Article

Efficient Split Synthesis for Targeted Libraries

Barry Cohen, and Steven Skiena

J. Comb. Chem., **2000**, 2 (1), 10-18• DOI: 10.1021/cc990028a • Publication Date (Web): 09 December 1999

Articles

Efficient Split Synthesis for Targeted Libraries

Barry Cohen and Steven Skiena*

Department of Computer Science, State University of New York at Stony Brook, Stony Brook, New York 11794-4400

Received June 3, 1999

We propose a new approach for fabricating more sophisticated combinatorial chemistry libraries via split synthesis and evaluate its potential through extensive simulation. Our algorithmically intensive method promises to reduce the time and materials costs of synthesizing libraries which are (1) too large to synthesize economically by sequential or parallel synthesis, (2) too long or irregular for conventional split synthesis generation techniques, and (3) not used in sufficient quantity to justify the setup costs of array makers. It also encourages the design of more focused and interesting libraries than are typically constructed using split synthesis. Our algorithms automate the design of efficient synthesis procedures for motif-based libraries which are too complex to design by hand. Our software allows the user to select the most desirable tradeoff between minimizing the number of steps in the synthesis process and containing the combinatorial explosion of the number of compounds synthesized.

1. Introduction

The foundation of combinatorial chemistry is the systematic synthesis and screening of large libraries of small molecules. In the standard one-bead, one-compound approach[1] to combinatorial chemistry, a large number of distinct small molecules are fabricated on resin beads. These beads are reacted against a target to establish which beads display affinities suggesting biologically active agents. The compound associated with a positively reacting bead can then be identified via sequencing or indirect coding methods.

The adjective "combinatorial" in "combinatorial chemistry" refers to the large number of distinct molecules which may be synthesized by split and combine (split synthesis) techniques. Unfortunately, the combinatorial explosion ensures that this number quickly exceeds the number of molecules (on the order of 10^6 to 10^7) which can simultaneously be assayed by high-throughput screening. For example, full libraries of all *l*-peptides cannot be assayed for $l \geq 6$, since there are 20^l distinct molecules containing *l* amino acid residues.

A primary combinatorial constraint on library size is the number of resin beads which can be reasonably employed in the synthesis procedure. While synthesis procedures for more compounds than beads (such as all $20^5 = 3\,200\,000$ pentapeptides) can be performed, clearly most of these compounds will not actually be constructed. Indeed, after accounting for the beads which are wasted by constructing duplicates of some compounds, we would expect to synthesize a random subset of the target library. Even if 3 200 000 beads are employed, the expected yield is only $(1 - (1 - (1/20^5))^{3200000}) = 63\%$ of the target library. The size of the sample required to achieve a desired degree of confidence that the error is held within a specified limit is examined in ref 2.

In response to such constraints, techniques for constructing representatives for all of a simple set of motifs have been developed using random synthesis steps.[1] However, the sophistication and hence effectiveness of such libraries is limited by the difficulty of designing cost-effective synthesis strategies.

Hence the challenge remains to design cost-effective synthesis methods for complex motifs.

In this paper, we propose a new algorithmicly intensive *optimized split synthesis* technology for fabricating combinatorial chemistry libraries and evaluate its potential through extensive simulation. We believe our methods hold promise in synthesizing libraries which are (1) too large to reasonably employ sequential or parallel synthesizers, (2) too long or irregular for conventional split synthesis generation techniques,[3,4] and (3) not used in production quantities, thus making array-based synthesis techniques[5] economically unattractive.

Our techniques enable an experimenter to specify a selective library of compounds with desirable characteristics. The aim may be a library with maximum diversity, or substantial similarity to promising leads, or some combination of these and other criteria. Our algorithms produce an efficient procedure for the synthesis of an arbitrary target library, along with a limited number of related compounds.

* To whom correspondence should be addressed. E-mail: skiena@cs.sunysb.edu.

10.1021/cc990028a CCC: $19.00
Published on Web 12/09/1999

These additional compounds may themselves provide a useful sample or generalization of the space being explored. The experimenter can specify an upper limit on how many such related compounds are acceptable.

The following scenarios illustrate advantages of the libraries we propose.

(i) Increased Specificity. Suppose that an experimenter has identified (based on either preliminary screening or theoretical grounds) a particularly interesting set of 500 hexapeptides which follow no simple motif. For example, such a library might be the set of all subsequences of length six of a particular 500-peptide protein. The standard split synthesis protocol to construct all $20^6 = 64\,000\,000$ possible hexapeptides entails $20 \times 6 = 120$ base-extension or "grow" steps. Such a protocol performed on an initial set of 1 000 000 beads could be expected to synthesize only 1.5% or roughly 8 of the 500 target compounds. Instead, our algorithms would design a synthesis procedure which, with high probability, produces *all* 500 compounds in the target library, together with 20 000 related compounds. The cost of achieving this is a minor increase in the number of base-extension steps, from 120 to 324. Parallel synthesis, by comparison, requires 3000 grow steps to produce just the 500 compounds.

(ii) Increased Generality. After on-bead screening of a large combinatorial library, H. Peter Nestler of Cold Spring Harbor Laboratory identified a set of 35 branched hexapeptides binding to the carboxy-terminus of the H-Ras protein.[6] To identify more active compounds, it is appropriate to construct a more focused library based on these preliminary results.

This set of 35 hexapeptides (N35) could be synthesized using 52 base-extension steps using classical split synthesis techniques, yielding a library of 151 200 compounds. A determined approach to building exactly the 35 hexapeptides would require $35 \times 6 = 210$ steps. Instead, our algorithms designed a synthesis using 56 base-extension steps, which yielded the target set among only 594 compounds. Such parsimony encouraged us to specify a larger, carefully designed target library of all hexapeptides which differ from one of the 35 compounds in one position, and all possible substitutions of amino acids with similar properties at a maximum of two positions. Our algorithm designed a synthesis procedure yielding all 2138 of these compounds (and 241 636 other hexapeptides) in only 100 base-extension steps.

(iii) Reduced Materials and Screening Costs. As illustrated in these two scenarios, our algorithms provide a synthesis technique which, in return for a modest increase in the synthesis complexity compared to classical split synthesis, drastically reduces the number of compounds which must be screened and also reduces the quantity of materials consumed.

