piece_list do
2:      contig ← new Contig
3:      piece ← start_piece
4:      while piece.next ≠ start_piece and Compatible(contig, piece.next)
5:          push contig piece.next
6:          piece ← piece.next
```

```
7:      end while
8:      push contig_list contig
9:  end for
10: EliminateEmptyOrSubsetContigs(contig_list)
11:     for all contig ∈ contig_list do
12:         for all piece ∈ contig do
13:         unique ← true
14:             for all other_contig ∈ contig_list and contig ≠ other_contig do
15:             if Contains(other_contig, piece) then
16:                 unique ← false
17:                 last
18:             end if
19:         end for
20:         if unique then
21:             for all other_contig ∈ contig_list and contig ≠ other_contig do
22:                 for all other_piece ∈ contig do
23:                     Remove(other_contig, other_piece)
24:                 end for
25:             end for
26:         end if
27:     end for
28: end for
29: EliminateEmptyOrSubsetContigs(contig_list)
30:     for all contig ∈ contig_list do
31:         for all piece ∈ contig do
32:         for all other_contig ∈ contig_list and contig ≠ other_contig do
33:             Remove(other_contig, piece)
34:         end for
35:     end for
36: end for
37: EliminateEmptyOrSubsetContigs(contigs_list)
38: failure ← false
39:     for all contig ∈ contig_list do
40:     compatible ← false
41:     while not failure and not compatible do
42:         for all other_contig ∈ contig_list and contig ≠ other_contig do
43:             if not 3'Compatible(contig, other_contig) then
44:                 if not Move3'Piece(contig, contig.next) then
45:                     failure ← true
46:                 else
47:                     contig.next.5'adjusted  ← true
48:                 end if
49:                 last
50:             end if
51:         end for
52:     end while
53:     compatible ← false
54:     while not failure and not compatible do
55:         for all other_contig ∈ contig_list and contig ≠ other_contig do
56:             if not 5'Compatible(contig, other_contig) then
57:                 if contig.5'adjusted or not Move5'Piece(contig, contig.previous) then
58:                     failure ← true
59:                 end if
```

```
                last
            end if
        end for
    end while
end for
if length contig_list > 1
    hierarchical ← true
else
    hierarchical ← false
    end if
```

where Compatible() returns true if the passed assembly piece is compatible with all of the pieces in the passed contig; otherwise returns false;

where 3'Compatible() returns true if the 3' terminus of the first passed contig is compatible with the second passed contig; otherwise returns false;

where 5'Compatible() returns true if the 5' terminus of the first passed contig is compatible with the second passed contig; otherwise returns false;

where Move3'Piece() returns true if the 3' assembly piece of the first passed contig is compatible with each piece contained within the second passed contig. If so, moves the 3' assembly piece of the first passed contig to the 5' end of the second passed contig; otherwise returns false;

where Move5'Piece() returns true if the 5' assembly piece of the first passed contig is compatible with each piece contained within the second passed contig. If so, moves the 5' assembly piece of the first passed contig to the 3' end of the second passed contig; otherwise returns false;

If a hierarchical assembly strategy cannot be found to mitigate the identified assembly piece incompatibilities, it is likely that a manual user adjustment (such as breaking a part into two sub-parts) will be required to design a successful assembly. For example, consider a variation of the assembly task shown in Figure S2A in which fragments "3" and "4" are a single contiguous assembly piece. The 5' end of this contiguous piece would be incompatible with the immediately downstream fragment "5", and the 3' end would be incompatible with the immediately upstream fragment "2". These incompatibilities are not able to be resolved using a hierarchical assembly strategy. However, as shown in Figure S2A, splitting this contiguous assembly piece into separate fragments "3" and "4", it is possible to identify a workable hierarchical assembly strategy. Further development will target the identification of such assembly piece splitting resolutions to incompatibilities that cannot be hierarchically resolved. It should be pointed out that Algorithm S3 is also directly applicable to the in vivo yeast method DNA assembler (*13*), which also uses sequence homology to direct the assembly process.

While the case for a hierarchical assembly mitigation strategy is clear for the example shown in Figure S2A with two sequence-identical assembly junctions ("1" to "2", and "5" to "6"), the inverse relationship between assembly junction similarity and assembly efficiency has yet to be quantitatively explored. A reasonable way to approach this would be to capture the assembly efficiency (i.e., success rate) of each reaction as an integral part of the workflow depicted in Figure S9. This large accumulated meta-data set could then be continually analyzed towards a refined quantitative relationship between assembly efficiency and junction similarity, which would inform the cost-benefit calculus for one-pot vs. hierarchical assembly strategies.

