On the Primer Selection Problem in Polymerase Chain Reaction Experiments

William R. Pearson[†], Gabriel Robins, Dallas E. Wrege, and Tongtong Zhang

Department of Computer Science, University of Virginia, Charlottesville, VA 22903-2442
[†] Department of Biochemistry, University of Virginia, Charlottesville, VA 22903-2442

Abstract

In this paper we address the problem of primer selection in polymerase chain reaction (PCR) experiments. We prove that the problem of minimizing the number of primers required to amplify a set of DNA sequences is $\mathcal{NP}$-complete. Moreover, we show that it is also intractable to approximate solutions to this problem to within a constant times optimal. We develop a branch-and-bound algorithm that solves the primers minimization problem within reasonable time for typical instances. Next, we present an efficient approximation scheme for this problem, and prove that our heuristic always produces solutions with cost no worse than a logarithmic factor times optimal. Finally, we analyze a weighted variant, where both the number of primers as well as the sum of their costs is optimized simultaneously. We conclude by addressing the empirical performance of our methods on biological data.

1 Introduction

The polymerase chain reaction (PCR) has revolutionized the practice of molecular biology, making it routine to synthesize millions of copies of a single gene or other portion of a genome [5]. PCR has been used to synthesize nanogram quantities of a gene from a single sperm (and thus a single DNA molecule), a 10^{14}-fold amplification [1].

Computer programs [7] [10] [11] are used extensively to design PCR *primers* (i.e., short stretches of DNA, 15 to 20 nucleotides long, that are used to establish the ends of the PCR reaction). In general, these programs have focused on optimizing a pair of nucleotide sequences for amplifying a single sequence in a complex genome (which may contain $3 \cdot 10^9$ sites) and avoiding various artifacts that can be encountered with PCR. Thus, the computer program is given a single DNA sequence, which might contain several hundred to thousands of potential primer sites, and the sites that optimize some simple sequence composition properties are selected.

In this paper, we describe an approach to the solution of a related problem - the amplification of previously undiscovered members of a multigene family by designing primers that will function on the largest possible set of known members of the family. Large families of related genes have become surprisingly common over the past 5 years. One the largest known families contains more than 1000 related genes that encode proteins called G-protein-coupled receptors [6]. However, many families encode a large set of related proteins with essential roles; PCR amplification is usually the most effective method for characterizing members of such large gene families. Here the problem is quite different from the typical primer selection process (where the

objective is to amplify a single specific sequence). We are given a set of 5 to 50 (or more) members of a family of genes, each of which has 20 to 100 potential primer sites on each end of the region to be amplified, and we must select a set of primers that would function on the largest possible number of family members, with the hope that such primers will also allow new members of the family to be amplified.

The typical primer selection program identifies two primers for the two ends of the PCR-amplified region. In contrast, our goal is to select a set of primers from sequences on one end of the region to be amplified. The process must be repeated on a second set of sequences to select primers for the other end of the region to be amplified.

We offer both theoretical and practical contributions. On the negative side, we prove that minimizing the number of primers required to amplify a given set of sequences is $\mathcal{NP}$-complete (Section 3). Moreover, we show that one can not even hope to solve this problem approximately in an efficient manner. On the positive side, in Section 4 we give a branch-and-bound algorithm that solves the primer minimization problem within reasonable time for typical instances. We also give an efficient approximation algorithm for this problem, and prove that our heuristic always produces solutions that are guaranteed to have cost no worse than a logarithmic factor times optimal (Section 5). Finally, in Section 6 we analyze a weighted variant, where both the number of primers as well as the sum of their "costs" must be minimized simultaneously. We discuss in Section 7 the empirical performance of our methods on biological data, and conclude in Section 8 with future research directions. A preliminary version of this work has appeared in [13].

2 Notation and Problem Formulation

Before we formulate the problem of minimizing the number of primers required to synthesize a given set of DNA sequences, we first develop the necessary notation. We use small lowercase italic letters (e.g. "a") to denote characters and strings, uppercase letters (e.g. "A") to denote sets, and uppercase calligraphic letters (e.g. "$\mathcal{A}$") to denote collections of sets.

Let $S = \{s_1, ..., s_n\}$ be a finite set of strings over a finite alphabet Σ (of nucleotides). The concatenation of two strings u and v, denoted by uv or $u \cdot v$, is defined as the string formed by all the symbols of u followed by all the symbols of v. For any finite set of symbols Σ, we use Σ^* to denote the set of all finite strings of symbols from Σ. For example, if $\Sigma = \{a, b\}$, then $\Sigma^* = \{\epsilon, a, b, aa, ab, ba, bb, aaa, aab, \ldots\}$, where ϵ denotes the unique *empty string* of length 0. For two strings $u, v \in \Sigma^*$, u is a *substring* of v if u is a contiguous subsequence of v, and we denote this as $u \preceq v$; i.e., $u \preceq v$ implies that there exist $x, y \in \Sigma^*$ such that $xuy = v$. The length of a string u is denoted by $|u|$. For a collection of sets $\mathcal{C}$, we denote the union of all of its members as $\bigcup \mathcal{C} = \bigcup_{C \in \mathcal{C}} C$.

A set of strings is said to be of *order* k if all the strings have a common substring of length k or more.