1.1. Split Synthesis

Split synthesis, developed independently by Furka[3] and Lam,[4] is the method of choice to build libraries for combinatorial chemistry. The basis of this technology is that molecules can be grown with one end tethered to a resin bead, while the other end is extended one residue at time in parallel across a set of beads.

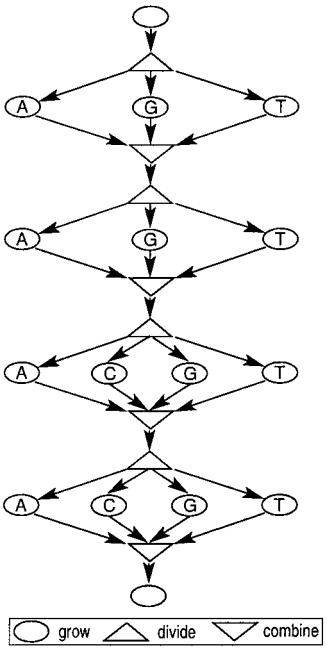

Figure 1. Classic split synthesis procedure synthesizing the 10-string library AATT, AGGA, ATTT, GAAT, GAGG, GTCC, TATT, TGGA, TTAA, and TTGT. It has 14 grow steps and synthesizes 144 compounds.

Library construction proceeds through an interleaved sequence of divide, grow, and combine operations. A set of beads can be *divided* or partitioned into different chambers, and all the molecules on the beads in a particular chamber can be *grown* or extended by the same set of residues. After this reaction, the sets of beads can be *combined* or mixed together so they can be grown within the same chamber. Once combined, two sets of beads cannot be separated again, although the bead mixture can be divided at will.

It is straightforward to use split synthesis to fabricate a complete library of, for example, all 20^l l-peptides (over the 20 amino acids) or 4^l oligonucleotides (over the bases A, C, G, and T): partition the initial beads into 4 or 20 chambers, grow each by a different residue, combine all the beads, and repeat for a total of l iterations.

Much more complicated is fabricating beads for each instance of a class of possible patterns or *motifs*. A typical simple pattern of interest might be all l-peptides with certain critical residues in fixed positions.

For slightly more complex motifs, an efficient synthesis schedule is difficult or impossible to design by hand. We illustrate this with a small example. Consider the target library consisting of the 10 oligonucleotides {AATT, AGGA, ATTT, GAAT, GAGG, GTCC, TATT, TGGA, TTAA, TTGT}. Figure 1 illustrates the straightforward split synthesis protocol for its construction, which requires 14 base-extension steps and yields $3 \times 3 \times 4 \times 4 = 144$ compounds. Figure 2 illustrates the parallel protocol for its construction, requiring 40 base-extension steps, but which yields exactly the 10 target compounds. Figure 3 illustrates an optimized procedure produced by our algorithm which constructs exactly the target library using only 18 base-extension steps.

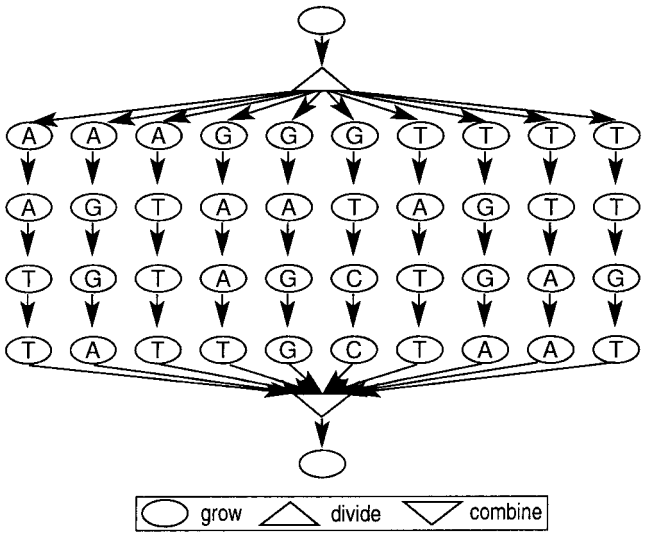

Figure 2. Parallel procedure synthesizing the same 10-string library as Figure 1. It has 40 grow steps and synthesizes 10 compounds.

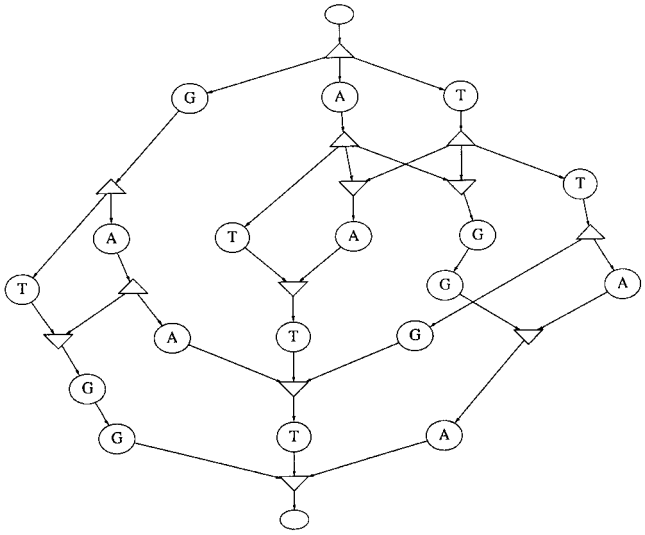

Figure 3. Optimized procedure synthesizing the same 10-string library as Figure 1. It has 18 grow steps and synthesizes 10 compounds.

In this paper, we present techniques for constructing efficient split synthesis protocols which fabricate a desired target library. We seek to minimize the number of base-extension or *grow* operations performed over the course of the synthesis, since these most accurately reflect the cost and complexity of the protocol. Note, however, that we can construct a superset of the specified library if it will simplify the synthesis, provided the total number of compounds synthesized does not grow too large. Thus optimal protocol design involves making the right tradeoff between these two criteria.

1.2. Organization

Our paper is organized as follows. In Section 2 we present our model for split synthesis library design and discuss our system architecture and associated implementation issues. Experimental results are reported in Section 3. Section 4 presents our conclusions concerning the sizes and types of libraries which our techniques hold promise to build, as well as open problems. The Appendix presents algorithmic details on our two distinct and complementary computational approaches to library construction which, although necessary for anyone building a program to optimize synthesis, may distract the reader from our main point—the new library design possibilities which our techniques provide.

