Algorithms for Compact Letter Displays: Comparison and Evaluation

Jens Gramm [a] Jiong Guo [b] Falk Hüffner [b,*] Rolf Niedermeier [b]
Hans-Peter Piepho [c] Ramona Schmid [a]

[a] *Wilhelm-Schickard-Institut für Informatik, Universität Tübingen, Sand 13, D-72076 Tübingen, Germany*

[b] *Institut für Informatik, Friedrich-Schiller-Universität Jena, Ernst-Abbe-Platz 2, D-07743 Jena, Germany*

[c] *Fachgruppe Bioinformatik, Universität Hohenheim, D-70593 Stuttgart, Germany*

Abstract

Multiple pairwise comparisons are one of the most frequent tasks in applied statistics. In this context, *letter displays* may be used for a compact presentation of results of multiple comparisons. A heuristic previously proposed for this task is compared with two new algorithmic approaches. The latter rely on the equivalence of computing compact letter displays to computing *clique covers* in graphs, a problem that is well-studied in theoretical computer science. A thorough discussion of the three approaches aims to give a comparison of the algorithms' advantages and disadvantages. The three algorithms are compared in a series of experiments on simulated and real data, e.g., using data from wheat and triticale yield trials.

Key words: multiple pairwise comparison, line display, NP-hard problem, graph problem, clique cover, efficient algorithm

1 Introduction

Multiple testing procedures continue to receive much attention in the statistical literature (Hochberg and Tamhane, 1987; Shaffer, 1995; Hsu, 1996). Multiple pairwise comparisons among all pairs in a set of n treatments are a

* Corresponding author. Tel.: +49 3641 9 46324; fax: +49 3641 9 46002
E-mail address: hueffner@minet.uni-jena.de.

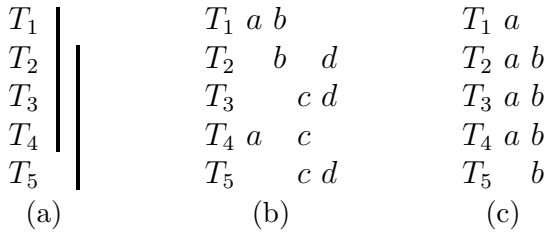


Fig. 1. (a) Line display for Example 1. (b) Letter display for Example 2. (c) Letter display for Example 1. Treatments connected by a common line (a) or having a common letter (b, c) are not significantly different.

common task in routine analyses based on analysis of variance (ANOVA) techniques (Hsu, 1996). Typically, the quantities of interest are treatment means, but multiple comparisons may also be applied to other statistics such as variances, quantiles, and estimates of entropy or biodiversity (Rogers and Hsu, 2001; Piepho, 2004). For multiple comparisons of means in a balanced ANOVA setting it is common to use a *line display* or *letter display* representing significant differences. These two types of display will now be briefly explained.

Assume that we are comparing five treatments $T_1, \ldots, T_5$. The comparison of T_1 and T_5 is significant, while all other comparisons are non-significant (Example 1). This can be conveyed by the line display (Steel and Torrie, 1980; Sokal and Rohlf, 1995; Clever and Scarisbrick, 2001) shown in Figure 1 (a).

An important feature of a line display is that all pairwise comparisons among treatments connected by a common line must be non-significant. Using an example, Piepho (2000) could show that it is not always possible to truthfully represent all statements of significance by a line display. Thus, he suggested the letter display as a generally applicable method (Piepho, 2004).

For a given set of n treatments and a set H of m pairwise comparisons where $\{T_i, T_j\} \in H$ if and only if the treatments T_i and T_j are significantly different, a letter display is a matrix $\mathbf{M}$ with n rows, one row for each treatment, which satisfies the following three conditions: (1) Each column contains a different *letter* and all non-empty entries of one column contain the same letter. (2) Every row has at least one non-empty entry. (3) Two treatments differ significantly if and only if the corresponding two rows of $\mathbf{M}$ contain no common letter. We call each non-empty entry of $\mathbf{M}$ a *letter occurrence*. For the sake of convenience, we say that a matrix $\mathbf{M}$ satisfying these three conditions *displays* H.