Thus, given a string set $S = \{s_1, ..., s_n\}$, if there exists a $u \in \Sigma^*$ with $|u| \geq k$, such that $u \preceq s_i$ for all $1 \leq i \leq n$, then S is a string set of order k, and u is their (not necessarily unique) common substring of length at least k. We then say that u *induces* the string set S, and that S is the string set associated with u. The *size* of S is the number of strings in S, denoted by $|S|$. If a subset S' of S is of order k, then this is denoted as $S' \sqsubseteq_k S$. A string subset is *maximal* if it is not a proper subset of any other string subset of the same order. We denote the collection of all string subsets of S of order k as $\mathcal{S}_k = \{S' \mid S' \sqsubseteq_k S\}$. If for some $\mathcal{C} \subseteq \mathcal{S}_k$, we have $S \subseteq \bigcup \mathcal{C}$, then we say that $\mathcal{C}$ is a *cover* for S of order k and size $|\mathcal{C}|$. An *optimal* cover of order k is a cover of order k having minimum size. In Section 6 below we extend the definition of "optimal" cover to take into account inexact string matching.

For example, the set $S = \{\underline{caba}ca, a\underline{caba}b, bba\underline{caba}\} \subset \{a, b, c\}^*$ is a string set of order 4, since $\underline{caba}$ is a common substring of length 4 for each string in S (we use the underline notation to highlight common substrings). Note that $S = \{ca\underline{ba}ca, \underline{ac}abab, bb\underline{ac}aba\}$ is also a string set of size 3 and order 2, since all strings in S have the common substring $\underline{ac}$ of length 2. On the other hand, S is not a string set of order 5, since there exists no substring of length 5 common to all the strings of S. We observe that S contains a maximal string subset of order 5 and size 2, namely $\{\underline{acaba}b, bb\underline{acaba}\}$, associated with the common substring $\underline{acaba}$ of length 5. Finally, the two string subsets contained in $\mathcal{C} = \{\{\underline{acaba}b, bb\underline{acaba}\}, \{\underline{caba}ca\}\}$ form an optimal cover for S of order 5 and size $|\mathcal{C}| = 2$, while the single string subset $\{\underline{caba}ca, a\underline{caba}b, bba\underline{caba}\}$ (i.e., S itself) forms an optimal cover for S of order 4 and size 1.

In our formulation, a string corresponds to a DNA sequence, a substring corresponds to a primer or a portion of a primer, and a string (sub)set corresponds to a *primer set*; we use these terms interchangeably in what follows, depending on context. Although above we did not restrict the alphabet size, in biological applications the alphabet typically consists of the four nucleotide bases adenine, cytosine, guanine, and thymine, abbreviated as $\Sigma = \{a, c, g, t\}$.

Given a set of DNA sequences (strings), there are many choices as to which primers (i.e., common substrings) we can synthesize to amplify (i.e., cover) different sequence subsets (i.e., string subsets). Moreover, to keep the problem realistic, we insist that all primers have length k or more, otherwise we could, for example, trivially use a single primer of length zero (i.e., the empty string) to cover all of the DNA sequences, which would not be useful biologically. Yet, even if we set an a priori lower limit on the primer length (not greater than the shortest DNA sequence), any set of DNA sequences can be covered by using a single distinct primer for every DNA sequence (e.g., the DNA sequence itself). However, such a solution would be unlikely to allow us to discover new genes. With this in mind, we seek to minimize the number of primers of a specified length necessary to cover a given set of DNA stands:

Optimal Primer Cover (OPC) Problem: Given a finite set S of DNA sequences and an integer k, find an optimal cover for S of order k.

3 Complexity of the OPC Problem

Our first theoretical result establishes the intractability of the optimal primer cover problem.

Theorem 3.1 *The OPC problem is $\mathcal{NP}$-complete.*

Proof: Clearly the decision version of the OPC problem is in $\mathcal{NP}$. To complete the proof that OPC is $\mathcal{NP}$-complete, we transform a well-known problem in $\mathcal{NPC}$ to the OPC problem, namely the *minimum set cover* (MSC) problem, which is defined as follows:

Minimum Set Cover (MSC) Problem: Given a collection $\mathcal{M}$ of subsets of a finite set T and a positive integer h, does there exist in $\mathcal{M}$ a cover for T of size at most h? (i.e., is there a $\mathcal{M}' \subseteq \mathcal{M}$ such that $|\mathcal{M}'| \leq h$ and $T \subseteq \bigcup \mathcal{M}'$?)

We now show how to transform an arbitrary instance $< T, \mathcal{M}, h >$ of MSC into an instance $< S, k, l >$ of OPC, in such a way that $< S, k, l >$ has a solution if and only if $< T, \mathcal{M}, h >$ has a solution.

Given an arbitrary instance $< T, \mathcal{M}, h >$ of the MSC problem, set $l = h$, $\Sigma = \{0, 1, b_1, b_2, \ldots, b_{|T|}\}$, and $k = \lceil \log_2 |\mathcal{M}| \rceil$ (the b_i's will be used as "separators" to delineate substrings in the encoding described below). We will construct a set S of strings over Σ where each string $s_i \in S$ represents a distinct element $t_i \in T$, with s_i encoding the subset membership information of its corresponding t_i (i.e., the encoding s_i reflects which M_i in $\mathcal{M}$ contain t_i). Thus, for every $M_i \in \mathcal{M}$, the construction places some common substring u_i in all strings in S that correspond to the elements in M_i. We encode each $M_i \in \mathcal{M}$ by a unique string u_i over $\{0, 1\} \subset \Sigma$ with $|u_i| = k$, and concatenate u_i and the unique "separator" symbol b_j to every $s_j \in S$ that corresponds to each $t_j \in M_i$. In other words, if the subsets $M_{i_1}, M_{i_2}, \ldots, M_{i_n}$ are exactly those that contain an element t_j, we construct $s_j = u_{i_1} b_j u_{i_2} b_j \cdots u_{i_n} b_j$. This scheme (see Figure 1) will clearly induce a string subset $S_i \sqsubseteq_k S$ corresponding to M_i, since $u_i \preceq s_j$ for all s_j corresponding to $t_j \in M_i$.