Our efforts to improve split synthesis library techniques are a natural follow-up to our previous work with a related technology: constructing dense oligonucleotide arrays on the Southern Array Maker apparatus for interactive sequencing by hybridization, more fully described in refs 7 and 8.

2. Modeling Split Synthesis

We seek to develop and implement algorithms which take as input a set of desired compounds, such as peptides or oligonucleotides, and output an efficient split synthesis schedule of reactions to fabricate them.

Split synthesis can be modeled by a directed acyclic graph (dag) *G*. Figure 1 is an example. *G* consists of *nodes*—circles or triangles—and *edges*, i.e., connecting lines between nodes. Each circular node represents a grow step on a particular subset of beads. The node is labeled with the unit which is appended by the grow operation. A combine step is represented by an inverted triangle, and a divide step is represented by a right-side-up triangle.

The topmost node is the *source* of *G*; it represents the pristine set of resin beads. Every path from the source along the edges represents a possible trajectory of a bead in the synthesis process. Therefore, the set of labels on a path represents a particular compound. The set of all paths from the source to node v_i corresponds to the set of commingled beads at v_i.

Of the three laboratory operations—grow, combine, and divide—the grow operation is the most time-consuming. Therefore, to optimize the design of a synthesis procedure, we seek to minimize the number of grows.

Our algorithms start by building a simple directed acyclic graph *G* which constructs all compounds of the desired target library and refines it by performing a series of local operations on nodes and edges, each of which brings us to a more efficient synthesis procedure, while ensuring that all desired paths remain.

We have shown[9] that finding a dag with a minimum number of nodes which synthesizes a specified target library is NP-complete.[10] This means that it is impractical to compute the *exact optimal* solution for nontrivial inputs. For this reason, we have developed two heuristic approaches to optimization: *bottom-up* and *top-down*, which differ in how the initial dag is constructed and which local operations are employed. In Section 3, we provide the results of simulations to help determine which of these heuristics is better.

We have implemented our algorithms in C++, as part of a system (Figure 4) which consists of the following components:

(i) The *optimizer* accepts a target set of compounds over an arbitrary alphabet. A typical alphabet might range from size 4 (nucleotides) to 20 (amino acids). The size of the input, i.e., the number of input compounds times their length, is limited by available memory. One million bases is feasible

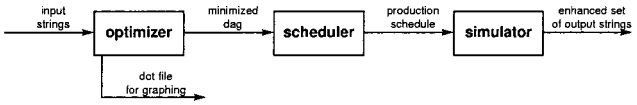

Figure 4. System architecture: accepts set of target strings and outputs an optimized laboratory procedure for synthesizing a superset of the input.

on a machine with 384 megabytes of memory. Our optimizer also takes as input an upper bound on the total number of compounds that is desirable to synthesize.

The optimizer outputs an abstract representation of the synthesis process in the form of a dag, various statistics on the dag, and also a dot file[11] which is suitable for displaying pictures of small dags.

(ii) The *scheduler* takes a dag file as input as well as taking various parameters of the laboratory setting. Most important is the number of grow operations which can be performed in parallel. The scheduler outputs a detailed specification of the steps of the laboratory synthesis procedure.

(iii) The *simulator* emulates the specified laboratory procedure and outputs the total set of compounds it produces.

3. Experimental Results

To establish the range of input sizes over which our techniques are practical and to compare the performance of our two heuristics, we performed extensive experiments on the following data sets.

(i) Experiments were performed on all *k*-mers of the 5243-nucleotide viral DNA sequence SIV (Simian immune deficiency) for $k = 10$, 12, 14, 16, 18, and 20. For example, there are a maximum of 5234 possible 10-mer subsequences, beginning at positions 1, 2, ... 5234. The actual number is lower due to duplicates. This data set is intended to be representative of libraries designed to combat a specific target organism.

(ii) Experiments were also performed on the language RE, containing 7776 10-mers formed by all combinations of *a*, *b*, and *c* in positions 1, 3, 5, 7, and 9, and *a* and *b* in positions 2, 4, 6, 8, and 10. This is the language of the regular expression $((a + b + c)(a + b))^5$. This data set represents libraries designed around highly structured motifs.

(iii) The data sets include the languages REMINUS and REPLUS, created by deleting or adding, respectively, 256 random elements to RE. These data sets are intended to represent minor variations of highly structured libraries.

(iv) Experiments were performed on all *k*-peptides of the 500-peptide protein FTF-HUMAN and the 1000-peptide protein MYSH-CHICK for $k = 5$, 6, 7, 8, 9, and 10. These data sets evaluate our techniques for the 20-character amino acid alphabet associated with proteins.

(iv) Experiments were performed on the languages RND-10, RND-12, RND-14, RND-16, RND-18, and RND-20, each consisting of 256 randomly generated words of the prescribed length over the alphabet {A,C,G,T}. These data sets evaluate our techniques on smaller libraries than presented above.

Our results for synthesizing these 21 libraries are given in Tables 1−5 and Figures 5−9. The following is a brief key to reading the tables. The first column, "input", gives the name of the target library. In library names with numeric parts, the numeric quantity indicates the length of the compounds in the library. For example "SIV-10" denotes the set of all subsequences formed by 10 consecutive nucleotides in SIV.

The second column in each table denotes the number of distinct compounds (oligonucleotides or polypeptides) in each target library. The third column, "base count", is the total number of bases (e.g., nucleotides or peptides) in the target library. This is the target library size times the length of each compound. For example, Table 2 reports that there are 991 distinct polypeptides in MYSH-10, each 10 bases long, for

Table 1. Optimization of SIV *k*-mers[a]

input	oligo count	base count	bottom-up optimization	top-down optimization
SIV-10	5 107	51 070	968	634
SIV-12	5 155	61 860	2 480	2 795
SIV-14	5 163	72 282	5 121	7 020
SIV-16	5 165	82 640	8 691	9 884
SIV-18	5 167	93 006	12 292	13 601
SIV-20	5 169	103 380	16 890	18 381

[a] Number of compounds is 100 000.

