Publication I

Patric R. J. Östergård, Vesa Vaskelainen, and Axel Mosig. 2007. Algorithms for the combinatorial best barbeque problem. MATCH Communications in Mathematical and in Computer Chemistry, volume 58, number 2, pages 309-321.

MATCH
Communications in Mathematical and in Computer Chemistry

MATCH Commun. Math. Comput. Chem. **58** (2007) 309-321

ISSN 0340 - 6253

Algorithms for the Combinatorial Best Barbeque Problem

Patric R. J. Östergård[*] and Vesa Vaskelainen[†]
Department of Electrical and Communications Engineering
Helsinki University of Technology
P.O. Box 3000
02015 TKK, Finland
Email: patric.ostergard@tkk.fi,vvaskela@cc.hut.fi

Axel Mosig
Lehrstuhl für Bioinformatik
Universität Leipzig
Haertelstrasse 16–18
04107 Leipzig, Germany
Email: axel@bioinf.uni-leipzig.de

(Received June 22, 2006)

Abstract

The best barbeque problem asks for the largest intersection of n sets that are taken one from each of n given collections of subsets of some universal set. This combinatorial optimization problem arises in the context of discovering so-called cis-regulatory modules in regulatory DNA sequences. There are some similarities between this problem and the problem of finding cliques in graphs. These similarities are here utilized to develop novel branch-and-bound algorithms for the best barbeque problem. The algorithms are evaluated using random data. Compared with previously used algorithms, the new algorithms are capable of solving substantially larger instances, which are crucial for some application scenarios.

[*] Supported in part by the Academy of Finland under Grants No. 100500 and No. 107493.

[†] Supported by the Academy of Finland under Grant No. 107493 and by the Foundation of Technology (Tekniikan edistämissäätiö).

1 Introduction

In a recent study of phylogenetic footprints in genes [4], the third author together with Bıyıkoğlu, Prohaska, and Stadler observe that one approach of detecting such footprints leads to instances of a certain combinatorial optimization problem, which they term the *best barbeque problem.* The best barbeque problem is a rather generic problem, and could well have other applications than those mentioned in [4]. The generic nature of the problem also makes it interesting from a discrete mathematics point of view. The best barbeque problem can be formulated as follows.

Combinatorial Best Barbeque Problem (CBBQ). For positive integers m and n, let $\mathcal{C}_i$, $1 \le i \le n$ be collections of subsets of $\{1, 2, \ldots, m\}$ and denote the subsets in a collection $\mathcal{C}_i$ by $C_{i,j}$, where $1 \le j \le |\mathcal{C}_i|$. Determine the maximum value of

$$\left| \bigcap_{i=1}^{n} C_{i,\nu(i)} \right|, \tag{1}$$

where $\nu : \{1, \ldots, n\} \rightarrow \mathbb{Z}^+$ with $\nu(i) \le |\mathcal{C}_i|$ and optimization is done over all such ν.

Example 1. Let $m = 4$, $n = 2$, $C_{1,1} = \{1\}$, $C_{1,2} = \{2, 3, 4\}$, $C_{2,1} = \{1, 2, 3\}$, $C_{2,2} = \{1, 3, 4\}$. Then $|C_{1,2} \cap C_{2,1}| = |\{2, 3\}| = 2$ (and also $|C_{1,2} \cap C_{2,2}| = |\{3, 4\}| = 2$) and it is easily seen that 2 is the maximum value.

Finding solutions to instances of the combinatorial best barbeque problem is computationally a hard problem. More precisely, the decision version of CBBQ—determining whether the solution is greater than or equal to a given integer k—is NP-complete [4]. Consequently, we can only expect rather small instances to be solvable using a reasonable amount of CPU time. Such exact algorithms are indeed considered in this paper. Other lines of research would be to consider stochastic algorithms for obtaining lower bounds for the solutions or approximation algorithms.

Exact algorithms for the combinatorial best barbeque problem are developed in Section 2 and evaluated—using random data—in Section 3.