Although it is clear from the construction that each subset $M_i \in \mathcal{M}$ has a corresponding string subset S_i, it is not obvious that our construction avoids introducing maximal subsets of order k that do not correspond to any subset $M_i \in \mathcal{M}$. We therefore now argue that the transformation does not induce such spurious maximal subsets.

Assume to the contrary that a spurious maximal string subset S' of order k exists, and consider the string/primer u associated with $S' \sqsubseteq_k S$. Since by assumption S' is not associated with any subset $M_i \in \mathcal{M}$, u cannot be equal to any u_i formed strictly from elements in $\{0, 1\}$ by the construction (otherwise S' would *exactly* correspond to some subset in $\mathcal{M}$). But the size of u is at least as large as the size of the u_i's (namely k symbols long), so if u is not equal to any of the u_i's, then u must contain some separator symbol b_j. However, the symbol b_j occurs only in the string s_j, and thus the size of the string subset S' is at most 1

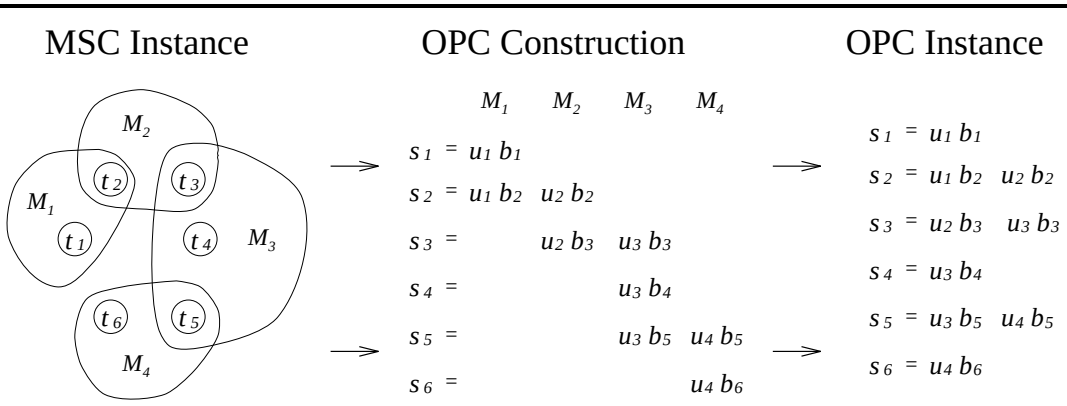

Figure 1: The construction of an instance of OPC from an arbitrary instance of MSC. A unique string u_i is used to encode each subset M_i (i.e., here the unique strings u_1, u_2, u_3, and u_4 are created to represent M_1, M_2, M_3, and M_4, respectively). Also, each string s_j uses its own unique separator b_j in between the u_i's. Each string s_j is formed according to the construction above; for example, t_5 is a member of both M_3 and M_4, and thus the corresponding string s_5 is set to $u_3 b_5 \cdot u_4 b_5$.

(i.e., $S' = \{s_j\}$). The fact that s_j is not the empty string (since it contains u) implies that the element $t_j \in T$ corresponding to s_j must be contained in some $M_{j'} \in \mathcal{M}$, and moreover $|M_{j'}| = 1$, otherwise there would be some substring $u_{j'} \preceq s_j$ that would induce a string subset of order k strictly containing S', contradicting the assumed maximality of S'. It follows that if S' is maximal, then it is not spurious. □

In Theorem 3.1, the alphabet size of the OPC instance depends on the MSC instance (i.e. $|\Sigma|$ is a function of $|T|$). In biological applications however, the alphabet is of constant size, independent of the input (i.e., $\Sigma = \{a, c, g, t\}$, so $|\Sigma| = 4$). We therefore need to show that the OPC problem with alphabet $\Sigma = \{a, c, g, t\}$ remains $\mathcal{NP}$-complete, and this will be accomplished using an argument similar to that used in the proof of Theorem 3.1. Thus, $\{a, c\}$ and $\{g, t\}$ can be used to encode $\{0, 1\}$ and $\{b_1, b_2, \ldots, b_{|T|}\}$ of the unrestricted alphabet, respectively. This enables a one-to-one correspondence between the subsets $M_i \in \mathcal{M}$ and the maximal string subsets $S_i \sqsubseteq_k S$, using only the restricted alphabet $\Sigma = \{a, c, g, t\}$ for the encoding.

4 An Exact Branch-and-Bound Algorithm

We saw above that the MSC problem reduces to the OPC problem. A reduction in the opposite direction (i.e., a transformation of the OPC problem to the MSC problem), will enable the application of techniques for the MSC problem in order to solve the OPC problem. In this section we outline a branch-and-bound exact algorithm for the OPC problem (the next section will outline a more efficient heuristic solution).

