On the Primer Selection Problem in Polymerase Chain Reaction Experiments

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

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. On the practical side, we give a simple branch-and-bound algorithm that solves the primers minimization problem within reasonable time for typical instances. Moreover, 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 (for a recent review, see [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]. The remarkable power of this technique and its importance for biological research was recognized with the selection of Kary Mullis, the inventor of PCR, to share the Nobel Prize in Chemistry in October, 1993.

Computer programs [8] [11] [12] 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 the nucleotide sequence for selecting a single

primer binding site in a complex mammalian genome (which contains up to $3 \cdot 10^9$ such sites) and avoiding various artifacts that can be encountered with PCR. Thus, the computer program is given a single DNA sequence, which might contain 100 potential primer sites, and the sites that optimize some relatively 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. Currently the largest known family is believed to contain as many as 1000 related genes that encode proteins called G-protein-coupled receptors [7]. However, there are many other such families that encode a large range of proteins with essential roles; PCR amplification is often the only technically feasible method for characterizing members of such large families of genes. Here the problem is quite different from the typical primer selection problem. 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, 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.

We offer both theoretical and practical contributions. On the negative side, we prove that it is computationally intractable to minimize the number of primers required to amplify a given set of sequences; in particular, we use a reduction from the set cover problem to establish that primer number minimization is $\mathcal{NP}$-complete, which implies that no polynomial-time algorithm is likely to exist for this problem. Moreover, we show that one can not even hope to solve this problem approximately very well: there does not exist an approximation algorithm that yields solutions with cost bounded by a constant times optimal, unless a longstanding open problem in computational complexity theory is resolved. On the positive side, we give a straightforward branch-and-bound algorithm that solves the primer minimization problem within reasonable time for typical instances. We also construct an efficient approximation scheme 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. Finally, we analyze a weighted variant, where both the number of primers as well as the sum of their "costs" must be minimized simultaneously. We conclude by discussing the empirical performance of our methods on biological data.

The rest of the paper is organized as follows. In Section 2 we develop notation and formulate the problem. Section 3 establishes the $\mathcal{NP}$-completeness of the primers number minimization problem. In Section 4 we present an exact branch-and-bound algorithm, and in Section 5 we develop a provably-good heuristic and analyze its performance in terms of solution quality. Section 6 introduces the weighted formulation and discusses the simultaneous optimization of solution cost as well as cardinality. In Section 7 we discuss the empirical performance of our algorithms on actual DNA data, and present our experimental results. We conclude in Section 8 with future research directions.

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 define Σ^* to be 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 a *string group* of *order* k if all the strings have a common substring of length k or more; in other words, 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 group 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 group S, and that S is the string group associated with u. The *size* of S is the number of strings in S, denoted by $|S|$. If a subset S' of S satisfies the string group definition with order k, then

we say that $S' \subseteq S$ is a *string subgroup* of S with order k, denoted $S' \sqsubseteq_k S$. A string subgroup is *maximal* if it is not a proper subset of any other string subgroup of the same order. We denote the collection of all string subgroups 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 group of order 4, since $\underline{caba}$ is also a common substring of length 4 for each string in S (we use the underline notation to highlight common substrings). Note that $S = \{cab\underline{ac}a, \underline{ac}abab, bb\underline{ac}aba\}$ is also a string group 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 group of order 5, since there exists no substring of length 5 common to all strings of S. We observe that S contains a maximal string subgroup 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 subgroups contained in $\mathcal{C} = \{\{\underline{acaba}b, bb\underline{acaba}\}\ ,\ \{\underline{caba}ca\}\}$ form an optimal cover for S of order 5 and size $|\mathcal{C}| = 2$, although the single string subgroup $\{\underline{caba}ca, a\underline{caba}b, bba\underline{caba}\}$ (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)group corresponds to a *primer group*; we shall therefore use these terms interchangeably in what follows, depending on context. Also note that 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) can synthesize to amplify (i.e., cover) different sequence subsets (i.e., string subgroups). Moreover, to keep the problem realistic, we insist that all primers have length k or more, otherwise we could 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 wasteful due to the large number of primers necessary to carry out the PCR experiment and 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 order necessary to cover a given set of DNA stands:

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

In the next section we analyze the complexity of the OPC problem and prove its computational intractability[1]. In subsequent sections we will present and analyze a provably-good heuristic for the OPC problem.