Algorithm S4: Search for the optimal set of Golden Gate assembly piece overhangs

The Golden Gate assembly method (*6*) utilizes 4-bp 5' overhang sequences to direct the assembly process. If two or more overhang sequences are sufficiently cohesive to a cognate overhang (e.g. the blue and grey bottom overhangs are both cohesive to the blue top overhang shown in Figure S2D), there is an ambiguity in the assembly process, which can lead to undesirable products (e.g. pieces assembled in the incorrect order or sections missing altogether). These overhang sequences are thus said to be incompatible with one another. Algorithm S4 first identifies putative overhang sequence regions (constrained by the maximum oligo size and the extents of combinatorial sequence identity about each assembly junction) and then searches these regions

for the set of overhang sequences that are compatible with themselves and each other, and that are as neutral as possible (see Figure S2E). If no set of compatible Golden Gate overhangs is found, an error message is issued.

```
1:  for all junction ∈ junction_list do
2:      GenerateOverhangList(junction)
4:      sort junction.full_overhang_list by increasing Position()
5:      for all overhang ∈ junction.full_overhang_list do
6:          if not Compatible(overhang)
7:              Remove(overhang)
8:          else
9:              for all prior_overhang ∈ junction.full_overhang_list before overhang do
10:                 if prior_overhang = overhang
11:                     Remove(overhang)
12:                     last
13:                 end if
14:             end for
15:         end if
16:     end for
17: end for
18: undefine stable
19: current_junction ← First(junction_list)
20: current_junction.overhang_list ← junction.full_overhang_list
21: resume ← false
22: while true do
23:     if not FindCompatibleOverhangs(junction_list, stable, current_junction, resume)
24:         last
25:     end if
26:     resume ← true
27:     if not defined best or MaxPosition(junction_list) < max
28:         best ← junction_list
29:         max ← MaxPosition(junction_list)
30:             for all junction ∈ junction_list do
31:                 for all overhang ∈ junction.full_overhang_list do
32:                 if Position(overhang) > max then
33:                     Remove(overhang)
34:                 end if
35:             end for
36:             for all prior ∈ junction_list before junction do
37:                 for all overhang ∈ junction.prior.full_overhang_list do
38:                     if Position(overhang) > max then
39:                         Remove(overhang)
40:                     end if
41:                 end for
42:             end for
43:         end for
44:     end if
45: end while

46: procedure FindCompatibleOverhangs(junction_list, stable, junction, resume)
47:     while true do
48:         for all prior ∈ junction_list after stable before junction do
49:             if prior = First(junction_list) then
```

```
50:                 junction.prior.overhang_list ← junction.full_overhang_list
51:             else
52:                 junction.prior.overhang_list ← junction.Previous(prior).overhang_list
53:             end if
54:                 for all overhang ∈ junction.prior.overhang_list do
55:                 if not Compatible(prior.current_overhang, overhang)
56:                     Remove(overhang)
57:                 end if
58:             end for
59:         if junction = Last(junction_list) and resume
60:             Remove(First(junction.Previous(junction).overhang_list))
61:         end if
62:         if junction = First(junction_list) then
63:             junction.current_overhang ← First(junction.overhang_list)
64:         else
65:             junction.current_overhang ← First(Previous(junction).overhang_list)
66:         end if
67:         while not defined junction.current_overhang do
68:             if junction = First(junction_list) then
69:                 return false
70:             end if
71:             junction ← Previous(junction)
72:             stable ← Previous(junction)
73:             repeat
74:                 if junction = First(junction_list) then
75:                     Remove(junction.current_overhang)
76:                     junction.current_overhang ← First(junction.overhang_list)
77:                 else
78:                     Remove(junction.Previous(junction).current_overhang)
79:                     junction.current_overhang ← First(junction.Previous(junction).overhang_list)
80:                 end if
81:             until not (defined junction.current_overhang and RedundantSearchPath(junction))
82:         end while
83:         if junction = Last(junction_list) then
84:             return true
85:         else
86:             junction ← Next(junction)
87:     end while
88: end procedure

89: procedure RedundantSearchPath(junction_list, junction)
90:     for all prior ∈ junction_list before junction do
91:         if junction.current_overhang ∈ prior.overhang_list and
92:             prior.current_overhang ∈ junction.overhang_list then
93:             if Max(Position(junction.current_overhang),Position(prior.current_overhang)) >
94:                 Max(Position(junction.overhang_list.(prior.current_overhang)),
