A New Approach to Primer Selection 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 22908

Abstract

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, and show that even approximating solutions to this problem to within a constant factor times optimal is intractable. On the practical side, we give a simple branch-and-bound algorithm that solves the primers minimization problem within reasonable time for typical instances. We present an efficient approximation scheme for this problem, and prove that our heuristic always produces solutions no worse than a logarithmic factor times the optimal, this being the best approximation possible within polynomial time. 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 presenting 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]. 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 one of the larger families contains as many as 1000 related genes that encode proteins called G-protein-coupled receptors [7]. 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 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 wish to identify 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 demonstrate that minimizing the number of primers needed is computationally intractable; 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. On the positive side, we give a straightforward branch-and-bound algorithm that solves the primer minimization problem within reasonable time for practical instances. We also construct an efficient approximation scheme for this problem, and prove that

[*]This research was supported in part by a National Library of Medicine grant LM04961 (Pearson) and by a National Science Foundation Young Investigator Award MIP–9457412 (Robins). Corresponding author is Professor Gabriel Robins, Department of Computer Science, University of Virginia, Charlottesville, VA 22903-2442, USA, Phone: (804) 982-2207, Email: robins@cs.virginia.edu, WWW URL http://uvacs.cs.virginia.edu/~robins/.

our heuristic always produces solutions that are guaranteed to have bounded cost with respect to optimal; moreover we show that our heuristic is the best possible within polynomial time. 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.

2 Notation and Problem Formulation

Before we formulate the problem of minimizing the number of primers required to synthesize from a given set of DNA sequences, we first develop the necessary notation. We use 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[1] Σ. 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 (i.e., 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 $\bigcup \mathcal{C} = \bigcup\limits_{C \in \mathcal{C}} C$.

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 is a substring of 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 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. A string subgroup is *maximal* if it is not a proper subset of any other string subgroup of the same order. If for some collection $\mathcal{C}$ of subsets of S we have $\bigcup \mathcal{C} = S$, 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 and size 3, 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 $S = \{cab\underline{ac}a, \underline{ac}abab, bb\underline{ac}aba\}$ is also a string group of size 3 with 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 $\mathcal{C} = \{\{\underline{acaba}b, bb\underline{acaba}\}, \{\underline{cabaca}\}\}$ 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*, and a string (sub)group corresponds to a primer group or a portion thereof; we shall therefore use these terms interchangeably in what follows, depending on context.

3 The Optimal Primer Cover Problem

Given a set of DNA sequences (strings), there are many choices as to which primers (i.e., common substrings) one can synthesize (i.e., cover) for 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 given order 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.

In addressing combinatorial problems, we seek efficient (i.e., polynomial-time) algorithms. Unfortunately, we can show that the OPC problem is $\mathcal{NP}$-complete, which serves as strong evidence of its intractability, and justifies the use of heuristic solutions (as opposed to exact ones) [6]. We establish the intractability of the OPC problem by transforming it to the well-known $\mathcal{NP}$-

[1] 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.

complete *minimum set cover* (MSC) problem, which is defined as follows: given a collection $\mathcal{M}$ of subsets of a finite set T, find in $\mathcal{M}$ a minimum-size cover for T (i.e., find a cover $\mathcal{M}' \subseteq \mathcal{M}$ with minimum $|\mathcal{M}'|$ such that $\bigcup \mathcal{M}' = T$). Our reduction of MSC to OPC is based on establishing a one-to-one correspondence between the subsets in $\mathcal{M}$ and the maximal string subgroups over S, using unique substrings to encode subset membership of the various elements of T. The full details of this transformation may be found in [14], and are omitted here for brevity.

4 Exact Branch-and-Bound Algorithm

In this section we outline a branch-and-bound exact algorithm for the OPC problem (the next section will outline a provably-good and more efficient heuristic). Since we can transform the OPC problem to the MSC problem, we are able to apply techniques for the MSC problem in order to solve the OPC problem. In particular, given an instance of the OPC problem with sequence set S and order k, for each string $s_i \in S$ we find all length-k substrings s_i', and for each one of these s_i' we form the maximal string subgroup in S associated with s_i'; these become the subsets of our corresponding MSC instance. This implies that a good solution to the resulting MSC instance would constitute a good solution to the original OPC instance. With this transformation in mind, we couch the rest of our discussion using the terminology of the minimum set cover problem.