3 Complexity of the OPC Problem

In analyzing the computational complexity of combinatorial problems, one often seeks a known intractable problem that reduces to the problem at hand. If this transformation can be achieved "efficiently", then the problem at hand is intractable also. Indeed this is a fundamental technique in the theory of $\mathcal{NP}$-*completeness* [6], which defines and characterizes the class $\mathcal{NP}$ of problems that are solvable in non-deterministic polynomial time. The "hardest" problems in $\mathcal{NP}$, namely the collection of $\mathcal{NP}$-*complete* problems ($\mathcal{NPC}$), all transform (or reduce) to one another within polynomial time, yet none are currently known to be solvable within *deterministic* polynomial time. Thus, a polynomial-time solution to one of the problems in $\mathcal{NPC}$ would immediately yield polynomial-time algorithms to *all* problems in $\mathcal{NPC}$. Since $\mathcal{NPC}$ is also known to contain hundreds of well-known difficult problems that for decades have resisted polynomial-time solutions, it is thought unlikely that *any* problem in $\mathcal{NPC}$ is solvable within polynomial time. Therefore, a proof that a given problem is in $\mathcal{NPC}$ serves as strong evidence of its intractability, and justifies the use of efficient but inexact heuristic solutions (as opposed to exact but inefficient ones).

We stated the OPC problem in Section 2 as an *optimization* problem; i.e. given an instance $< S, k >$ of the OPC problem, we seek a cover for S of order k with minimum size. The theory

[1] As usual, "efficient" in this context means "solvable within time polynomial in the input size", and "intractable" implies the contrary.

of $\mathcal{NP}$-completeness usually considers the *decision* versions of problems, where the output is restricted to be simply either "yes" or "no". This convenience is undertaken without loss of generality because the optimization version of a problem is "not any easier" than the corresponding decision version (since if we knew what the optimal answer was, we could then trivially determine whether it satisfies a given size/cost bound). Thus, for a given optimization problem in $\mathcal{NP}$, if the decision version is $\mathcal{NP}$-complete, then so is the optimization version.

For the purpose of the intractability results below, we therefore recast the OPC problem as a decision problem as follows: given a finite set S of DNA sequences and integers k and l, does there exist a cover for S of order k and of size l or less? Our first theoretical result establishes the intractability of the decision version of the optimal primer cover problem; it is clear from the discussion above that the intractability of the optimization version of the OPC problem immediately follows.

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

Proof: Clearly the OPC problem is in the class $\mathcal{NP}$, since given an instance $< S, k, l >$ of OPC and a primer cover $\mathcal{C}$, we can easily verify within polynomial time that $\mathcal{C}$ has order k and cardinality l, and moreover that $\mathcal{C}$ covers S. To verify that $\mathcal{C}$ is of order k, we check that each of its constituent string sets $C_i \in \mathcal{C}$ is of order k, which in turn is accomplished by considering all substrings of length k in each sequence in C_i and verifying that at least one of them is a substring of *all* other sequences in C_i. To see if $\mathcal{C}$ covers S, we simply check whether $S \subseteq \bigcup \mathcal{C}$.

To complete the proof that OPC is $\mathcal{NP}$-complete, we must next transform a known problem in $\mathcal{NPC}$ to the OPC problem. Toward this end we choose the well-known $\mathcal{NP}$-complete *minimum set cover* (MSC) problem, which is defined as follows: 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 "seperators" 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[2] 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 and Figure 2) will clearly induce a string subgroup $S_i \sqsubseteq_k S$ corresponding to M_i, since $u_i \preceq s_j$ for all s_j corresponding to $t_j \in M_i$.