2 Algorithms

The algorithms presented in this paper are backtrack algorithms [3, Ch. 4]. A backtrack algorithm is a recursive method of building up feasible solutions to a combinatorial optimization problem one step at a time. Due to its exhaustive nature, a backtrack algorithm will always find an optimal solution. There are at least two general approaches for developing a backtrack algorithm for CBBQ. Either one may focus on the elements that are to occur in the intersection, or on the values of ν_i (determining what subsets of the collections are chosen), and build up a solution accordingly. Some naive ideas related to these approaches are presented in [4], but no further study of (efficient) algorithms is carried out in that paper. In this study we develop algorithms of both types, starting with algorithms building up a desired intersection one element at a time.

Given the collections $\mathcal{C}_i$ and an element s that is to occur in the desired intersection—the element s must occur in some of the sets in $\mathcal{C}_i$ for all $1 \leq i \leq n$—one arrives at a new instance with collections $\mathcal{C}'_i$, $1 \leq i \leq n$, where

$$\mathcal{C}'_i = \{C \setminus \{s\} : s \in C \in \mathcal{C}_i\}.$$

This leads to a complete backtrack algorithm after deciding procedures for choosing the next element s and—for improved performance—for pruning the search. One possibility for pruning is provided by the following formula that gives an upper bound on the number of elements that can be added:

$$\min_{1 \leq i \leq n} \max_{1 \leq j \leq |\mathcal{C}_i|} |C_{i,j}|. \tag{2}$$

One can refine this idea even further to get the upper bound

$$\max_{1 \leq k \leq m} \min_{1 \leq i \leq n} \max_{1 \leq j \leq |\mathcal{C}_i|} \{|C_{i,j}| : k \in C_{i,j}\}. \tag{3}$$

However, determining (3) is computationally harder than determining (2). It turns out that utilizing (3) rather than (2) in the algorithms to be presented does not give an improvement of the overall performance.

A basic backtrack algorithm utilizing these bounds is presented as Algorithm 1, which is invoked with $cbbq(\mathcal{C}, 0)$. The size of the largest solution encountered is given by the global variable *record*, whose initial value is 0, and the solution is saved in the global variables x_i. Note that in describing the algorithm, our main focus is on its general principles, so to optimize the performance of the algorithm it should obviously not be implemented verbatim but some nontrivial but yet standard data structures and techniques should be employed—see, for example, [3].

Algorithm 1 Backtrack algorithm for CBBQ

```
procedure $cbbq(\mathcal{C}$: collection of collection of sets, $l$: integer)
 1: $S = \cap_{i=1}^{n} \cup_{j=1}^{|\mathcal{C}_i|} C_{i,j}$; The elements of $S$ are denoted by $s_1, s_2, \ldots$.
 2: if $|S| = 0$ then
 3:    if $l > record$ then
 4:       $record = l$
 5:       Save the current solution $x_1, x_2, \ldots, x_l$
 6:    end if
 7:    return
 8: end if
 9: for $c = 1, 2, \ldots, m$ do
10:    if $c \notin S$ then
11:       Delete $c$ from all sets $C_{i,j}$
12:    end if
13: end for
14: for $c = 1, 2, \ldots, |S| - record + l$ do
15:    for $i = 1, 2, \ldots, n$ do
16:       $\mathcal{C}'_i = \{C \setminus \{s_1, s_2, \ldots, s_c\} : s_c \in C \in \mathcal{C}_i\}$
17:    end for
18:    $t = \min_{1 \le i \le n} \max_{1 \le j \le |\mathcal{C}'_i|} |C'_{i,j}|$
19:    if $l + 1 + t > record$ then
20:       $x_{l+1} = s_c$
21:       $cbbq(\mathcal{C}', l + 1)$
22:    end if
23: end for
24: return
end procedure
```

The following example illustrates Algorithm 1.