One straightforward scheme to solve the OPC 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 algorithm runs in time exponential in the number of subsets $\mathcal{M}$. We can improve the performance of the 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 be able to 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.

The brute-force algorithm 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. By invoking this modified algorithm with all values of h, $1 \le h \le |\mathcal{M}|$, 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 lemma, which enables an effective branch-and-bound strategy:

Lemma 4.1 *Consider an instance $< T, \mathcal{M}, h >$ of OPC, 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 $b = \max\limits_{M_i \in \mathcal{M} - \mathcal{M}'} |M_i|$. Then $\mathcal{M}'$ can not be "extended" by m additional subsets into a cover for T of size $|\mathcal{M}'| + m$, unless $|T'| + m \cdot b \ge |T|$.*

Proof: (omitted for brevity)

Based on this lemma, we can avoid trying to augment partial covers if there are no remaining untried subsets that 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 (see Figure 1 for a formal description).

```
Exact Algorithm for Optimal Primer Cover
Input:   Set $T$ of sequences, a set $\mathcal{M}$ of subsets $M_i \subseteq T$,
         and integer $h$.
Output:  A collection $\mathcal{M}' \subseteq \mathcal{M}$, $|\mathcal{M}'| = h$,
         such that $|\bigcup \mathcal{M}'|$ is maximum.

 1. Procedure Optimal_Algorithm ($T$, $\mathcal{M}$, $h$)
 2.     Sort $\mathcal{M} = \{M_1, \ldots, M_{|\mathcal{M}|}\}$
                by non-increasing cardinality of $M_i$
 3.     OPT $\leftarrow \emptyset$
 4.     Try_Primer(OPT, $h$, 1)
 5.     Return OPT

 6. Procedure Try_Primer ($\mathcal{M}'$,left,next)
 7.     If $|\bigcup \mathcal{M}'| \ge |\text{OPT}|$ Then OPT $\leftarrow \mathcal{M}'$
 8      If left$= 0$ Then Return
 9.     For $i$ =next to $|\mathcal{M}|$ Do
10.         If $|\bigcup \mathcal{M}'|$+left$\cdot|\mathcal{M}_i| \ge |\text{OPT}|$
                Then Try_Primer ($\mathcal{M}' \cup \{M_i\}$,left$-1, i+1$)
```

Figure 1: An exact algorithm for the OPC problem, based on set covering. Branch-and-bound is used to speed up the search: out of all $\binom{|\mathcal{M}|}{h}$ possible covers, the one that covers the greatest number of elements of T is returned.

5 An Efficient 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. Based on recent results by Lund and Yannakakis [13], we can prove that that no polynomial-time approximation heuristic is likely to solve the OPC problem to within less than a factor of $\frac{1}{4} \log |T|$ 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; below we show how this theoretical optimum can actually be achieved using a simple heuristic.

A strategy that iteratively selects the 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 implemented efficiently. In particular, one possible greedy algorithm for the OPC problem will select a 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 2; it can be implemented within time $O(|\mathcal{M}| \log |\mathcal{M}|)$, or, with slight modifications, it can be implemented within linear time [4].

```
Greedy Heuristic for Optimal Primer 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 $|M_i \cap U|$
5.     $U \leftarrow U - M_i$
6.     $\mathcal{M}' \leftarrow \mathcal{M}' \cup M_i$
7. Return $\mathcal{M}'$
```

Figure 2: A greedy heuristic for the OPC problem, based on set covering. At each stage we select a subset M_i that covers the greatest number of the remaining uncovered elements.

The performance of the greedy heuristic 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]. 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 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.