Transformation of the MSC problem to the OPC problem
Input: An arbitrary instance $< T, \mathcal{M}, h >$ of MSC **Output:** A Corresponding instance $< S, k, l >$ of OPC
1. **For** each $t_i \in T$ **Do Create** $s_i \in S$ with $s_i \leftarrow \epsilon$ 2. **For** $i = 1$ to $\lvert\mathcal{M}\rvert$ **Do** 3. **Let** u_i = a unique string of length $\lceil \log_2 \lvert\mathcal{M}\rvert \rceil$ over $\{0,1\}$ 4. **For** each $t_j \in M_i$ **Do** $s_j \leftarrow s_j \cdot u_i \cdot b_j$ 5. **Output** $< S, k, l > \leftarrow < \{s_1, \ldots, s_{\lvert T\rvert}\}, \lceil \log_2 \lvert\mathcal{M}\rvert \rceil, h >$

Figure 1: Polynomial-time transformation of an instance of MSC into an instance of OPC, using the alphabet $\Sigma = \{0, 1, b_1, b_2, \ldots, b_{|T|}\}$ for the encoding.

Although it is clear from the construction that each subset $M_i \in \mathcal{M}$ has a corresponding string subgroup S_i, it is not obvious that our construction avoids introducing maximal[3] subgroups 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 subgroups.

Assume towards contradiction that a spurious maximal string subgroup 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

[2] Observe that using the alphabet $\{0,1\}$, we can form $2^k = 2^{\lceil \log_2 |\mathcal{M}| \rceil} \geq |\mathcal{M}|$ unique strings of length k which are sufficient to uniquely encode all $|\mathcal{M}|$ subsets.

[3] Since the objective of the OPC problem is to minimize the cardinality of the cover, it suffices to consider only maximal string subgroups.

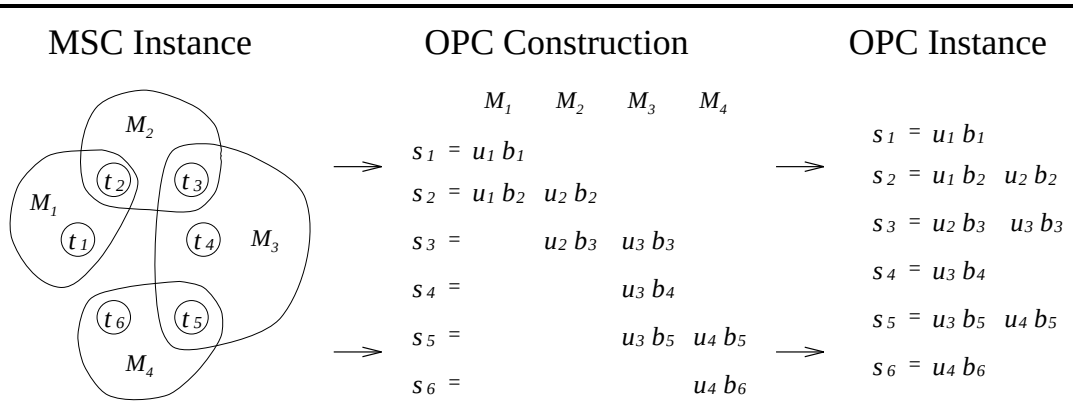

Figure 2: The construction of an instance of OPC from an arbitrary instance of MSC. A unique string u_i is used to encode each subset M_i (i.e., here the unique strings u_1, u_2, u_3, and u_4 are created to represent M_1, M_2, M_3, and M_4, respectively). Also, each string s_j uses its own unique separator b_j in between the u_i's. Note that 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 $u_3 b_5 \cdot u_4 b_5$.

only in the string s_j, and thus the size of the string subgroup S' is at most 1 (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 subgroup 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 summary, our construction puts the maximal string subgroups over S into a one-to-one correspondence with the subsets in $\mathcal{M}$; therefore, a minimum set cover in $< T, \mathcal{M}, h >$ corresponds to a minimum primer cover in $< S, k, l >$. □

Note that in Theorem 3.1, the alphabet size of the OPC instance is dependent 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. As we will see, $\{a, c\}$ and $\{g, t\}$ can be used to encode $\{0, 1\}$ and $\{b_1, b_2, \ldots, b_{|T|}\}$ of the unrestricted alphabet, respectively; we next outline a scheme that enforces a one-to-one correspondence between the

subsets $M_i \in \mathcal{M}$ and the maximal string subgroups $S_i \sqsubseteq_k S$, using only the restricted alphabet $\Sigma = \{a, c, g, t\}$ for the encoding.