Example 2. Let $m = 7$, $n = 2$, $\mathcal{C}_1 = \{\{1, 2, 5, 6\}, \{2, 3, 5, 6\}, \{1, 4, 6\}, \{1, 2, 3, 5\}\}$ and $\mathcal{C}_2 = \{\{2, 3, 6\}, \{3, 4, 6\}, \{1, 2, 5, 7\}, \{1, 2, 3, 4, 6\}\}$. The execution of $cbbq(\mathcal{C}, 0)$ produces the search tree in Figure 1. The operation carried out in lines 9 to 13 of the algorithm is shown by a directed edge. The branches that are outside the loop in line 14 are indicated by one crossing line, and the pruning carried out in line 19 is indicated by two crossing lines. The execution terminates with the solution $\{1, 2, 3\}$ of cardinality 3.

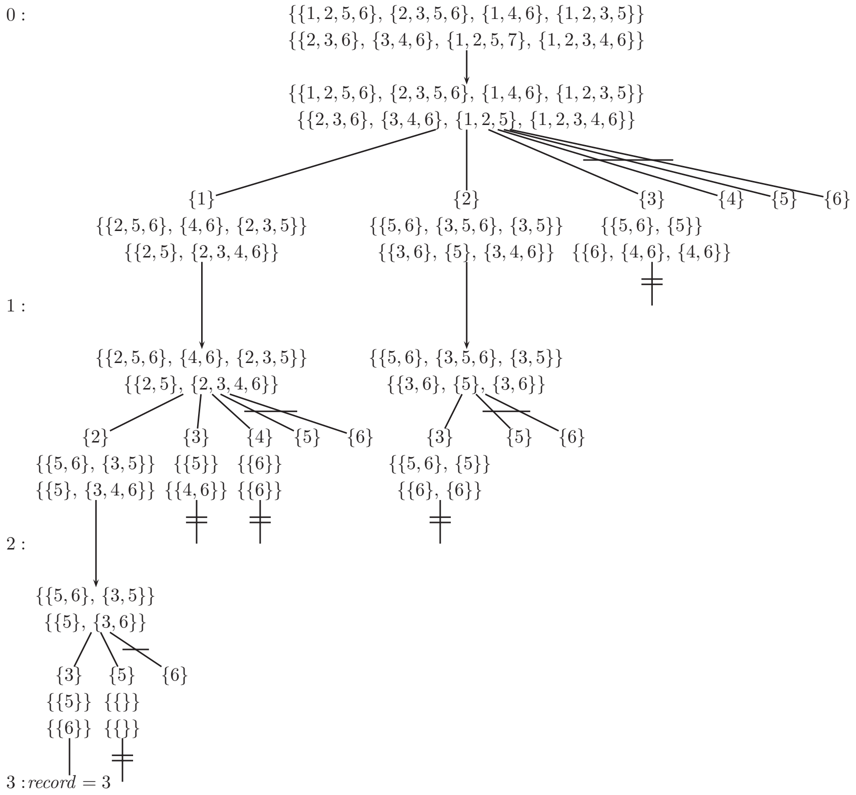

Figure 1: Search tree for Algorithm 1

As the reductions in [4] suggest, the best barbecue problem is somewhat related to the maximum clique problem. In fact, the principle ideas underlying Algorithm 1 are motivated from and work analogously to the backtracking algorithm for the maximum clique problem presented in [1]. Note, however, that CBBQ has more parameters than the maximum clique problem. An important trait of maximum clique algorithms is that the order in which the vertices of the graph are considered significantly affects the computation times; consequently, reordering the input also has an impact on our algorithms for CBBQ.

Due to the analogy of Algorithm 1 with the maximum clique algorithm presented in [1], it seems plausible that ideas from other types of maximum clique algorithms could also be utilized to develop efficient algorithms for the current problem.