95:                     Position(prior.overhang_list.(junction.current_overhang))) then
96:                 return true
97:             else if Max(Position(junction.current_overhang),Position(prior.current_overhang)) =
98:                     Max(Position(junction.overhang_list.(prior.current_overhang)),
99:                             Position(prior.overhang_list.(junction.current_overhang))) then
101:                if junction.current_overhang ∈ junction.overhang_list after
102:                    prior.current_overhang ∈ junction.overhang_list then
103:                    return true
```

```
104:                 end if
105:             end if
106:         end if
107:     end for
108:     return false
109: end procedure
```

where GenerateOverhangList() returns the list of putative 4-bp overhangs that are located within the putative overhang sequence region (see Figure S2E) that spans the assembly junction (constrained by the maximum oligo size and the extents of combinatorial sequence identity about the assembly junction) from which to select a 4-bp overhang;

where Compatible() returns true for a single passed overhang if the overhang is compatible with itself (the maximum number of ungapped aligned identities (all frame shifts, both strands) is below threshold, see Figure S2D); similarly returns true for two passed overhangs if the two overhang sequences are compatible with one another (see Figure S2D); utilizes a hash lookup table to avoid redundant calculations; otherwise returns false;

where Position() returns the relative overhang position in bp from neutral (see Figure S2E);

where MaxPosition() returns the maximum relative overhang position in bp from neutral across all assembly junctions;

If no set of compatible set of Golden Gate overhangs is found, it is likely that a manual user adjustment (such as adding scar sequences at one or more assembly piece junctions) will be required to design a successful assembly. Further development will target the automated design of minimal scar sequences that allow for a compatible set of Golden Gate overhangs to be identified. A scar-less alternative option is to utilize a variant of Algorithm S3 to design a hierarchical Golden Gate assembly, analogous to that shown in Figure S2A-C. Further development will target the automated design of this alternative hierarchical Golden Gate assembly strategy. It should be pointed out that Algorithm S4 is also directly applicable to the USER DNA assembly methodology (*14*), which also uses overhang sequences (although frequently longer than 4-bp) to direct the assembly process. A variant of Algorithm S4 could also be applied to (combinatorial) SLIC, Gibson, CPEC, *in vivo* yeast DNA assembler, or other methods, and would likely be preferable to the utilization of hierarchical assembly processes (depicted in Figure S2A-C and designed by Algorithm S3) wherever possible. Further development will target the application of Algorithm S4 to designing these homology sequence recombination methodologies.

Algorithm S4 utilizes dynamic programming to reduce search complexity. Algorithm S4 stores previous compatible/incompatible overhang sequence calculations in a look-up table (the Compatible() procedure), recursively determines the residual set of overhang sequences to choose from at each junction (see for example pseudo-code line 52), and dynamically avoids redundant search paths (the RedundantSearchPath() procedure). For many simple Golden Gate assembly designs, the complexity of Algorithm S4 may appear to be overkill. However, we have found that as the number of assembly pieces approaches (or narrowly exceeds) ten, and/or if the sequences spanning assembly junctions are highly homologous (e.g. repeated or highly similar RBS sites), the search process needs to be kept as efficient as possible to terminate in a reasonable amount of time. This is because the complexity of the exhaustive search for compatible Golden Gate overhang sequences is roughly $O(M^N)$, where M is the number of overhang sequences to choose from for each junction, and N is the number of junctions. Algorithm S4 is not embarrassingly parallelizable since the optimal search process is dependent on the characteristics of the best compatible overhang set found so far. Nevertheless, it would be possible to parallelize it (without inducing too much waste) by tasking each thread/process with a subset of the overhang possibilities for the first junction(s) and having each thread/process broadcast their best set parameters as they are found. As the price of direct DNA synthesis continues to fall, and replaces the need for embedding sequence resulting from non-neutral overhang position selection into the corresponding primers, there will be less of a premium placed on maximizing the neutrality of the overhang positions, and more of an emphasis on compatibility stringency. This change in emphasis will not require any change to Algorithm S4, but will rather just require a perturbation to the stringency of the Compatible() function and an extension of the putative overhang sequence regions beyond what is currently constrained by the maximum oligo length.