Theorem 3.2 *The OPC problem with restricted alphabet $\Sigma = \{a, c, g, t\}$ is $\mathcal{NP}$-complete.*

Proof: Since this problem is a special case of the more general OPC problem that was shown to be in $\mathcal{NP}$ in Theorem 3.1, this restricted version is clearly also contained in $\mathcal{NP}$. We now show that the restricted problem is indeed $\mathcal{NP}$-complete as well, using a transformation from the minimum set cover problem.

Let $< T, \mathcal{M}, h >$ be an arbitrary instance of MSC, and set $l = h$. As before, we would like to create an instance $< S, k, l >$ of OPC that will encode elements of T as strings in S. As in the proof of Theorem 3.1, we encode subsets of $\mathcal{M}$ using a unique binary representation over $\{a, c\}$. However, due to the restricted alphabet size, we no longer have the freedom to use $|T|$ distinct symbols to separate the u_i's, but rather we must now also encode the separators b_j themselves using a binary representation over the remaining alphabet symbols $\{g, t\}$. This is indeed what we do, but the only problem is that pieces of the encodings of the u_i's may inadvertently combine with fragments of the encodings for the b_j's to form spurious primers and thereby induce unintentional matches with other strings of S.

This problem is avoided by duplicating the encoding of each $M_i \in \mathcal{M}$ *twice* in the appropriate strings in S. In other words, if the sets $M_{i_1}, M_{i_2}, \ldots, M_{i_n}$ contain an element $t_j \in T$, we construct $s_j = u_{i_1} u_{i_1} b_j u_{i_2} u_{i_2} b_j \cdots u_{i_n} u_{i_n} b_j$, where the u_i's are unique binary encodings of the M_i's, and the b_j's are unique binary encodings of the separators. We then set the order of the string subgroups that we wish to find in S to twice the length of the minimum unique encoding length, i.e., set $k = 2 \cdot \lceil \max\{\log_2 |\mathcal{M}|, \log_2 |T|\} \rceil$. The encodings of the u_i's and b_j's are all of length $k/2$.

To see that this transformation works, note that in $s_j = u_{i_1} u_{i_1} b_j u_{i_2} u_{i_2} b_j \cdots u_{i_n} u_{i_n} b_j$, any length-$k$ common substring u associated with a maximal substring group can not completely contain *any* separator b_j in its entirety, because otherwise the string subgroup induced by u will have cardinality 1 (since the separators are unique among strings). On the other hand, since the length of u is twice the length of each u_i which in turn appear as identical adjacent pairs in s_j, u must necessarily contain at least one whole copy of one of the u_i's, and therefore can not

match any other string that does not contain the same u_i. It follows that no spurious matches can occur across strings due to "unintentional" combinations of symbols from both the u_i's and the b_j's. Thus, we preserves the one-to-one correspondence between the string subgroups $S_i \sqsubseteq_k S$ and the subsets $M_i \in \mathcal{M}$, and therefore $< S, k, l >$ has a solution if and only if $< T, \mathcal{M}, h >$ does. This polynomial-time transformation implies the $\mathcal{NP}$-completeness of the OPC problem with a restricted alphabet of 4 symbols. □

4 An Exact Branch-and-Bound Algorithm

We saw above that the MSC problem reduces to the OPC problem, which indicates that there exists a similarity between the two problems. Thus if we can achieve a reduction in the opposite direction (i.e., a transformation of the OPC problem to the MSC problem), this will enable the application of well-known 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).

Recall that at 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 subgroups of the OPC instance. With this in mind, we transform the OPC problem to the MSC problem as follows: for each maximal string subgroup 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 subgroup 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 (see Figure 3). 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.