Algorithm 1 considers the elements in ascending order. It is also possible to develop a so-called Russian doll search algorithm [6] for this problem; this is done for the maximum clique problem in [5]. A Russian doll search algorithm for a problem with n variables proceeds by solving n subproblems, where the ith subproblem considers the last i variables. The solutions of the subproblems then give additional possibilities for pruning.

A Russian doll search algorithm for CBBQ is presented as Algorithm 2 and is invoked with *russian_doll*$(\mathcal{C})$. Subproblem i considers only elements in the set $\{m-i+1, m-i+2, \ldots, m\}$ and the largest barbeque with respect to those element is recorded in the global array $b[i]$. The possibility for pruning provided by $b[i]$ is implemented in line 16 of the algorithm. Specifically, if we search for a barbeque of size greater than *record*, then we can prune the search if we consider i to be the next element in the barbeque and the difference between the largest encountered solution and the size of the current partial solution is greater than or equal to $b[i]$. The same variables are used as in Algorithm 1. The boolean variable *found* is used to render an early return from the routine possible; namely, $b[i+1] = b[i]$ or $b[i+1] = b[i]+1$, so whenever a solution of size $b[i]+1$ is found we can immediately return. Algorithm 2 leads to the search tree in Figure 2 when applied to the instance in Example 2. In Figure 2 the new pruning in line 16 of Algorithm 2 is indicated by three crossing lines.

Algorithm 3 focuses on the values of ν_i and is the simplest of these three presented algorithms. Compared with the previous two algorithms this algorithm does not consider elements at all. The variable l indicates the collection rather than the element that we are examining. The current record can be updated only on the last level, where $l = n+1$. The global variable *record* has initial value 0 and the algorithm is invoked with *cbbq2*$(\mathcal{C}, 1)$. Algorithm 3 leads to the search tree in Figure 3 when applied to the instance in Example 2. In Figure 3 pruning carried out in line 9 is indicated by one crossing line, and pruning in line 14 is indicated by two crossing lines.

Algorithm 2 Russian doll search algorithm for CBBQ

```
procedure $cbq(\mathcal{C}$: collection of collection of sets, $l$: integer)
$S = \cap_{i=1}^{n} \cup_{j=1}^{|\mathcal{C}_i|} C_{i,j}$; The elements of $S$ are denoted by $s_1, s_2, \ldots$.
if $|S| = 0$ then
    if $l > record$ then
        $record = l$
        $found = true$
        Save the current solution $x_1, x_2, \ldots, x_l$
    end if
    return
end if
for $c = 1, 2, \ldots, m$ do
    if $c \notin S$ then
        Delete $c$ from all sets $C_{i,j}$
    end if
end for
for $c = 1, 2, \ldots, |S| - record + l$ do
    if $l + b[s_c] \leq record$ then
        return
    end if
    for $i = 1, 2, \ldots, n$ do
        $\mathcal{C}'_i = \{C \setminus \{s_1, s_2, \ldots, s_c\} : s_c \in C \in \mathcal{C}_i\}$
    end for
    $t = \min_{1 \leq i \leq n} \max_{1 \leq j \leq |\mathcal{C}'_i|} |C'_{i,j}|$
    if $l + 1 + t > record$ then
        $x_{l+1} = s_c$
        $cbq(\mathcal{C}', l + 1)$
    end if
    if $found = true$ then
        return
    end if
end for
return
end procedure
```

Algorithm 2 (cont.) Russian doll search algorithm for CBBQ

```
procedure russian_doll($\mathcal{C}$: collection of collection of sets)
  $record = 0$
  Preprocess the instance so that $\cap_{i=1}^{n} \cup_{j=1}^{|\mathcal{C}_i|} C_{i,j} = \{1, 2, \ldots, m\} = S$, and delete all other elements
  for $i = m, m-1, \ldots, 1$ do
      $found = false$
      for $j = 1, 2, \ldots, n$ do
          $\mathcal{C}'_j = \{C \cap \{i+1, \ldots, m\} : i \in C \in \mathcal{C}_j\}$
      end for
      $x_1 = i$
      $cbq(\mathcal{C}', 1)$
      $b[i] = record$
  end for
  return
end procedure
```