To illustrate a letter display, consider the case of five treatments $T_1, \ldots, T_5$ with the following significant comparisons: $H := \{\{T_1, T_5\}, \{T_1, T_3\}, \{T_2, T_4\}\}$ (Example 2). A letter display for this example is shown in Figure 1 (b).

It is important to note that a line display can always be converted to a letter display, and this is usually done in practice. The conversion is achieved by

replacing each line by a different letter. For Example 1 this yields the display given in Figure 1 (c). Conversely, it is not generally possible to convert a letter display into a line display. This is the case for Example 2. As it stands (Figure 1 (b)), only the letters b and c can be replaced by lines, while the columns for a and d show gaps and thus cannot be replaced by lines. It is sometimes possible to remove gaps by permutation of treatments (Piepho, 2000), but for Example 2 this is not possible. Thus, a line display is not generally feasible, while a letter display can always be devised.

For the sake of simplicity, we refer to the treatments only by their indices and equivalently consider letter displays as binary matrices which have entries "1" and "0" only, where a "1"-entry represents a letter occurrence and "0" is used to represent an empty entry. Clearly, there are only two conditions to be satisfied by a binary matrix for the purpose of displaying pairwise comparisons: (1) Each row has to contain at least one "1"-entry. (2) Two treatments have significantly different means if and only if there is no column in which the rows corresponding to the two treatments have "1"-entries.

Obviously, for a given set H of pairwise differences there can be several matrices $\mathbf{M}$ that display H. Among these possible letter displays we want to choose one that is as easy to perceive as possible. This may be achieved by meeting at least one of the following two goals: (1) minimizing the number of columns, and (2) minimizing the number of 1s in the letter display matrix. These two optimization criteria are specified by the following problem definition:

Compact Letter Display (CLD)
Input: A positive integer n and a set H of m unordered pairs of integers where $H = \{\{i_1, j_1\}, \{i_2, j_2\}, \ldots, \{i_m, j_m\}\}$ with $1 \leq i_r, j_r \leq n$ and $i_r \neq j_r$ for all $r = 1, \ldots, m$.
Objective of CLD-C**:** Find a binary matrix with minimum number of columns that displays H.
Objective of CLD-Σ**:** Find a binary matrix with minimum number of "1"-entries that displays H.

In situations where we address both objectives, we will refer to the problem as CLD. We use the term "nsd" (not significantly different) for pairs of treatments that are not in H.

For balanced data, line displays are readily available in most packages (SAS, SPSS, etc.), and such displays are often converted to letter displays for ease of presentation. By way of contrast, until recently such displays have not been available for unbalanced data. The new SAS procedure GLIMMIX produces a line display for unbalanced data, but this is not guaranteed to truthfully represent all significances; some significances may have to be suppressed. Piepho (2004) suggested a heuristic method to derive a letter display with respect to

the first optimization goal, i.e., to solve CLD-C. Here, we present two further methods for CLD-C. The methods considerably improve upon the method by Piepho (2004) in terms of computational efficiency *and* of economy of columns. In particular, while Piepho's approach is purely heuristic without guarantees on the quality of the solution, one of our new algorithms provides a provably optimal solution in terms of the number of columns that are needed. Our new methods rely on the equivalence of CLD-C and a well-known problem from graph theory, namely the problem of computing a so-called clique cover for a graph (see Conforti and Hochberg (1987) for an early example for using graph theory in the context of multiple comparisons). While the alternative methods have been described before in terms of graph theory, as one of our contributions, we present them here in their application to computing letter displays, thus making them more accessible to an audience interested in this application. Moreover, to solve CLD-Σ, we complement these methods with a problem-specific postprocessing introduced by Piepho (2004). We compare the algorithms based on their theoretical analysis. Most importantly, we present a set of experiments with real and simulated data that illustrate the differences between the approaches and thus provide concrete recommendations for statistical applications.

The paper is structured as follows. In Section 2, we briefly review the method by Piepho (2004) and present an overview of the two alternative approaches. This section also contains a thorough discussion of the three approaches, comparing their advantages and disadvantages from a theoretical point of view. We evaluate the algorithms experimentally in Section 3 and close with some conclusions in Section 4.