Table 2. Optimization of MYSH-CHICK[a]

input	peptide count	base count	bottom-up optimization	top-down optimization
MYSH-05	996	4 980	503	629
MYSH-06	995	5 970	906	1 060
MYSH-07	994	6 958	1 411	1 531
MYSH-08	993	7 944	1 918	2 090
MYSH-09	992	8 928	2 531	2 639
MYSH-10	991	9 910	3 180	3 190

[a] Number of compounds is 20 000.

Table 3. Optimization of Protein FTF-HUMAN[a]

input	peptide count	base count	bottom-up optimization	top-down optimization
FTF-05	496	2 480	324	361
FTF-06	495	2 970	515	688
FTF-07	494	3 458	824	965
FTF-08	493	3 944	1 108	1 283
FTF-09	492	4 428	1 426	1 597
FTF-10	491	4 910	1 748	1 915

[a] Number of compounds is 20 000.

Table 4. Optimization of RE, REPLUS, and REMINUS

			bottom-up optimization		top-down optimization	
input	string count	base count	final nodes	strings added	final nodes	strings added
RE	7 776	77 760	25	0	25	0
REMINUS	7 520	75 200	99	234	25	0
REPLUS	8 032	80 320	404	5 505	524	1 703

Table 5. Synthesis of Random Strings over {*A*, *C*, *G*, *T*}[a]

input	string count	base count	bottom-up optimization	top-down optimization
RND-10	256	2 560	353	234
RND-12	256	3 072	530	430
RND-14	256	3 584	753	602
RND-16	256	4 096	998	808
RND-18	256	4 608	1 243	1 082
RND-20	256	5 120	1 543	1 578

[a] Number of compounds is 50 000.

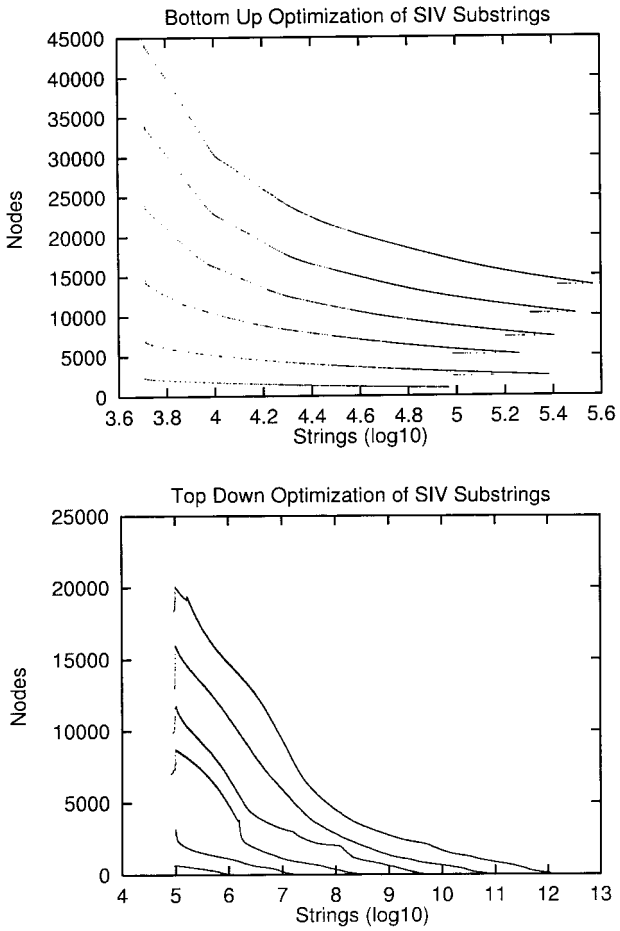

Figure 5. Bottom-up and top-down synthesis of all *k*-mers of the virus SIV ($k = 20$, 18, 16, 14, 12, and 10).

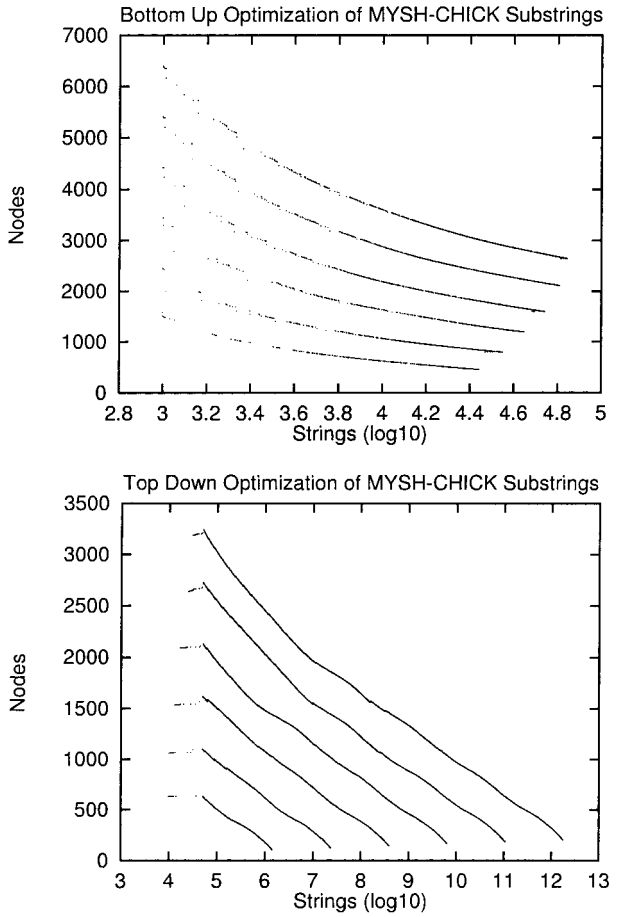

Figure 6. Bottom-up and top-down synthesis of all *k*-peptides of MYSH-CHICK ($k = 10$, 9, 8, 7, 6, and 5).

a total of 9910 bases. Note that most of these libraries are infeasible to fully construct using classical split synthesis methods, for the combinatorial reasons outlined in the introductory section. However, these libraries may be constructed by parallel synthesis (see Figure 2), which requires a number of grow steps equal to the total number of bases in the input. Hence the "base count" provides a baseline for evaluating the efficiency of our synthesis.

We designed synthesis procedures for each target library using both our "bottom-up" and "top-down" algorithms. The fourth and fifth columns of Tables 1−3 present the number of grow steps in the corresponding procedures, where the upper limit on the number of compounds is specified in the table caption.