The heart of the reduction from MSC to OPC (Theorem 3.1) was a one-to-one correspondence between the subsets of the MSC instance and the string subsets of the OPC instance. With this in mind, we transform the OPC problem to the MSC problem as follows: for each maximal string subset in the OPC instance, exactly

one subset in the MSC instance is created. This enables us to think of the optimal primer cover problem as a "special case" of the minimum set cover problem. In particular, given an instance of $< S, k, l >$ of the OPC problem, for each string $s_i \in S$ we find all length-k substrings $s_j \preceq s_i$, and for each one of these s_j we form the maximal string subset in S associated with s_j; these become the subsets of our corresponding MSC instance. Clearly, a good solution to the resulting MSC instance would constitute a good solution to the OPC instance. We therefore now turn our attention to strategies for solving the minimum set cover problem.

One straightforward scheme to solve the MSC problem optimally is to exhaustively enumerate all $2^{|\mathcal{M}|}$ subset combinations, and select the one containing the smallest number of subsets that covers T. This algorithm considers all possible solutions, and is therefore guaranteed to find the optimal one. However, this brute-force approach runs in time exponential in the number of subsets $|\mathcal{M}|$.

We can greatly improve the performance of this exhaustive algorithm in practice by eliminating large portions of the search space using a branch-and-bound technique. In particular, we use a tree-structured search scheme in which we keep information about partial covers during our search, so that we may recognize certain partial covers that cannot possibly lead to solutions better than the best solution seen so far. Using this information, we prune the search tree and thus avoid examining large portions of the search space. In particular, we search for a maximal cover containing at most h subsets, as shown in Figure 2. By invoking this modified algorithm with all values of h, $1 \leq h \leq |\mathcal{M}|$, we can still consider the entire feasible solution space as before. However, during our search, we keep track of the current best candidate solution and make use of the following Lemma, which enables the branch-and-bound strategy:

Lemma 4.1 *Consider an instance $< T, \mathcal{M}, h >$ of MSC, and a "partial cover" $\mathcal{M}'$ for $T' \subset T$ (i.e., a collection of subsets $\mathcal{M}' \subset \mathcal{M}$, where $\mathcal{M}'$ covers $T' = \bigcup \mathcal{M}'$), and let the cardinality of the largest unused subset in $\mathcal{M}$ be $b = \max_{M_i \in \mathcal{M} - \mathcal{M}'} |M_i|$. Then $\mathcal{M}'$ can not be "extended" by m additional subsets into a cover for T of size $|\mathcal{M}'| + m$, unless $|T'| + m \cdot b \geq |T|$.*

Proof: The number of elements that are not covered by $\mathcal{M}'$ is $|T'| - |T|$. Therefore, if we augment $\mathcal{M}'$ by m additional subsets $\mathcal{M}'' \subset \mathcal{M}$, $|\mathcal{M}''| = m$ such that $\mathcal{M}' \cup \{\mathcal{M}''\}$ covers T, then $|\bigcup \mathcal{M}''|$ must be at least of size $|T| - |T'|$. Thus, the largest subset in $\mathcal{M} - \mathcal{M}'$ must have cardinality $b \geq \lceil (|T| - |T'|)/m \rceil$. □

Based on this observation, we can avoid trying to augment partial covers if there are no remaining unused subsets which are large enough to yield a complete cover competitive with the best cover seen so far during the search. This obviates the examination of large portions of the search space, and leads to significant improvements in actual run times. This scheme is formalized in Figure 2, and we discuss the empirical performance of this optimization in Section 7.

Branch-and-Bound Exact Algorithm for Minimal Set Cover
Input: A set T of elements, a set $\mathcal{M}$ of subsets $M_i \subseteq T$, and integer h.
Output: A collection $\mathcal{M}' \subseteq \mathcal{M}$, $\lvert\mathcal{M}'\rvert = h$, such that $\lvert\bigcup\mathcal{M}'\rvert$ is maximum.
1. **Procedure** Optimal_Algorithm (T, $\mathcal{M}$, h)
2. **Sort** $\mathcal{M} = \{M_1, \ldots, M_{\lvert\mathcal{M}\rvert}\}$ by non-increasing cardinality of M_i
3. OPT $\leftarrow \emptyset$ /* OPT is a global variable */
4. Try_Subset(OPT, h, 1)
5. **Return** OPT
6. **Procedure** Try_Subset($\mathcal{M}'$,left,next)
7. **If** $\lvert\bigcup\mathcal{M}'\rvert >$ |OPT| **Then** OPT $\leftarrow \mathcal{M}'$
8 **If** left= 0 **Then Return**
9. **For** i =next to $\lvert\mathcal{M}\rvert$ **Do**
10. **If** $\lvert\bigcup\mathcal{M}'\rvert$+left$\cdot\lvert\mathcal{M}_i\rvert >$ |OPT| **Then** Try_Subset($\mathcal{M}' \cup \{M_i\}$,left$-1, i+1$)

Figure 2: An exact set cover algorithm, using branch-and-bound to speed up the search: out of all $\binom{\lvert\mathcal{M}\rvert}{h}$ possible covers, the one that covers the greatest number of elements of T is returned. Branch-and-bound occurs when it is determined that the current partial cover can not be extended so that the number of elements it covers exceeds that of the best cover seen so far during the search.