2 Algorithms for Compact Letter Display

In this section, we give an overview on computational complexity theory and then review three algorithms for computing compact letter displays under this aspect. All of them aim at computing a valid solution matrix with a minimum number of columns (the optimization criterion of CLD-C). In Section 2.6 we additionally describe *Sweeping*, a post-processing step aimed at minimizing the number of 1s in the solution matrix (the optimization criterion of CLD-Σ). Sweeping can be combined with any of the three algorithms.

To simplify the presentation, we sometimes treat a column of a letter display as a set of treatments, which is then the set of all treatments i for which the column has a 1 in row i.

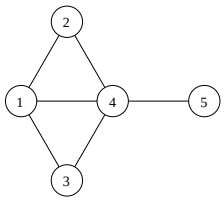
2.1 Computational Complexity Theory

Determining the computational complexity of problems is a key issue in theoretical computer science. It is crucial to distinguish between problems that can be solved efficiently and those that presumably cannot. To this end, theoretical computer science has coined the notions of *polynomial-time solvable* on the one hand and *NP-hard* on the other hand (Garey and Johnson, 1979; Papadimitriou, 1994). In this sense, polynomial-time solvability has become a synonym for efficient solvability. This means that for an input instance of a problem of "size" n an optimal solution can be computed in at most n^c computation steps, where c is some constant. By way of contrast, the unproven (!) working hypothesis of theoretical computer science is that NP-hard problems cannot be solved in n^c steps for any constant c. More specifically, typical runtimes for NP-hard problems are of the form 2^n or even worse; that is, we have an exponential growth of the number of computation steps.

As there are thousands of practically important NP-hard optimization problems (Papadimitriou, 1997) (one of which we will also encounter in this work), several approaches have been developed that try to circumvent the assumed computational intractability of NP-hard problems. One such approach is based on polynomial-time approximation algorithms, where one gives up looking for optimal solutions in order to have efficient algorithms. In other words, one is satisfied with a provably approximate solution that can be found in polynomial time (Ausiello et al., 1999; Vazirani, 2001). Fixed-parameter algorithms (Niedermeier, 2006) are another way of coping with computational intractability. Optimally solving NP-hard problems implies an "explosion" of the required time. The idea of fixed-parameter algorithms is to restrict this seemingly unavoidable combinatorial explosion to a "small part" of the input, the *parameter*, so that the given NP-hard problems can be solved efficiently as long as the parameter is small. Another commonly employed strategy is that of heuristics, where one gives up any provable performance guarantees concerning runtime and/or solution quality by developing algorithms that "usually" behave well in "most" practical applications. Since there may be no useful alternatives, this is sometimes the best one can do (Pearl, 1984; Michalewicz and Fogel, 2005).

2.2 Equivalence of Compact Letter Displays and Clique Cover

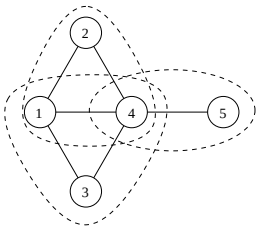
The Clique Cover problem, also known as Keyword Conflict problem (Kellerman, 1973) or Covering by Cliques (Garey and Johnson, 1979) or Intersection Graph Basis (Garey and Johnson, 1979), has applications in diverse fields of computer science. In this section, we show that Compact



(a)

1	1	1	0
2	1	0	0
3	0	1	0
4	1	1	1
5	0	0	1

(b)



(c)

Fig. 2. Equivalence of CLD-C and Clique Cover: Consider the CLD-C instance with $n = 5$ and $H = \{\{1,5\},\{2,3\},\{2,5\},\{3,5\}\}$. This instance can be translated into the graph $G = (V, E)$ with $V = \{1,2,3,4,5\}$ and $E = \{\{1,2\},\{1,3\},\{1,4\},\{2,4\},\{3,4\},\{4,5\}\}$, shown in (a): Treatments correspond to vertices in the graph, and two vertices i and j are connected iff $\{i,j\}$ is not in H. In (b) we give a letter display for H and in (c) the corresponding clique cover: Columns in the letter display correspond to cliques in the graph where the "1"-entries of the column determine the set of vertices in the corresponding clique.