Simple Exact Algorithm for Minimal Set Cover
Input: A set T of elements, and a collection $\mathcal{M}$ of subsets $M_i \subseteq T$
Output: A collection $\mathcal{M}' \subseteq \mathcal{M}$ of minimum size such that $\mathcal{M}'$ covers T
1. **For** $i = 1$ **To** $\|\mathcal{M}\|$ **Do**
2. **For** each $\mathcal{M}' \subseteq \mathcal{M}$, $\|\mathcal{M}'\| = i$ **Do**
3. **If** $T \subseteq \bigcup \mathcal{M}'$ **Then Return** $\mathcal{M}'$

Figure 3: A simple optimal set cover algorithm which considers all $2^{|\mathcal{M}|}$ combinations of the subsets in $\mathcal{M}$ in order to find the optimal cover.

The brute-force algorithm of Figure 3 can easily be modified to incorporate a branch-and-bound optimization. First, we modify the overall structure of our algorithm to look for a maximal cover containing at most h subsets, as shown in Figure 4. By invoking the modified algorithm with all values of h, $1 \leq h \leq |\mathcal{M}|$, we still consider the entire solution space as in the naive algorithm. However, during our search, we keep track of the current best candidate solution and make use of the following fact, which enables a branch-and-bound strategy:

Fact 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 $|T'| - |T|$, so 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 $|T| - |T'|$, which implies that 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 untried 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 the running times. This scheme is formalized in Figure 4, and we discuss the empirical performance of this optimization technique 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 $\leftarrow \emptyset$
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 4: 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 therefore seek efficient heuristics that yield near-optimal solutions. Our transformation in the previous section of an arbitrary instance of OPC into an instance of MSC suggests that in our search for an efficient heuristic for the OPC problem, it suffices to address the MSC problem. Lund and Yannakakis [13] showed that no polynomial-time algorithm can approximate solutions to MSC within less than $\frac{1}{4}\log_e |T|$ times optimal, unless a longstanding open problem in complexity theory is resolved in the negative, an unlikely situation. Thus, the best polynomial-time approximation algorithm that we can hope to achieve would have a theoretical performance bound of $O(\log_e |T|)$ times optimal.

A strategy that iteratively selects a best choice among the available choices is called *greedy*.

Greedy algorithms thus make a locally optimal choice in order to approximate a globally optimal solution; they are often simple and can be implementated efficiently. In particular, one possible greedy algorithm for the MSC problem will 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 is illustrated in Figure 5; it is indeed simple and can be implemented within time $O(|\mathcal{M}| \log_2 |\mathcal{M}|)$, or with slight modifications, it can be implemented within linear time [4].

Greedy Heuristic for Minimal Set Cover
Input: A set T of elements and a set $\mathcal{M}$ of subsets of T **Output:** A set $\mathcal{M}' \subseteq \mathcal{M}$ such that $\mathcal{M}'$ covers T
1. $U \leftarrow T$ 2. $\mathcal{M}' \leftarrow \emptyset$ 3. **While** $U \neq \emptyset$ **Do** 4. **Select** an $M_i \in \mathcal{M}$ maximizing $\lvert M_i \cap U \rvert$ 5. $U \leftarrow U - M_i$ 6. $\mathcal{M}' \leftarrow \mathcal{M}' \cup \{M_i\}$ 7. **Return** $\mathcal{M}'$

Figure 5: A greedy heuristic for set cover selects at each stage the subset M_i that covers the greatest number of the remaining uncovered elements.

The performance of this greedy heuristic for the set cover problem has been analyzed extensively in the literature [9] [10] [13]. 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 [9]. A simple example where the greedy strategy computes a cover of size $(\log_2 |T|)\cdot$OPT is presented in Figure 6. 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. Although the $(\log_e |T| + 1)\cdot$OPT upper bound on the performance of the greedy heuristic is already known, we present here an argument that is considerably simpler and more concise than previously known proofs.