Theorem 5.1 *The greedy heuristic produces a cover of size at most* $\log_e |T|$ *times optimal.*

Proof: Let $< T, \mathcal{M}, h >$ be an arbitrary instance of OPC, 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 OPC, with $N_0 = |T|$. Now focus on some iteration of the greedy algorithm, where $j' \leq j$ of the subsets 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 $N_{i+1} \leq \frac{j-1}{j} \cdot N_i$. Thus, 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, and it follows that $N_i \leq (\frac{j-1}{j})^i \cdot N_0$.

Consider $N_{j \cdot \log_e |T|}$, the number of elements that remain uncovered after the greedy heuristic selects $j \cdot \log_e |T|$ subsets. We know 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. It follows that the greedy heuristic produces a cover of size at most $\log_e |T|$ times optimal. □

6 The Weighted OPC Problem

The discussion 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 from 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 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 $\sum_{S_i \in \mathcal{C}} W(S_i, u_i)$.

Given that the OPC problem is $\mathcal{NP}$-complete, it is not surprising that the more general WOPC is also $\mathcal{NP}$-complete. This is established by setting all the subset costs in the weighted version to 1, which will guarantee that the weight of a cover will be equal to its size.

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.

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 OPC problem, it is not surprising that a greedy heuristic for the WOPC problem also has a worst-case performance bound of $(\log_e |T| + 1) \cdot \mathrm{OPT}$. The only difference between the unweighted greedy heuristic (from Figure 2) 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[2]). The extension of the unweighted approximation algorithm for OPC to a weighted approximation algorithm is 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, 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 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; this is proved by exhibiting an instance of WOPC where any cover will either have the worst possible weight or else the worst possible cardinality [14].

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 our result 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 pro-

[2]This weighting criterion and its performance with respect to the weighted MSC problem are discussed in [2]; a heuristic for the unweighted MSC problem is analyzed in [3].

gramming 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 1), the greedy (GREEDY1) heuristic (see Figure 2), 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]. 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 an hour to several days, dependent upon the size of the optimal cover.

Both 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.

Though 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 3. Each data point represents the average values over the 30 runs on the random permutation for selected values of t. As expected, we observe a smooth tradeoff between cover cardinality and weight.

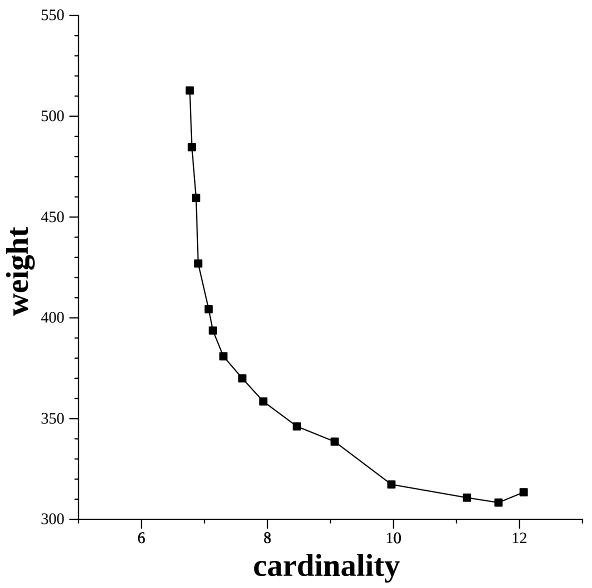

Figure 3: Average cardinality and weight over 30 data sets using GREEDY1 in a simultaneous optimization of both weight and cardinality. Different parameters are used in the cost function to achieve a smooth tradeoff between the two objectives (e.g., the two points (7, 430) and (11, 315) indicate that improved cardinality is achieved at the expense of higher cover weight).

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 to 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 off of the optimal cost, and that this is the best possible within polynomial time. Finally, we analyzed a weighted variant, where both the number of primers as well as the sum of their "costs" are to be optimized simultaneously. Our algorithms are easy to implement and produce modest numbers of primers on biological data.

For the approach to work even more effectively on biological data, more sophisticated weighting schemes are required. 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.

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 (1972), 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] 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.

[14] G. Robins, D. E. Wrege, T. Zhang, and W. R. Pearson, *On the Primer Selection Problem in Polymerase Chain Reaction Experiments*, Tech. Rep. CS-93-68, Department of Computer Science, University of Virginia, November 1993.