5 A Provably-Good Heuristic

Since the OPC problem is $\mathcal{NP}$-complete, efficient exact algorithms are not likely to exist, and we seek efficient heuristics that yield near-optimal solutions. Based on the results of [12], it can be proved that that no polynomial-time approximation heuristic is likely to solve the OPC problem to within less than a factor of $\frac{1}{4}\log_e |T|$ times optimal in terms of solution quality. Thus, the best polynomial-time approximation scheme that we can hope to find would have a theoretical performance bound of $O(\log |T|)$ times optimal. Next, we show how this theoretical optimum can actually be achieved using a greedy heuristic.

One greedy strategy for the MSC problem is to select the subset M_i that covers the most remaining uncovered elements, and iterate until all elements are covered. This greedy heuristic for set cover can be implemented within time $O(|\mathcal{M}|\log_2 |\mathcal{M}|)$, or with slight modifications, it can be implemented within linear time [4]. A simple worst-case example where the greedy strategy produces a cover of size $(\log_2 |T|)\cdot$OPT is presented in Figure 3.

The performance of this greedy heuristic for the set cover problem has been analyzed extensively in the literature [3] [8] [9] [12]. Johnson presents an example in which the greedy heuristic yields a cover of size of $(\log_e |T|)\cdot$OPT, where OPT is the size of an optimal set cover [8]. Lovasz and Johnson both present a $(\log_e |T| + 1)\cdot$OPT upper bound on the greedy heuristic. Thus, the greedy heuristic performs as well as can be expected, given that it matches the asymptotic lower bound on the performance of any polynomial-time approximation scheme for MSC.

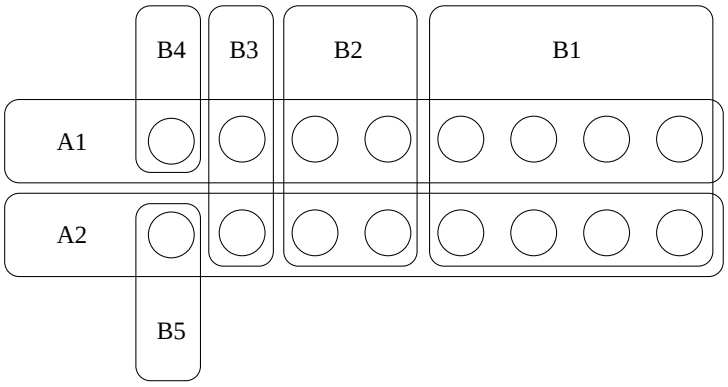

Figure 3: An example for which the greedy heuristic will produce a cover of size $(\log_2 |T|)\cdot$OPT. Here the circles represent the elements to be covered, and the $A1$, $A2$, $B1$, $B2$, $B3$, $B4$, and $B5$ ovals represent the various subsets. Observe that the optimal cover consists of $A1$ and $A2$, while the greedy heuristic may select subsets $B1$, $B2$, $B3$, $B4$, and $B5$, a logarithmic factor times optimal. This example extends to an arbitrary number of elements.

6 The Weighted OPC Problem

The discussion above thus far has been restricted to address the problem of minimizing the *cardinality* of the cover (i.e., the number of primers that are required to specify one end of a PCR reaction from a set of DNA sequences). Thus, the algorithms in Sections 4 and 5 strive to minimize the number of string subsets. In practice, however, the requirements for the length of a PCR primer (15 nucleotides) virtually ensure that a reasonable number of primers (e.g. 5-8) cannot be found that match exactly to 20 or more members of a diverse gene family. Since we wish to identify new members of a family by finding from known sequences a modest number of primers, we must consider how to construct inexact primers.

One method is to produce degenerate oligonucleotide primers. The machines that synthesize primers can be programmed to incorporate 2, 3, or 4 nucleotides in a single polymerization step, thus, it is possible to construct a primer that is actually a mixture of many different sequences. The disadvantage of this approach is that the concentration of each individual sequence is reduced and the mixture of primers may no longer be specific for the gene family of interest. Alternatively, one can construct primers that do not match each sequence exactly, but match all of the members of a set of sequences with only one or two mismatches. In general, because of the biochemistry of the PCR reaction, primers must have an exact match of about 5 nucleotides at one end of the primer; degeneracies or mismatches are then allowed in the remainder of the primer molecule. Thus, primer selection becomes the problem of finding an optimal primer covering of order 5, and then a weighted covering, where the weighting incorporates values for degeneracies or mismatches, for the 10 remaining adjacent nucleotides.

With this in mind, we introduce a *cost function* W that assigns a nonnegative weight to each primer u_i

and its string subset S_i. The cover weight is inversely proportional to the cover "quality": a cover with low weight is considered superior to a cover with higher weight. We define the *optimal cover* in this new *weighted* version to be a cover with minimum total weight. The weighted version of the OPC (WOPC) problem may be formally stated as follows:

Weighted Optimal Primer Cover (WOPC) Problem: Given a finite set S of DNA sequences, a positive integer k, and a nonnegative cost function that assigns a weight to each string set S_i and its associated primer u_i, find a cover $\mathcal{C}$ for S of order k, which minimizes the total weight $\sum_{S_i \in \mathcal{C}} W(S_i, u_i)$.