Let $< T, \mathcal{M}, h >$ be an arbitrary instance of minimum set cover, and define $j \leq |\mathcal{M}|$ to be the size of the optimum cover. We denote by N_i the number of elements that remain uncovered after i iterations of the greedy heuristic for MSC, so that $N_0 = |T|$.

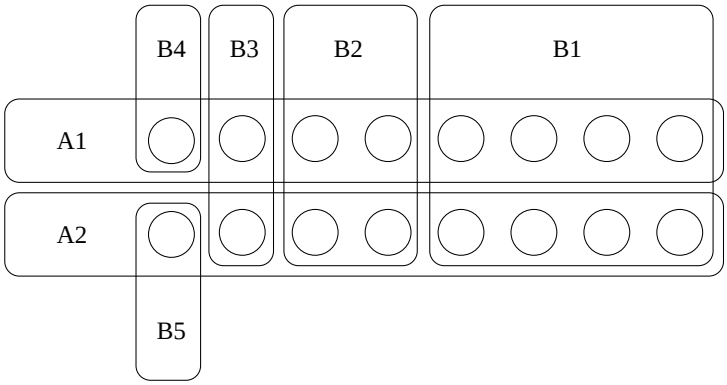

Figure 6: An example for which the greedy heuristic will produce a cover of size $(\log_2 |T|)\cdot$OPT. Here the circles represent the elements to be covered, and the $A1$, $A2$, $B1$, $B2$, $B3$, $B4$, and $B5$ ovals represent the various subsets. Observe that the optimal cover consists of $A1$ and $A2$, while the greedy heuristic may select subsets $B1$, $B2$, $B3$, $B4$, and $B5$, a logarithmic factor times optimal. Note that this example extends to an arbitrary number of elements.

Lemma 5.1 *Given N_i uncovered elements with an optimal cover of size j, an additional single iteration of the greedy heuristic will leave $N_{i+1} \leq \frac{j-1}{j} \cdot N_i$ elements uncovered.*

Proof: Given that there are j sets in the optimal cover, assume that $j' \leq j$ of them contain uncovered elements. Observe that at least one of the subsets in the optimal cover must have size at least N_i/j'. Since the greedy heuristic selects the subset of greatest size, at most $N_i - N_i/j'$ elements will be left uncovered after an additional greedy iteration step. Thus, $N_{i+1} \leq N_i - N_i/j' = \frac{j'-1}{j'} \cdot N_i$. But $j' \leq j$ implies $\frac{j'-1}{j'} \leq \frac{j-1}{j}$, which combined with the previous inequality yields $N_{i+1} \leq \frac{j-1}{j} \cdot N_i$. □

From Lemma 5.1 we obtain the following immediate corollary:

Corollary 5.2 $N_i \leq (\frac{j-1}{j})^i \cdot N_0$.

We are now ready to prove the main result regarding the performance ratio of the greedy heuristic for the MSC problem:

Theorem 5.3 *The greedy heuristic produces a cover of size at most* $\log_e |T|$ *times optimal.*

Proof: Recall that the optimal cover is of size j; we will calculate $N_{j \cdot \log_e |T|}$, the number of

elements that remain uncovered after the greedy heuristic selects $j \cdot \log_e |T|$ subsets. Corollary 5.2 implies that $N_{j \cdot \log_e |T|} \leq |T| \cdot (\frac{j-1}{j})^{j \cdot \log_e |T|} = |T| \cdot (1 - \frac{1}{j})^{j \cdot \log_e |T|}$. Using the well known fact from calculus that $(1 - \frac{1}{j})^j < \frac{1}{e}$, we see that $N_{j \cdot \log_e |T|} < |T| \cdot (\frac{1}{e})^{\log_e |T|} = 1$. Thus, $N_{j \cdot \log_e |T|} < 1$, which means that after $j \cdot \log_e |T|$ greedy iterations, all elements of T will be covered. □

6 The Weighted OPC Problem

The discussion above thus far has been restricted to address the problem of minimizing the *cardinality* of the cover - the number of primers that are required to amplify a set of DNA sequences. Thus, the algorithms in Sections 4 and 5 strive to minimize the number of string subgroups. 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 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 subgroup 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 group S_i and its associated primer u_i, find a cover $\mathcal{C}$ for S of order k, which minimizes the total weight $\sum_{S_i \in \mathcal{C}} W(S_i, u_i)$.