Algorithm S5: Closely approximate the optimal distribution of PCR reactions in multi-well plates across thermocycler block annealing temperature zone gradient(s)

Depending on the design of a given DNA assembly process, PCR may be required to generate (some of) the assembly pieces. While primer and flanking homology sequence design attempt to constrain melting temperature to a narrow acceptable range where possible (see Algorithm S2), extreme %GC template composition may skew the resulting temperatures to well below (AT-rich) or above (GC-rich) the targeted optimum. Most modern thermocyclers feature standardized multi-well format blocks, and some (such as the Applied Biosystems Veriti Thermal Cycler employed in this study) now feature temperature gradients with individually controllable annealing temperature zones. Algorithm S5 takes as input a set of PCR reactions with target annealing temperatures, taken here to be the minimum of the forward and reverse primer melting temperatures + 2 °C, and optimizes the annealing temperature zones of the thermocycler block(s) and the distribution of the PCR reactions (in multi-well plates) across the zones so as to minimize the objective function, namely the summed difference squared between the targeted annealing temperatures of the PCR reactions and the actual annealing temperatures of the thermocycler zones in which they are placed (as shown in Figure S5E). Algorithm S5 exploits a Monte-Carlo simulated annealing approach to converge upon the optimal distribution. Simulated annealing is a classical computational technique to find global minima in discrete search spaces with complicated energy landscapes. This approach is well suited to the optimization problem addressed by Algorithm S5 because the search space (the placement of each PCR reaction in its own well, and the annealing temperature of each zone) is discrete, and there is a complicated relationship between zone temperatures, PCR reaction placements, and the objective function to be minimized.

```
number_blocks = MinBlocksRequired(reaction_list) - 1
repeat
    number_blocks ← number_blocks + 1
    block_list ← InitializeBlocks(number_blocks, reaction_list)
    FillBlocks(block_list, reaction_list)
    current ← Objective(block_list, reaction_list)
    best ← current
    current_temperature ← initial
    for all move ← 1, n  do
        trial_list ← block_list
        TrialMove(trial_list)
        FillBlocks(trial_list, reaction_list)
        trial ← Objective(trial_list, reaction_list)
        if trial < current or Random() < Exp((current – trial)/current_temperature) then
            block_list ← trial_list
            current ← trial
            if current ≤ best then
                best ← current
                best_block_list ← block_list
            end if
        end if
        current_temperature ← current_temperature – (initial – final)/n
    end for
until MaxDeviance(best_block_list, reaction_list) < threshold
```

where MinBlocksRequired() returns the minimum number of thermocycler blocks required to contain all of the PCR reactions;

where InitializeBlocks() returns a set of the specified number of thermocycler blocks whose zone annealing temperatures have been initialized to span from the lowest optimal annealing temperature across the PCR reactions to the highest optimal annealing temperature across the PCR reactions (or highest temperature that can be achieved given temperature gradient limitations) with linear step annealing temperature increases between zones;

where FillBlocks() fills the thermocycler blocks with the PCR reactions; repeats the following procedure for each PCR reaction sorted from lowest to highest optimal annealing temperature: given the zone annealing temperatures, identify the best zone with an empty well remaining to which to add the current PCR reaction, and deposit the PCR reaction in this zone; after depositing all of the PCR reactions into the thermocycler block(s), rearrange the PCR reactions in place (same thermocycler wells) such that the annealing temperatures of the PCR reactions are sorted monotonically from low to high with the increasing zone annealing temperature gradient;

where Objective() returns the sum of the difference squared between the optimal annealing temperature of each PCR reaction and the actual annealing temperature of the zone it has been placed in;

where TrialMove() randomly select one of the zones within the specified thermocycler blocks, and randomly perturbs the annealing temperature of the zone by either adding or subtracting a delta temperature; if this perturbation collaterally affects adjacent zones (due to temperature gradient limitations) adjust the temperatures of the affected zones accordingly;

where Random() returns a number from the half-closed interval [0,1) with uniform probability;

where MaxDeviance() returns the maximum temperature deviance between the optimal annealing temperature of a PCR reaction and the actual annealing temperature of the zone it has been placed in.

Depending on the parameters selected and search scheme adopted, simulated annealing can act as a random search, prematurely converge on local minima, or converge on the desired global minimum. It is crucial to explore the search space sufficiently well so as to ensure confidence that the global minimum has been encountered, but an excessive number of trial moves is computationally wasteful. Some of the parameters (e.g., MAXDELTATEMPERATUREADJACENTZONES, NCOLUMNSMULTIWELLPLATE, NROWSMULTIWELLPLATE, WELLSPERTHERMOCYCLERZONE, ZONESPERTHERMOCYCLERBLOCK, and TRIALDELTATEMPERATURE) governing Algorithm S5 are determined by thermocycler specifications and multi-well plate format geometry. The MAXDELTATEMPERATUREREACTIONOPTIMUMZONEACCEPTABLE parameter is determined by the experimental preference of the user. Two parameters in particular (MAXMCSTEPSPERZONE and MCTEMPERATUREINITIAL) determine whether Algorithm S5 acts as a random search, or converges on local or global minima. Figure S6 shows Algorithm S5 convergence as a function of MAXMCSTEPSPERZONE and MCTEMPERATUREINITIAL. The default parameters for j5 (MCTEMPERATUREINITIAL = 0.1 and MAXMCSTEPSPERZONE = 1000) are set conservatively so as remain putatively effective for more frustrated searches than that pursued in Figure S6.