Figures 5−9 present, in graphical form, the computed tradeoffs between the number of grow steps and the number of compounds synthesized for each family of target libraries. For example, Figure 5a (the "Bottom Up Optimization of SIV Substrings") contains six curves representing (beginning from the top) the results achieved by the "bottom-up" algorithm on target libraries SIV-20, SIV-18, SIV-16, SIV-14, SIV-12, and SIV-10. Each *point* on each curve represents a possible synthesis for the given library. The *y*-coordinate of a given point denotes the number of base-extensions (grow nodes) the synthesis entails, and the *x*-coordinate is the number of compounds (strings) it produces. Thus the experimenter may choose a desired parameter value along either axis and then can find the optimal synthesis which is achieved for that value for the given target library. For example, the appropriate point on the top curve of Figure 5a implies that the synthesis for SIV-20 involving 15 000 grow steps created a total of $10^{5.2} \approx 158\,000$ compounds.

Our primary observations for each of the target libraries are as follows:

(i) SIV Target Libraries. Table 1 and Figure 5 present the results for optimization of the SIV target libraries. Columns 4 and 5 in Table 1 give the number of nodes for the bottom-up and top-down methods when 100 000 total compounds are permitted. The top-down method reduces the number of grow steps for SIV-10 to 634, a reduction in work by a factor of 80.6 compared to the parallel construction.

(ii) MYSH-CHICK Target Libraries. Table 2 and Figure 6 present the results for substrings of the 1000-peptide protein MYSH-CHICK. The optimized procedure reduces the number of grow steps (compared to the parallel synthesis) by a factor 7.9, while keeping the number of extra compounds generated at under 20 000.

(iii) FTF-HUMAN Target Libraries. Table 3 and Figure 7 present similar results for the smaller 500-peptide protein FTF-HUMAN. Grow steps are reduced by a factor of 7.7 over parallel synthesis for FTF-05.

These protein results are quite encouraging, showing that interesting protein libraries can be efficiently synthesized in a modest laboratory using our methods. For comparison, note

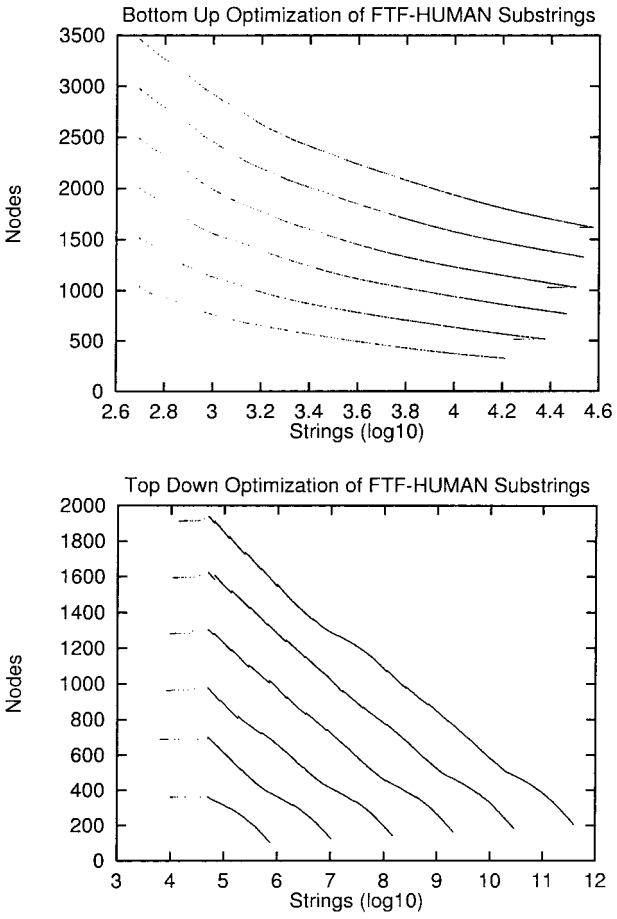

Figure 7. Bottom-up and top-down synthesis of all *k*-peptides of FTF-HUMAN (k = 10, 9, 8, 7, 6, and 5).

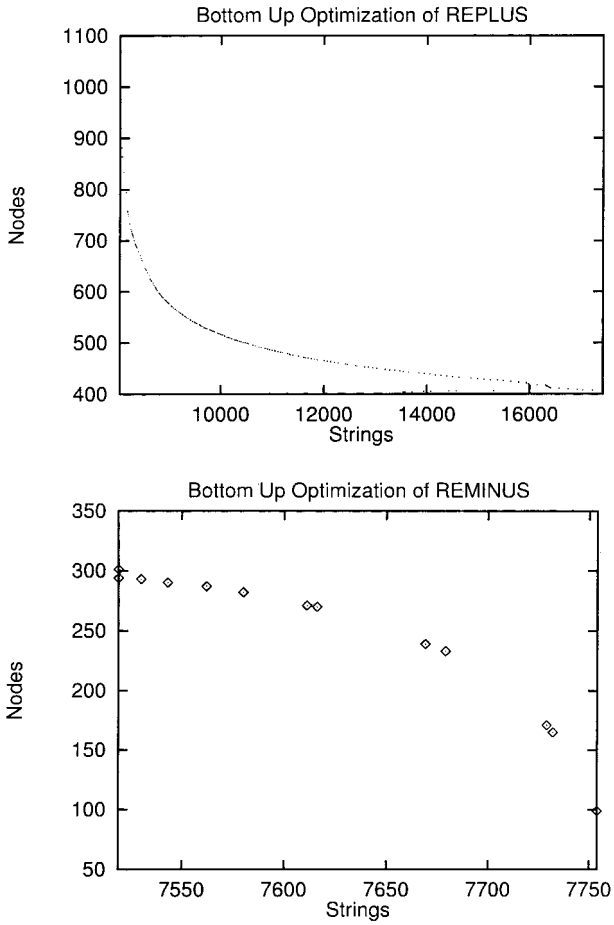

Figure 8. Synthesis of the language of 7776 strings defined by $((a + b + c)(a + b))^5$ with 256 randomly generated strings added (left) and 256 randomly selected strings deleted (right).

that it already strains the capacity of conventional split synthesis technology to construct the complete set of $20^5 =$ 3 200 000 pentapeptides.