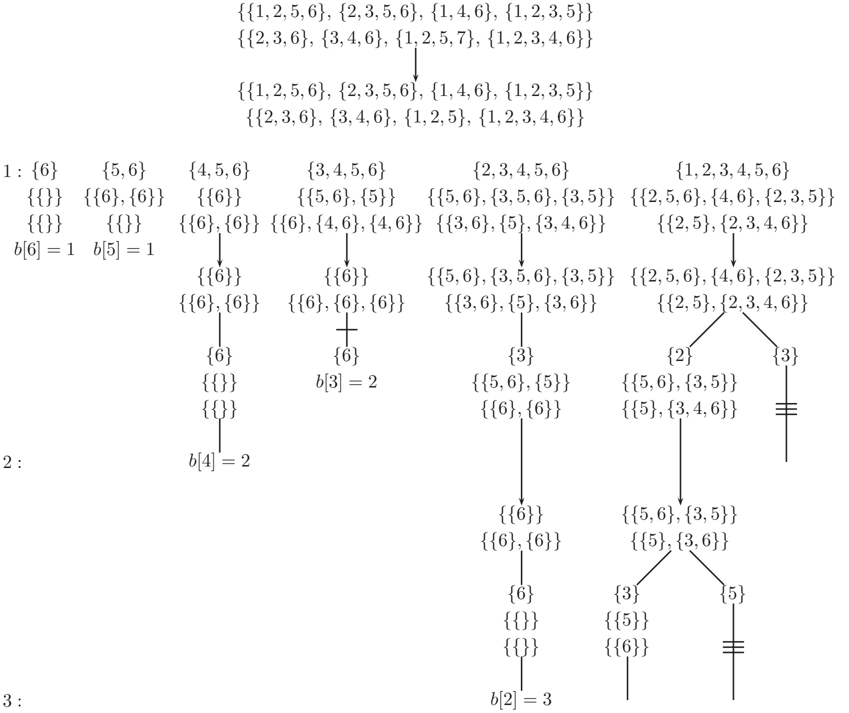

Figure 2: Search tree for Algorithm 2

Algorithm 3 Backtrack algorithm for CBBQ focusing on subsets

```
procedure cbbq2($\mathcal{C}$: collection of collection of sets, $l$: integer)
if $l = n + 1$ then
    if $|\cap_{i=1}^{n} C_{i,x_i}| > record$ then
        $record = |\cap_{i=1}^{n} C_{i,x_i}|$
        Save the current solution $\cap_{i=1}^{n} C_{i,x_i}$
    end if
    return
end if
for $j = 1, 2, \ldots, |\mathcal{C}_l|$ do
    if $|C_{l,j}| > record$ then
        for $i = l + 1, l + 2, \ldots, n$ do
            $\mathcal{C}'_i = \{C \cap C_{l,j} : C \in \mathcal{C}_i\}$
        end for
        $t = \min_{l+1 \le i \le n} \max_{1 \le j \le |\mathcal{C}'_i|} |C'_{i,j}|$
        if $t > record$ then
            $x_l = j$
            cbbq2($\mathcal{C}, l + 1$)
        end if
    end if
end for
end procedure
```

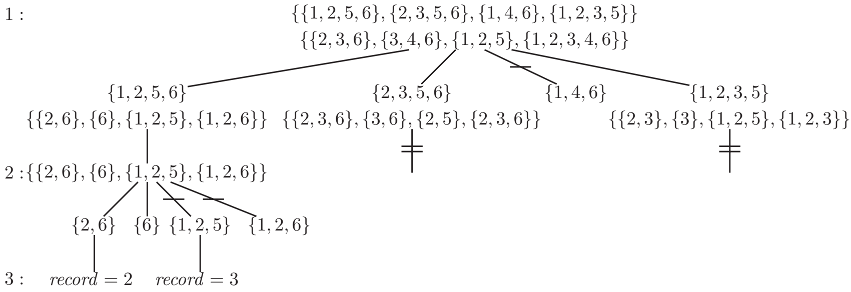

Figure 3: Search tree for Algorithm 3

3 Evaluation of the Algorithms

The performance of the presented algorithms was evaluated using randomly generated data. The results are presented in Tables 1–4. The computation times are in CPU seconds on a 1.4-GHz PC with Linux operating system. Algorithm 2 is usually the fastest for all types of data but if n is small then Algorithm 3 is the best choice.