For simple DNA assembly designs that do not require too many PCR reactions, Algorithm S5 may seem excessive. In addition, anecdotal experience may suggest that precisely tuning the annealing temperature for a given PCR reaction might not yield significantly superior PCR results, since the optimal annealing temperature range may be fairly broad (spanning several °C) for any given PCR reaction. While these points are well taken, it should be pointed out that multiple small assembly tasks can be condensed into a sizable meta-assembly project (see Results) with many collective prerequisite PCR reactions, and furthermore, there is no compelling reason not to exploit available thermocycler gradient features if the design process is automated and effectively effortless. Sets of PCR reactions with non-uniformly distributed target annealing temperatures with extreme highs and lows will be the most likely to derive benefit from Algorithm S5.

Algorithm S5 would need to be adjusted for a strictly linear (non-zone type) gradient thermocycler (such as a MJ Research Tetrad PTC-225 Thermo Cycler). This could be accomplished by modifying the subroutine that generates the initial distribution of zone temperatures, and changing the Monte Carlo move set such that either of the linear gradient temperature extremes may be perturbed, and internal intermediate zones are linearly adjusted accordingly. Further development will focus on an implementation variant of Algorithm S5 for strictly linear thermocycler gradient blocks.

REFERENCES

1. Khlebnikov, A., Datsenko, K. A., Skaug, T., Wanner, B. L., and Keasling, J. D. (2001) Homogeneous expression of the P(BAD) promoter in Escherichia coli by constitutive expression of the low-affinity high-capacity AraE transporter, *Microbiology 147*, 3241-3247.
2. Gibson, D. G., Young, L., Chuang, R. Y., Venter, J. C., Hutchison, C. A., 3rd, and Smith, H. O. (2009) Enzymatic assembly of DNA molecules up to several hundred kilobases, *Nat Methods 6*, 343-345.
3. Li, M. Z., and Elledge, S. J. (2007) Harnessing homologous recombination in vitro to generate recombinant DNA via SLIC, *Nat Methods 4*, 251-256.
4. Quan, J., and Tian, J. (2009) Circular polymerase extension cloning of complex gene libraries and pathways, *PLoS One 4*, e6441.
5. Engler, C., Gruetzner, R., Kandzia, R., and Marillonnet, S. (2009) Golden gate shuffling: a one-pot DNA shuffling method based on type IIs restriction enzymes, *PLoS One 4*, e5553.
6. Engler, C., Kandzia, R., and Marillonnet, S. (2008) A one pot, one step, precision cloning method with high throughput capability, *PLoS One 3*, e3647.
7. Yu, B. J., Kang, K. H., Lee, J. H., Sung, B. H., Kim, M. S., and Kim, S. C. (2008) Rapid and efficient construction of markerless deletions in the Escherichia coli genome, *Nucleic Acids Res 36*, e84.
8. Musso, M., Bocciardi, R., Parodi, S., Ravazzolo, R., and Ceccherini, I. (2006) Betaine, dimethyl sulfoxide, and 7-deaza-dGTP, a powerful mixture for amplification of GC-rich DNA sequences, *J Mol Diagn 8*, 544-550.
9. Bilitchenko, L., Liu, A., Cheung, S., Weeding, E., Xia, B., Leguia, M., Anderson, J. C., and Densmore, D. (2011) Eugene--a domain specific language for specifying and constraining synthetic biological parts, devices, and systems, *PLoS One 6*, e18882.
10. Bilitchenko, L., Liu, A., and Densmore, D. (2011) The Eugene language for synthetic biology, *Methods Enzymol 498*, 153-172.
11. Rozen, S., and Skaletsky, H. (2000) Primer3 on the WWW for general users and for biologist programmers, *Methods Mol Biol 132*, 365-386.
12. Zhang, Z., Schwartz, S., Wagner, L., and Miller, W. (2000) A greedy algorithm for aligning DNA sequences, *J Comput Biol 7*, 203-214.
13. Shao, Z., and Zhao, H. (2009) DNA assembler, an in vivo genetic method for rapid construction of biochemical pathways, *Nucleic Acids Res 37*, e16.
14. Bitinaite, J., Rubino, M., Varma, K. H., Schildkraut, I., Vaisvila, R., and Vaiskunaite, R. (2007) USER friendly DNA engineering and cloning method by uracil excision, *Nucleic Acids Res 35*, 1992-2002.