(iv) Motif Target Libraries. The motif target libraries include the set of compounds RE of all 7776 possible compounds of length 10 when bases *a*, *b*, and *c* are permitted in positions 1, 3, 5, 7, and 9 and when only bases *a* and *b* are permitted in positions 2, 4, 6, 8, and 10. This set can be easily synthesized using 25 grow steps; both of our optimization techniques arrive at this result. Figure 8 and Table 4 present our results on RE and two related libraries.

The library REMINUS, created by deleting 256 random elements from RE, can be efficiently synthesized using 25 steps by adding back the 256 deleted elements. The top-down approach initially constructs exactly this optimal experiment; the bottom-up method constructs REMINUS with 99 nodes, while adding fewer than 256 elements. The library REPLUS, formed by adding 256 random compounds to RE, is more difficult; the trivial construction of the 256 added elements alone requires 2560 nodes. Our bottom-up approach constructs REPLUS with only 404 nodes when the maximum number of compounds is 15 000; the split approach employs 456 nodes.

(v) RND Target Libraries. Finally, we consider the construction of the family of smaller random libraries. Figure 9 and Table 5 present the results, keeping the number of compounds below 50 000 in every case. Parallel synthesis requires 10.9 times as many grow steps as our synthesis for the case of RND-10.

All the data presented here were generated on a 300 MHz PC with 192 MB of memory. All results presented here for the bottom-up technique were computed in under 15 min. The top-down approach is more time-consuming, but takes under 8 h on each run. We believe that the run times for the second method could be improved significantly were it important to do so.

4. Conclusions

These results indicate that our combinatorial approach to split synthesis can achieve significant improvements across a range of input sizes and on varied alphabets. Our methods may be effectively applied both to DNA/RNA synthesis with an alphabet of four and to protein synthesis with an alphabet of 20.

The efficiencies achieved are greatest in synthesizing libraries with strong regularities, but which are not simple enough to be designed by hand. This suggests our methods may best be applied when the desired library is a set of variations on one or more motifs. The "excess" compounds produced, which we have discussed primarily as an inefficiency, are themselves in fact potentially interesting variations on the target compounds, and hence potentially valuable for drug discovery.

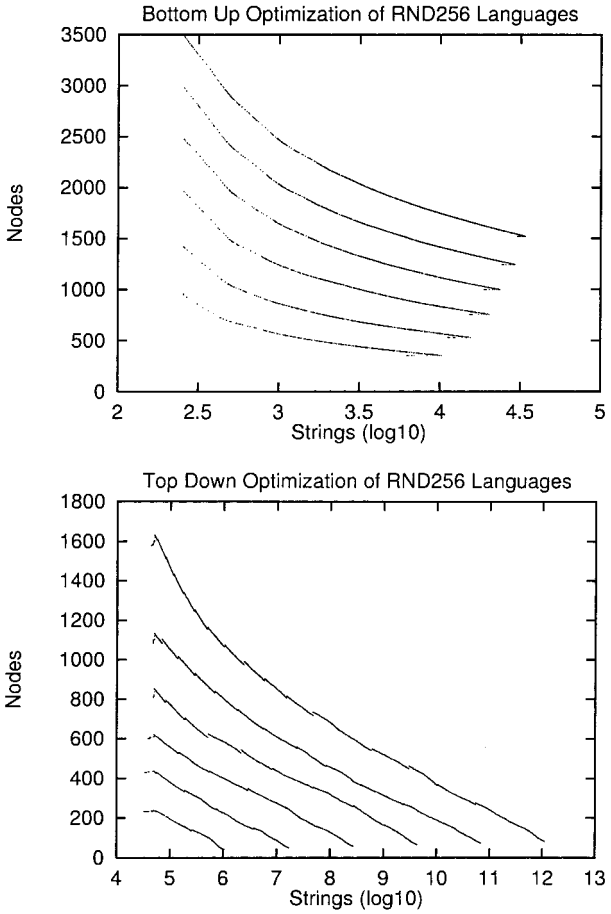

Figure 9. Bottom-up and top-down synthesis of 256 random *k*-strings over {A, C, G, T} ($k = 20$, 18, 16, 14, 12, and 10).

It is interesting to note that the top-down and bottom-up heuristics, which take opposite starting points and use dissimilar methods, achieve closely comparable results on varied inputs in the range we believe most interesting. The top-down approach, which in this range seems slightly inferior to the bottom-up heuristic, offers more promise in applications where the penalty for constructing extra compounds is less than we assume here. In the simple cases of regular expressions, where we can bound the optimum solution, both our methods approach these bounds. Our techniques extend the range of combinatorial libraries which can be synthesized by hand by a factor of perhaps 3 to 10. We can also produce efficient schedules for larger libraries exceeding 10 000 bases, for which an automated synthesis procedure may be necessary to implement the resulting production schedules.

Several interesting algorithmic open problems remain:

(i) Whenever a full set of specified molecules cannot be fabricated within a given resource constraint, we seek to synthesize as large a subset as possible. How best can we exploit this freedom?

(ii) The split synthesis protocol provides the freedom to grow one residue using a subset of more than one amino acid. For example, extending the beads in a chamber using equal amounts of two different reagents will extend half of the molecules by one residue and half by the other. Since each bead contains a large number of molecules, we get equal numbers of copies per bead. How best can we exploit this freedom?

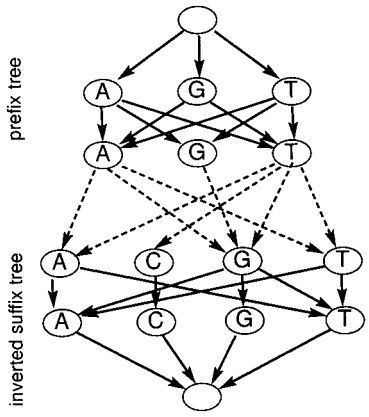

Figure 10. Initial bottom-up construction of a dag for the 10-string library AATT, AGGA, ATTT, GAAT, GAGG, GTCC, TATT, TGGA, TTAA, and TTGT. It has 14 grow steps and synthesizes 51 compounds.

(iii) Similarly, the protocol provides the freedom to construct longer molecules on beads, each of which contains as submolecules representatives of more than one motif. This significantly reduces the number of beads needed, at the cost of more complex synthesis design. How best can we exploit this freedom?