The running times of our three algorithms are affected significantly by the order in which the input is given. Obviously, the input can be shuffled in several ways while leaving the instance of the best barbecue problem unchanged. First of all, the elements of the universal set can be *permuted* arbitrarily. Secondly, we can change the order of the collections as well as the order of the sets within each collection. We will refer to changing those orders as *sorting*.

In order to study the influence of shuffling the elements in the universal set, we tested several different permutation heuristics. The following permutation led to the best performance with Algorithm 1. We first determine how often each element occurs in the data. Then, the smallest element is relabelled to a least frequent element, the second smallest element is renamed to the element which is the next least frequent, and so on. This ordering is also good for Algorithm 2 and is used in the evaluations. Occasionally some other orderings perform better; for Algorithm 2 more research could be carried out to further optimize the performance of this algorithm for various type of data.

With Algorithm 3, as a heuristic for sorting, we order the sets within each collection descending with respect to their cardinalities, while the collections themselves are ordered descending with respect to the cardinality of the largest subset in them. If all subsets have the same size, as in our experiments with random data, this sorting heuristic obviously does nothing.

Since the instances have many parameters, we have divided the evaluation into four cases, in each of which all but one of the values n, $|\mathcal{C}_i|$, $|C_{i,j}|$ and m are fixed. For each entry, ten random instances were considered and the average solution and computation time is listed for the three algorithms (A1, A2 and A3).

Table 1: Computational results for $|\mathcal{C}_i| = 20$, $|C_{i,j}| = 50$ and $m = 100$

n	maximum	A1	A2	A3
10	7.2	7.95	3.41	1.28
20	5.0	7.51	3.68	22.99
30	4.0	6.62	4.12	111.55
40	4.0	5.69	3.25	380.03
50	3.1	6.06	5.27	2056.05

Table 2: Computational results for $n = 20$, $|C_{i,j}| = 50$ and $m = 100$

$\lvert\mathcal{C}_i\rvert$	maximum	A1	A2	A3
10	3.4	0.36	0.22	1.98
20	5.0	7.29	3.50	20.51
30	5.6	41.11	26.29	191.37
40	6.0	165.75	98.47	377.01
50	6.3	471.95	325.93	1392.41

Table 1 shows that Algorithms 1 and 2 are rather immune to the growth of n and Algorithm 3 becomes impractically slow when n grows. However, Algorithm 3 is the fastest if $n \lesssim 10$. Table 2 shows that all algorithms slow down quite uniformly when $|\mathcal{C}_i|$ grows. Tables 3 and 4 present the influence of $|C_{i,j}|/m \to 1$.

Table 3: Computational results for $n = 20$, $|\mathcal{C}_i| = 20$ and $m = 100$

$\lvert C_{i,j}\rvert$	maximum	A1	A2	A3
30	2.0	0.10	0.09	0.41
40	3.1	0.87	0.64	3.12
50	5.0	7.60	3.73	23.04
60	7.2	159.87	95.88	455.58
70	12.0	12877.21	5919.67	6395.50

To give some hints on the performance of these algorithms for real-world biological data from phylogenetic footprinting, we present the results from two computational tests. For the first instance, $n = 4$, $273 \leq |\mathcal{C}_i| \leq 711$, $1 \leq |C_{i,j}| \leq 37$, $m = 52$ with maximum value 6, and for the second instance, $n = 8$, $2317 \leq |\mathcal{C}_i| \leq 4289$, $1 \leq |C_{i,j}| \leq 140$, $m = 307$ with maximum value 30. The computational times (in CPU seconds) for the first instance were 0.01, 0.02 and 0.01 with A1, A2 and A3, respectively, and for the second instance they were 0.66, 10.20 and 3.14, respectively. Computing the second instance with the algorithms presented in [4] takes several CPU months.