We first analyze the complexity of the WOPC problem. Given that the OPC problem is $\mathcal{NP}$-complete, it is not surprising that the more general WOPC is also $\mathcal{NP}$-complete:

Theorem 6.1 *The WOPC problem is $\mathcal{NP}$-complete.*

Proof: We establish that the WOPC problem is $\mathcal{NP}$-complete using a reduction from the OPC problem, which was shown to be $\mathcal{NP}$-complete in Section 3. As discussed above, we utilize the decision version of WOPC problem, namely the one which asks: "given a finite set S of DNA sequences, an integer k, a weight function W for string subgroups and their associated primers, and a real number w, does there exist a cover for S of order k and total weight $\leq w$?" We transform an arbitrary instance $< S, k, l >$ of OPC into an instance $< \hat{S}, \hat{k}, W, w >$ by setting $\hat{S} = S$, $\hat{k} = k$, and $w = l$, and also defining W to be the constant function $W(S_i, u_i) = 1$ for all string subgroups S_i and their associated primers u_i. This guarantees that the weight of a cover $\mathcal{C}$ will be equal to its cardinality $|\mathcal{C}|$, and thus $\sum_{S_i \in \mathcal{C}} W(S_i, u_i) \leq w$ holds exactly when $|\mathcal{C}| \leq l$ holds. It follows that a polynomial-time algorithm for OPC would induce a polynomial-time algorithm for WOPC, which establishes the $\mathcal{NP}$-completeness of the WOPC problem. □

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: 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 also well-studied, and we can therefore use known techniques developed for the WMSC problem in solving the WOPC problem [2] [3]. 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 (from Figure 5) and the weighted variant of the heuristic lies in 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; this is also the selection criteria that Chvatal analyzed in [3]). Thus, the extension of the unweighted approximation algorithm to a weighted approximation algorithm is also quite straightforward.

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 subgroups having total weight 0. Although under our formulation above this solution would be considered "optimal" (since it has 0 weight), this is not particularly useful. It would therefore be more interesting to pursue an algorithm that simultaneously minimizes both the weight and the number of string subgroups in a cover. Unfortunately, we can show that there does not exist an algorithm that can simultaneously minimize both the weight and cardinality of a cover with

provable non-trivial bounds:

Theorem 6.2 *There does not exist any approximation scheme for the WMSC problem that can simultaneously minimize both the weight and cardinality of a cover within any nontrivial bounds with respect to the optimal values.*

Proof: We show that there does not exist an approximation algorithm with nontrivial simultaneous bounds because there exist instances of WMSC for which there is no smooth tradeoff between cardinality and weight. For example, Figure 7 gives an instance where any cover will either have infinite weight (clearly a bad bound) or cardinality $|T|$ (which is the worst possible). Thus, in the general case we can not hope to prove theoretical bounds for any simultaneous approximation algorithm. □

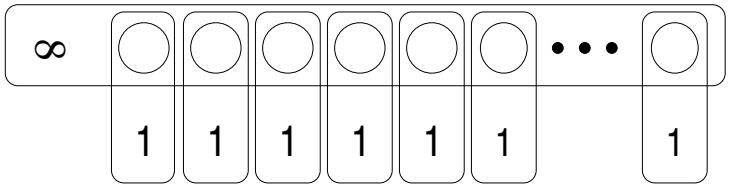

Figure 7: An instance of WMSC illustrating that no algorithm can achieve nontrivial simultaneous bounds on both weight and cardinality of a cover. The circles denote the elements to be covered, while the ovals denote the weighted subsets. Observe that the optimal cardinality of a cover is 1, while the 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 even achieve a smooth tradeoff between these two objectives for typical instances (this does not contradict Theorem 6.2, which states that no simultaneous theoretical performance bounds can be guaranteed in the worst case). For example, we can easily construct a new cost function W' that considers both the cardinality and weight of a string subgroup 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 Section

7 (i.e., this algorithm simultaneously minimizes both cardinality and weight empirically, but not within any provable simultaneous bounds).