We would be very interested in helping laboratory groups design custom synthesis procedures for their libraries.

Acknowledgment. We thank Jonathan Montague and Peter Nestler of Cold Spring Harbor Laboratory for useful discussions concerning split synthesis and combinatorial chemistry.

Algorithm Appendix

Here we sketch our two distinct but complementary approaches to experimental design, each of which yields an efficient, though not necessarily optimal, solution. See ref 9 for greater detail on our algorithms. In each method, we construct an initial dag *G*, containing all the desired strings. Then we apply a greedy heuristic to identify a sequence of local operations on nodes and edges, each of which brings us closer to the minimal solution, while ensuring that all desired paths remain. Our two approaches, *bottom-up* and *top-down*, differ in how the initial dag is constructed and which local operations are employed.

A.1. Bottom-Up Experimental Design. In this section, we consider a bottom-up approach to optimizing split synthesis. It starts with an initial dag which realizes exactly the target set of strings and then performs local operations that reduce the number of nodes at the cost of possibly adding strings.

A.1.1. Initial Dag Construction. The initial dag for our bottom-up strategy is the conjunction of two rooted trees. Figure 10 illustrates the initial dag construction of the same 10-compound library as in Figures 1−3. One tree, rooted at the source, is built from the prefixes of *S*; and the other (inverted) tree, rooted at the sink, is built from the suffixes of *S*. We divide each input string of length *k* into a prefix of

length $k/2$ and a suffix of length $k/2$. The prefixes are represented by a conventional prefix tree. Level 0 of the tree consists of the source, and on level i at most α^i nodes, one for each distinct prefix of length i.

The suffixes are similarly represented by an inverted tree. The sink will be at level $k + 1$, and there is one leaf for each distinct suffix at level $k/2$. In the suffix tree we reverse the direction of the edges, so that edges point away from the leaves toward the sink.

For each input string $s \in S$ we create an edge between the node corresponding to the last letter of its prefix and the node corresponding to the first letter of its suffix (dashed lines in Figure 10). The result is a dag with a single source and a single sink. Moreover, for each input string there is a unique path through the dag and every path corresponds to an input string.

We can now perform two classes of optimizations. *Fixed library* optimizations alter the dag, but not the set of labeled paths through it. *Enhanced library* optimizations alter the dag, preserving all input paths while adding and/or deleting other paths.

A.1.2. Fixed Library Optimizations. Note that if two nodes, x and y, have the same label and an identical set of parents, they can be merged without altering the set of labeled paths through the dag. This *pairwise reduction* can also be applied to any two nodes with a common set of children, since there is a symmetry between parents and children.

In our experiments, pairwise reductions account for between 2% and 33% of the total reductions achieved over the initial dag, depending on the input.

A second fixed library optimization is *refinement*. Refinement is the deletion of an edge from the dag, possibly adding other edges, in such a way that the total number of paths through the dag is reduced.

A.1.3. The Path Merger Technique. Our strategy for reducing the number of nodes in the dag G is to identify pairs of paths in G which may be merged, eliminating redundant nodes.

We call a path in G *simple* if every node after its head node has only one parent and every node before its tail node has only one child. Each node in G is a simple path of length 1.

Let S and T be two identically labeled simple paths in G of length L occurring the same distance from the source. Figure 11a shows an example in which path S consists of nodes ($v_{i+1,1}$, $v_{i+2,1}$, $v_{i+3,1}$) and path T consists of ($v_{i+1,2}$, $v_{i+2,2}$, $v_{i+3,2}$).

We may merge S and T by merging the corresponding nodes (Figure 11b). Path merger preserves all existing labeled paths through G and reduces the number of nodes in G by the length of the paths (the benefit) while possibly adding new paths through G (the cost). In Figure 11, three nodes are deleted and two paths are added.

The merger of two nodes is a special case of path merger in which the paths are of length 1.

In our implementation, we apply path merging in strictly greedy fashion. All combinations of cost and benefit are ranked, and the cheapest are performed first. Mergers are performed until the cumulative cost in strings equals the maximum price we are willing to pay.

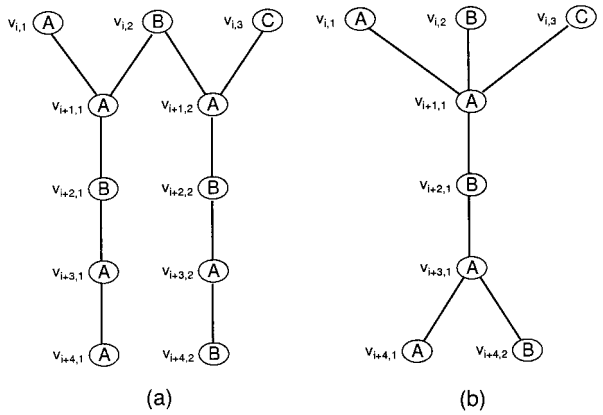

Figure 11. (a) Simple paths consisting of nodes ($v_{i+1,1}$, $v_{i+2,1}$, $v_{i+3,1}$) and ($v_{i+1,2}$, $v_{i+2,2}$, $v_{i+3,2}$) are candidates for merging. (b) Dag after the simple paths have been merged.

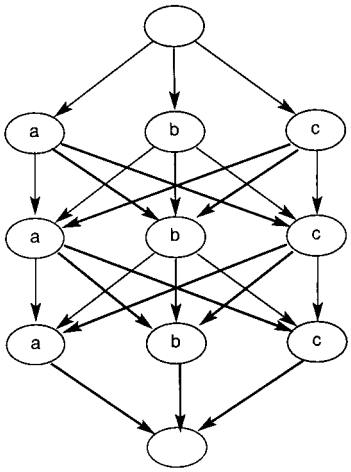

Figure 12. Initial construction of a top-down dag. There is at most one node per base per level. This dag constructs all trimers over the alphabet a, b, c.