Given that the OPC problem is $\mathcal{NP}$-complete, it is not surprising that the more general WOPC is $\mathcal{NP}$-complete as well (i.e., set the weights to 1). We next consider a weighting scheme that is tailored specifically to the primers selection problem in biology. To permit inexact matching, we need to develop a weighting scheme that quantifies the "accuracy" of the matches between primers and sequences. Toward this end, we make the cost function W depend on weight contributions from inexact matches between the primer u and the individual strings $s_i \in S'$, denoted by $w(s_i, u)$, so that $W(S', u) = \sum_{s_i \in S'} w(s_i, u)$. Given a primer u and a string s_i, we thus set $w(s_i, u)$ to the number of positions in which s_i differs from u. For example, if $u = abbab$ and $s_1 = ababb$, $w(s_1, u) = 2$, since s_1 differs from u in positions 3 and 4.

The OPC problem naturally extends to the WOPC problem via the introduction of a weighting scheme, and just as we have used techniques from the minimum set cover problem to attack the unweighted case, we can address the weighted case using techniques from the *weighted minimum set cover* (WMSC) problem. The WMSC problem is defined as follows:

Weighted Minimum Set Cover (WMSC) Problem: Given a collection $\mathcal{M}$ of subsets of a finite set T, each subset $M_i \in \mathcal{M}$ having a nonnegative weight $w(M_i)$, and a real value h, does there exist in $\mathcal{M}$ a cover for T of weight at most h? (i.e. is there a $\mathcal{M}' \subseteq \mathcal{M}$ such that $T \subseteq \bigcup \mathcal{M}'$ and $\sum_{M_i \in \mathcal{M}'} w(M_i) \leq h$?)

This weighted variant of the minimum set cover problem is well-studied, and we can therefore use known techniques developed for the WMSC problem in solving the WOPC problem [2] [3]. Clearly, an exact solution to WOPC can clearly be obtained by performing an exhaustive search of all subset combinations. As we did in Section 4, we can decrease the computation time of this exponential algorithm by resorting to branch-and-bound techniques: keeping track of the weights of partial solutions will enable the pruning of numerous branches of the search tree.

Given the analysis in Section 5 of the greedy heuristic for the MSC problem, it is not surprising that a greedy heuristic for the WMSC problem also has a worst-case performance bound of $(\log_e |T| + 1) \cdot \text{OPT}$ [2] [3]. The only difference between the unweighted greedy heuristic and the weighted variant of the heuristic is the selection criteria. At each step, we now select the subset that covers the maximum number of yet-uncovered

elements in T at the lowest cost *per element* (i.e. we select the subset M_i for which $w(M_i)/|M_i|$ is minimum [3]). This extends the unweighted approximation algorithm to a weighted approximation algorithm.

Although the weighted version of OPC is more general than the unweighted version, the following trivial solution must be avoided: for each string $s_i \in S$, consider an exact-match primer being the string itself (i.e., let $u_i = s_i \preceq s_i$), and thus we obtain a trivial solution with $|S|$ string subsets having total weight 0. Although under our formulation above this solution would be considered "optimal" (since it has 0 weight), this is not biologically meaningful. It would therefore be more interesting to pursue an algorithm that simultaneously minimizes both the weight and the number of string subsets in a cover. Unfortunately, there does not exist an algorithm that can simultaneously minimize both the weight and cardinality of a cover with provable non-trivial bounds (Figure 4).

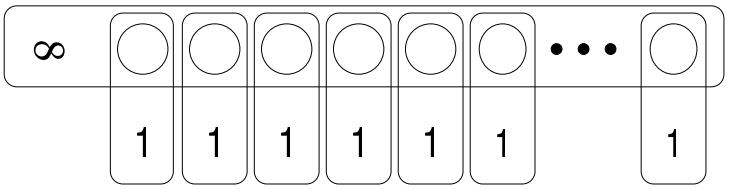

Figure 4: An instance of WMSC illustrating that no algorithm can achieve nontrivial simultaneous bounds on both weight and cardinality of a cover. The circles denote the elements to be covered, while the ovals denote the weighted subsets. Observe that the optimal cardinality of a cover is 1, while the optimal weight of a cover is $|T|$, where T is the set of elements. Clearly there exist no cover which has both small weight and small cardinality.

Despite this negative result, in practice we can nevertheless still construct algorithms that will simultaneously optimize both cover size and weight, and indeed achieves a smooth tradeoff in practice between these two objectives for typical biological instances. For example, we can easily construct a new cost function W' that considers both the cardinality and weight of a string subset S_i by setting $W'(S_i, u_i) = t * W(S_i, u_i) + (1-t) * K$, for some constant K and a real parameter $0 \leq t \leq 1$. If we set $t = 0$, this cost function will consider only cardinality, while setting $t = 1$ will make the cost function consider weight only. As t varies in the interval $[0, 1]$, a reasonably smooth tradeoff will be observed in practice, as we show in the next section.

7 Experimental Results

We implemented the exact algorithm and the approximation algorithms discussed above using the C programming language in the UNIX environment (code is available from the authors upon request). In this section we compare the performance and running-times of three algorithms: the efficient branch-and-bound optimal (BBOPT) algorithm (Figure 2), the greedy (GREEDY1) heuristic, and a greedy variant (GREEDY2) that differs from GREEDY1 in that it selects at each iteration, the *pair* of subsets that together constitute the

best choice. These algorithms were implemented for both the weighted and the unweighted cases. We also implemented the scheme mentioned in Section 6 that simultaneously minimizes both cardinality and weight.