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 (see Figure 4), the greedy (GREEDY1) heuristic (see Figure 5), 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 one of the transmembrane domains (TM3) from 56 G-protein coupled receptors [7]; the data itself is shown in Figure 8. We have 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, dependent 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. Recall that 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. Note that GREEDY1 and GREEDY2 produced an optimal cover for 21 out of the 30 random permutations, and for the remaining permutations the solutions produced by GREEDY1 and GREEDY2 are at most 1 primer off of optimal. We conclude that the heuristics are thus quite effective in primer number minimization. For the unweighted case GREEDY2 often did not perform as well as GREEDY1, so the additional complexity of GREEDY2 is not justified. In the weighted case, GREEDY2 does outperform GREEDY1 on many instances.

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 does outperform GREEDY1 on many 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 9. 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.

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; and (3) exploring additional heuristics for simultaneous tradeoffs between subgroup cardinality and weight.

9 Acknowledgement

This research was supported by a grant (LM04961) from the National Library of Medicine.

Figure 8: 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 group containing 24 sequences, with the last 5 characters of the primer being an exact match of gaccg.

Unweighted OPC Statistics			
input sets	BBOPT Card.	GREEDY1 Card.	GREEDY2 Card.
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.

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.

Weighted OPC Statistics				
input	GREEDY1		GREEDY2	
sets	Weight	Card.	Weight	Card.
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.

[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 (1972), pp. 233–235.

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

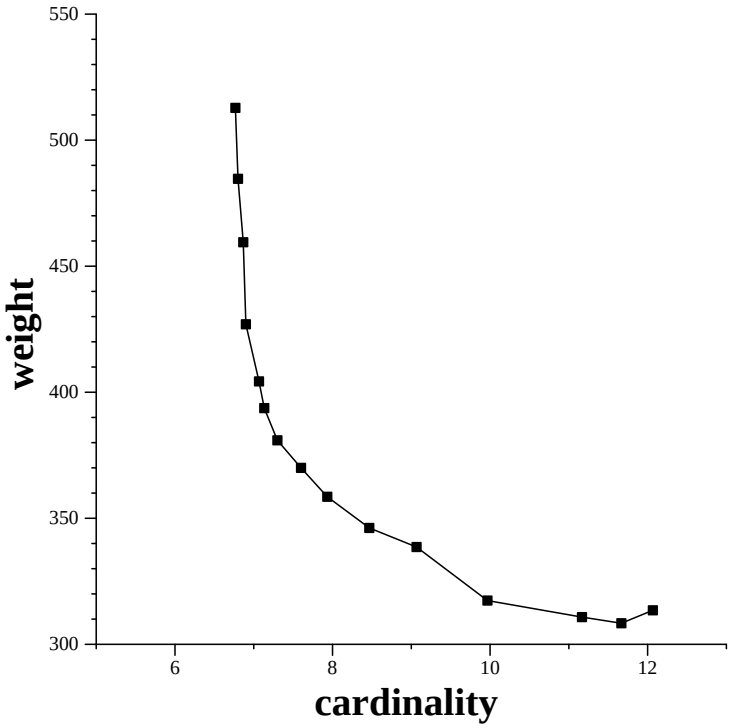

Figure 9: Average cardinality and weight over 30 data sets using GREEDY1 in a simultaneous optimization of both weight and cardinality. Different parameters are used in the cost function to achieve a smooth tradeoff between the two objectives (e.g., the two points $(7, 430)$ and $(11, 315)$ indicate that improved cardinality is achieved at the expense of higher cover weight).

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

[6] M. R. Garey and D. S. Johnson, *Computers and Intractability: a Guide to the Theory of NP Completeness*, W. H. Freeman, San Francisco, 1979.

[7] 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.

[8] 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.

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

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

[11] 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.

[12] 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.

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