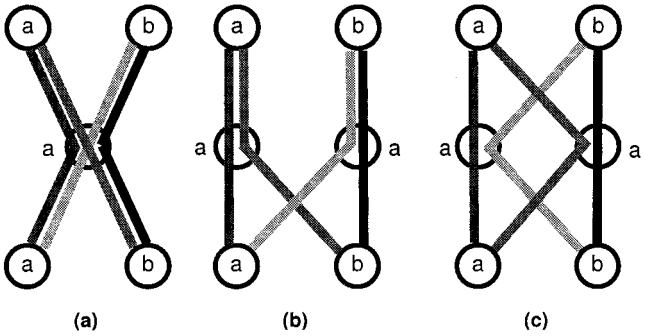

Figure 13. In certain cases, no split of a single node decreases the number of paths through a dag. Continuous lines represent compounds in the target library. A path may be composed of edges (segments) from multiple lines. Here, any split of the middle node of (a) (four paths) results in the same (b) or a greater (c) number of paths (eight paths).

After optimizing G using this merge strategy, refinement typically reduces the number of paths through G by an additional 15−25%.

A.2. Top-Down Experimental Design. Our top-down technique for split synthesis design is similar to the straightforward split synthesis technique; it starts with an initial dag

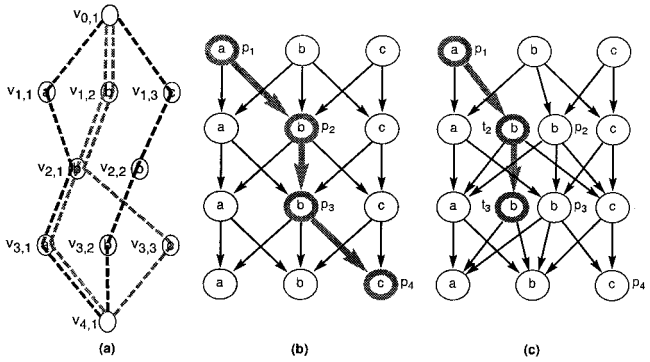

Figure 14. (a) A path is forbidden if no string passes through all its vertexes. Path ($v_{1,1}$, $v_{2,1}$, $v_{3,3}$) is forbidden. (b) Assume $p = (p_1, p_2, p_3, p_4)$ is forbidden. Construction (c) eliminates all paths through G containing p as a subpath.

with the smallest possible number of nodes which realizes the target library. It differs from classical split synthesis in that it omits inessential edges. Such an initial construction (Figure 12) realizes a large fraction (perhaps all) of possible length k strings over the given alphabet. We then perform local operations to reduce the number of undesired strings while adding a small number of additional nodes. The potential advantage of this approach is that our initial dag comprises at most $k\Sigma$ nodes (where Σ is the size of the alphabet), so we can afford a large number of splits which reduce unneeded strings.

We start with either zero or one node of each given label on each level of G, depending on whether the given label appears at that position in the input. On a large input, this may yield a level-wise complete dag, in which each node is a neighbor of every node on neighboring levels. Such a complete dag generates Σ^k paths, most of which are likely to be unnecessary.

Our strategy is to reduce the number of paths through G by splitting nodes. To split a node v, we replace it by two child nodes, v' and v'', which have the same label as v. We must construct edges for the child nodes which ensure that no paths required to construct the target library are eliminated.

Unfortunately, after our initial construction it is often the case that no node can be split in a way which reduces the number of paths through G, as in Figure 13a. Every split produces the same number of paths (Figure 13b) or a greater number (Figure 13c). A more powerful technique is required.

This technique is *path splitting*. We call a path *forbidden* if it is not necessary for the construction of any string in the target library. Path splitting is a construction which splits the internal nodes, that is, all except the first and last nodes, of any path of length 3 or greater (Figure 14). When $|p| = 3$, a single node is split. This construction preserves all paths necessary to synthesize the target library.

The benefit is the number of paths eliminated from G. The cost is the number of nodes added, which is $|p| - 2$.

We apply the path-splitting strategy according to a greedy criterion. All forbidden paths less than a critical length are located, their cost/benefit ratio calculated, and the best split performed. At specified intervals, pairwise reductions are performed. G is refined at the conclusion of path splitting.

References and Notes

(1) Lebl, M.; Krchnak, V.; Sepetov, N.; Seligmann, B.; Strop, P.; Felder, S.; Lam, K. One-bead-one-structure combinatorial libraries. *Biopolymers (Pept. Sci.)* **1995**, *37*, 177−198.

(2) Zhao, P. L.; Zambias, R.; Bolognese, J. A.; Boulton, D.; Chapman, K. T. Sample size determination in combinatorial chemistry. In *Proc. Natl. Acad. Sci. U.S.A.* **1995**, *92*, 10212−10216.

(3) Furka, A.; Sebestyen, F.; Asgedom, M.; Dibo, G. General method for rapid synthesis of multicomponent peptide mixtures. *Int. J. Pept. Protein Res.* **1991**, *37*, 487−493.

(4) Lam, K.; Salmon, S.; Hersh, E.; Hruby, V.; Kazmierski, W.; Knapp, R. A new type of synthetic peptide library for identifying ligand-binding activity. *Nature* **1991**, *354*, 82−86.

(5) Fodor, S.; Read, J.; Pirrung, M.; Stryer, L.; Lu, A.; Solas, D. Light-directed, spatially addressable parallel chemical synthesis. *Science* **1991**, *251*, 767−773.

(6) Dong, D. L.; Liu, R.; Sherlock, R.; Wigler, M. H.; Nestler, H. P. Molecular Forceps from Combinatorial Libraries Prevent the Farnesylation of Ras by Binding to Its Carboxy-Terminus. *Chem. Biol.* **1999**, *6* (3), 133−141.

(7) Bradley, R.; Skiena, S. Fabricating arrays of strings. In *Proceedings of the First International Conference of Computational Molecular Biology (RECOMB ′97)*; 1997, pp 57−66.

(8) Margaritis, D.; Skiena, S. Reconstructing strings from substrings in rounds. In *Proceedings of the 36th IEEE Symposium Foundations of Computer Science (FOCS)*; , 1995, pp 613−620.

(9) Cohen, B.; Skiena, S. Optimizing Combinatorial Library Construction via Split Synthesis. In *Proceedings of the Third Annual International Conference of Computational Molecular Biology (RECOMB ′99)*, 1999, pp 124−133.

(10) Garey, M. R.; Johnson, D. S. *Computers and Intractability: A Guide to the theory of NP-completeness*; W. H. Freeman: San Francisco, 1979.

(11) Kousofios, E.; North, S. Drawing graphs with dot−dot user's manual. Technical Report, AT&T Bell Laboratories, 1993.

CC990028A