Table 4: Computational results for $n = 20$, $|\mathcal{C}_i| = 20$ and $|C_{i,j}| = 60$

m	maximum	A1	A2	A3
100	7.3	155.26	88.05	399.09
150	3.2	2.79	1.88	11.76
200	2.0	0.51	0.42	2.21
250	2.0	0.40	0.41	0.42
300	1.3	0.42	0.36	0.43

Table 5: Biological meaning of the parameters involved

n	Number of transcription factors	≤ 1000
m	Number of related genes	≤ 30
$\|\mathcal{C}_i\|$	Binding sites within complete promoter sequence	≤ 3000
$\|C_{i,j}\|$	Binding sites within 500 nucleotides	≤ 100

Relative easiness of these real-world instances are due to several reasons: after the removal in the preprocessing step of elements not occurring in all $\mathcal{C}_i$, the instances become substantially smaller; and almost all elements in the preprocessed instances occur in a solution. The first instance is reduced so that we get the parameters $n = 4$, $273 \leq |\mathcal{C}_i| \leq 710$, $1 \leq |C_{i,j}| \leq 6$, $m = 6$, and the second instance is reduced to $n = 8$, $2317 \leq |\mathcal{C}_i| \leq 4284$, $1 \leq |C_{i,j}| \leq 31$, $m = 31$.

While further studies of biological data with the new algorithms will be carried out in subsequent work, it should be mentioned that the results here indicate suitability of our algorithms for such instances. As can be seen in Table 5, the relevant parameters considered in our artificial data sets comprise the complete range of relevant parameters for the cis-regulatory module discovery instances discussed in [4]. It should be mentioned that the parameters considered here are much larger than those considered in the computational experiments in [4], which were essentially limited to $m \leq 3$ and $n \leq 15$. The values in Table 5 are based on the assumption that a binding site is found at at most every 5 nucleotides, which can be considered a minimum requirement for sufficiently specific matches of transcription factor binding sites in a typical application setting. The number of transcription factor binding sites n is derived from the number of known profiles in databases such as TRANSFAC [2].

References

[1] R. Carraghan and P. M. Pardalos, An exact algorithm for the maximum clique problem, *Oper. Res. Lett.* **9** (1990), 375–382.

[2] T. Heinemeyer, E. Wingender, I. Reuter, H. Hermjakob, A. E. Kel, O. V. Kel, E. V. Ignatieva, E. A. Ananko, O. A. Podkolodnaya, F. A. Kolpakov, N. L. Podkolodny, and N. A. Kolchanov, Databases on transcriptional regulation: TRANSFAC, TRRD, and COMPEL, *Nucl. Acids Res.* **26** (1998), 364–370.

[3] D. L. Kreher, D. R. Stinson, Combinatorial Algorithms: Generation, Enumeration, and Search, CRC Press, Boca Raton, 1999.

[4] A. Mosig, T. Bıyıkoğlu, S. J. Prohaska, and P. F. Stadler, Detecting phylogenetic footprint clusters by optimizing barbeques, submitted for publication.

[5] P. R. J. Östergård, A fast algorithm for the maximum clique problem, *Discrete Appl. Math.* **120** (2002), 197–207.

[6] G. Verfaillie, M. Lemaître, and T. Schiex, Russian doll search for solving constraint optimization problems, in *Proc. 13th National Conference on Artificial Intelligence (AAAI-96), Portland, Oregon, August 4–8, 1996*, AAAI Press, Menlo Park, California, 1996, pp. 181–187.