We evaluated the performance of these algorithms on biological data consisting of 56 DNA sequences, each 75 nucleotides long, from transmembrane domain three (TM3) from 56 G-protein coupled receptors [6]; the data itself is given in Figure 5. As mentioned in Section 1, these primers would determine one end of the amplified sequence. The other end would be determined by a second set of primers from a second set of sequences. For the G-protein-coupled receptors, TM7 would provide a natural second target, since it is also highly conserved [6].

We also created 30 random permutations of the codons (i.e., 3-base triplet substrings) of each sequence of the data, and tested our method on all of the resulting instances. For each input instance, both GREEDY1 and GREEDY2 executed within a few milliseconds, while BBOPT required anywhere from several minutes to several hours, depending upon the size of the optimal cover.

The performance of the unweighted versions of the algorithms on the data sets is shown in Table 1. The objective here is to minimize the cardinality of the cover. The cardinality of the solutions produced by BBOPT, GREEDY1, and GREEDY2 are shown in the table. GREEDY1 and GREEDY2 both produced an optimal cover for 19 out of the 30 random permutations, and for the remaining permutations, the solutions produced by GREEDY1 and GREEDY2 are at most 1 primer more than the optimal. We conclude that the two greedy heuristics are quite effective in primer number minimization.

Table 2 shows the performance of the various algorithms for the (weighted) WOPC problem, where the objective is to minimize the total weight of the cover rather than its cardinality. Both the weight and cardinality of the solutions produced by GREEDY1 and GREEDY2 for the data sets are shown in the table. Here GREEDY2 outperforms GREEDY1 on some instances.

Though as we saw in Section 6 that it is impossible to achieve provably-good simultaneous bounds on both the cardinality and weight of a cover, in practice we can still design algorithms which exhibit a smooth tradeoff between these two objectives. We implemented a greedy heuristic with objective function $W'(u_i, M_i) = t * W(u_i, M_i) + (1-t) * K$ mentioned in Section 6 for various values of t in the interval $[0, 1]$. The results are presented in Figure 6. Each data point represents the average values over the 30 runs on the random data for selected values of t. As expected, we observe a smooth tradeoff between cover cardinality and weight.

8 Conclusions and Future Directions

We investigated the problem of minimizing the number of primers in polymerase chain reaction experiments. We proved that minimizing the number of primers necessary is intractable, as is approximating optimal solutions within a constant factor. On the positive side, we gave a practical branch-and-bound exact algorithm,

and an efficient approximation scheme for primer number minimization. We proved that our heuristic is guaranteed to produce solutions with cost no worse than a logarithmic factor times the optimal cost. Finally, we analyzed a weighted variant, where both the number of primers as well as the sum of their "costs" is to be optimized simultaneously. Our algorithms are easy to implement and perform very well in practice on biological data. It is our hope that these techniques would be helpful in finding new proteins.

Future research directions include: (1) investigating alternative heuristics for both the weighted and the unweighted versions of the OPC problem; (2) experimenting with various weighting schemes and criteria for primer selection; (3) exploring additional heuristics for simultaneous tradeoffs between subset cardinality and weight; and (4) running actual PCR experiments to investigate the practical efficacy of these approaches.

9 Acknowledgments

The authors are grateful to the anonymous referees and to the editors of the special issue for their thoughtful comments. This research was supported in part by National Library of Medicine grant LM04961 (Pearson), National Science Foundation Young Investigator Award MIP–9457412, and a Packard Foundation Fellowship (Robins). Copies of related papers by our group may be found at http://www.cs.virginia.edu/~robins/.

References

[1] N. Arnheim, H. Li, and X. Cui, *PCR Analysis of DNA Sequences in Single Cells: Single Sperm Gene Mapping and Genetic Disease Diagnosis*, Genomics, 8 (1990), pp. 415–419.

[2] R. Bar-Yehuda and S. Even, *A Linear-Time Approximation Algorithm for the Weighted Vertex Cover Problem*, J. Algorithms, 2 (1981), pp. 199–203.

[3] V. Chvatal, *A Greedy Heuristic for the Set-Covering Problem*, Mathematics of Operations Research, 4 (1979), pp. 233–235.

[4] T. H. Cormen, C. E. Leiserson, and R. Rivest, *Introduction to Algorithms*, MIT Press, 1990.

[5] H. A. Erlich, D. Gelfand, and J. J. Sninsky, *Recent Advances in the Polymerase Chain Reaction*, Science, 252 (1991), pp. 1643–1651.

[6] J. K. Harrison, W. R. Pearson, and K. R. Lynch, *Molecular Characterization of Alpha-1 and Alpha-2 Adrenoceptors*, Trends Pharm. Sci., 12 (1991), pp. 62–67.

[7] L. Hillier and P. Green, *OSP: a Computer Program for Choosing PCR and DNA Sequencing Primers*, PCR Methods and Applications, 1 (1991), pp. 124–128.

[8] D. S. Johnson, *On the Ratio of Optimal Integral and Fractional Covers*, J. Comput. System Sci., 9 (1974), pp. 256–278.

[9] L. Lovasz, *On the Ratio of Optimal Integral and Fractional Covers*, Discrete Mathematics, 13 (1975), pp. 383–390.

[10] T. Lowe, J. Sharefkin, S. Q. Yang, and C. W. Dieffenbach, *A Computer Program for Selection of Oligonucleotide Primers for Polymerase Chain Reactions*, Nuc. Acids Res., 18 (1990), pp. 1757–1761.

[11] K. Lucas, M. Busch, S. Mossinger, and J. A. Thompson, *An Improved Microcomputer Program for Finding Gene- or Gene Family-Specific Oligonucleotides Suitable as Primers for Polymerase Chain Reactions or as Probes*, Comp. Appl. Biosci., 7 (1991), pp. 525–9.

[12] C. Lund and M. Yannakakis, *On the Hardness of Approximating Minimization Problems*, Proc. ACM Symp. the Theory of Computing, 25 (1993), pp. 286–293.

[13] W. R. Pearson, G. Robins, D. E. Wrege, and T. Zhang, *New Approach to Primer Selection in Polymerase Chain Reaction Experiments*, in Proc. Intl. Conf. on Intelligent Systems for Molecular Biology, Cambridge, England, July 1995, pp. 285–291.

Unweighted OPC Statistics			
input sets	BBOPT Cardinality	GREEDY1 Cardinality	GREEDY2 Cardinality
1	7	7	7
2	7	8	8
3	5	5	5
4	6	6	6
5	7	8	8
6	6	7	7
7	7	7	7
8	6	7	7
9	6	7	7
10	7	7	8
11	6	6	6
12	6	6	6
13	7	7	7
14	7	7	7
15	7	7	7
16	7	7	7
17	6	6	6
18	7	7	7
19	7	7	7
20	6	7	7
21	6	6	6
22	7	7	7
23	7	7	7
24	7	7	7
25	6	7	7
26	6	7	7
27	6	6	6
28	6	7	7
29	6	6	6
30	6	7	7

Table 1: Cardinality of the covers produced by the various algorithms over 30 random permutations of a data set consisting of biological data (56 sequences of 75 nucleotides each). We see that GREEDY1 typically finds optimal solutions, while GREEDY2 has performance very similar to that of GREEDY1.

Figure 5: The biological data used to evaluate the empirical performance of our algorithms; this data consists of 56 DNA sequences, each 75 nucleotides long, from one of the transmembrane domains (TM3) from 56 G-protein coupled receptors. The names in the left column indicate the organism from which the sequence originated. The substrings underlined indicate a maximal primer set containing 24 sequences, with the last 5 characters of the primer being an exact match of gaccg.

Weighted OPC Statistics				
input	GREEDY1		GREEDY2	
sets	Weight	Cardinality	Weight	Cardinality
1	317	14	317	14
2	337	15	313	12
3	320	13	278	9
4	322	14	317	13
5	319	13	334	14
6	327	12	318	12
7	313	14	308	13
8	314	13	314	13
9	320	14	315	14
10	301	12	303	12
11	325	14	325	14
12	317	13	309	13
13	298	11	321	13
14	315	13	315	13
15	294	12	288	12
16	316	12	316	12
17	298	13	298	13
18	321	12	321	12
19	307	13	307	13
20	287	12	313	13
21	314	12	318	13
22	292	12	319	13
23	294	12	294	12
24	312	13	312	13
25	253	11	253	11
26	320	11	335	13
27	349	15	334	14
28	299	12	299	12
29	333	14	333	14
30	318	13	318	13

Table 2: Weight and cardinality statistics of the covers produced by the various algorithms on 30 random permutations of a data set consisting of biological data (56 sequences of 75 nucleotides each). Here GREEDY2 does outperform GREEDY1 on many instances.

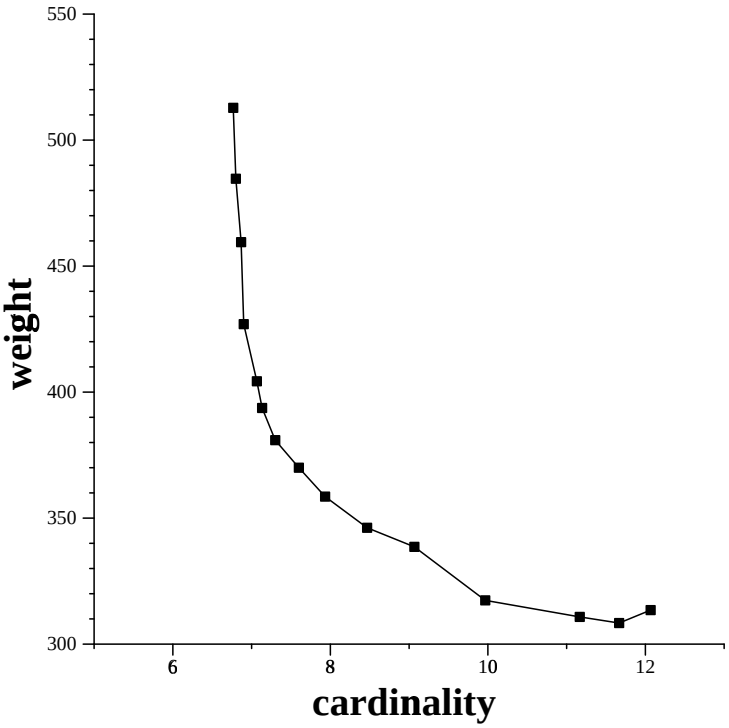

Figure 6: Average cardinality and weight over 30 data sets using GREEDY1 in a simultaneous optimization of both weight and cardinality. Different parameters are used in the cost function to achieve a smooth tradeoff between the two objectives.