Letter Display-C is equivalent to Clique Cover. This has two important implications: Like Clique Cover, CLD-C is NP-hard (see Section 2.1); and we can transfer solving methods from Clique Cover to CLD-C as demonstrated in Sections 2.4 and 2.5.

Clique Cover
Input: An undirected graph $G = (V, E)$.
Objective: Find a smallest possible set of cliques in G such that each edge in E has both its endpoints in at least one of the selected cliques. Here, a clique in a graph is a set of vertices such that every pair of vertices in this set is connected by an edge.

We establish the following correspondences between a CLD-C instance and a Clique Cover instance (see Figure 2 for an example):

- Treatments or rows of the letter display correspond to graph vertices;
- Pairs of nsd treatments correspond to edges, that is, the elements of H correspond to non-edges;
- Letter display columns correspond to cliques in the clique cover.

It is then not too hard to see that, with these correspondences, a solution for Clique Cover implies a solution of the same size for the corresponding CLD-C instance, and vice versa. Special handling is only required for treatments significantly different from all other treatments (corresponding to vertices with no edges attached): they require a letter display column, but no clique is needed to cover them in the given problem formulation. This is easily accounted for by a straightforward preprocessing step.

The Insert-Absorb heuristic is a method for solving CLD-C (Piepho, 2004).

Algorithm. The idea is to start with a letter display with only one column and all entries set to 1, that is, considering all treatments as nsd. Then this letter display is successively updated by taking pairs of significantly different treatments into account. Once all pairwise differences from H are processed, we obtain a correct letter display. The update of the letter display consists of two steps, *insertion* and *absorption*.

The *insertion step* updates a letter display with an additional significant difference between treatments i and j. A column having 1-entries both in rows i and j violates the conditions of a correct letter display. Therefore, each of these columns is duplicated; in one of its copies entry i is flipped to 0, and in the other copy entry j is flipped to 0. The resulting matrix is a correct letter display for the set of pairwise differences augmented by $\{i, j\}$.

The insertion step frequently creates redundant columns. In an *absorption step*, we compare each new column against all old columns to determine whether it is *absorbed* by one of the old columns. An old column absorbs the new column if it has a 1 in every row in which the new column has one. Every absorbed column is deleted. It is easy to see that absorbed columns are in fact redundant and that we obtain a correct letter display.

Notably, the order of processing the pairwise differences does not influence the conformation of the resulting letter display (a proof is given by Schmid (2005)).

Analysis. Unfortunately, the Insert-Absorb heuristic has an exponential worst-case runtime: There are inputs on which the Insert-Absorb heuristic requires at least $2^{n/2} \cdot n^2/2$ steps for n treatments and $n/2$ pairwise differences. An example of this is given by considering the Insert-Absorb heuristic for the input $n = 6$ and $H = \{\{1, 2\}, \{3, 4\}, \{5, 6\}\}$. The matrices produced when processing the single pairs of H by the Insert-Absorb heuristic are given as follows:

1	1		1	0		1	1	0	0		1	2	3	4	5	6	7	8
2	1	{1,2}	0	1	{3,4}	0	0	1	1	{5,6}	1	1	1	1	0	0	0	0
3	1	→	1	1	→	1	0	1	0	→	0	0	0	0	1	1	1	1
4	1		1	1		0	1	0	1		1	1	0	0	1	1	0	0
5	1		1	1		1	1	1	1		0	0	1	1	0	0	1	1
6	1		1	1		1	1	1	1		1	0	1	0	1	0	1	0
											0	1	0	1	0	1	0	1

The exponential runtime dependency is given by the term $2^{n/2}$: e.g., 32 for $n = 10$, 33 554 432 for $n = 50$, and already 10^{18} for $n = 120$, which is not feasible to compute. The fact that for some inputs the Insert-Absorb heuristic produces auxiliary matrices with $2^{n/2}$ columns stands in sharp contrast to the observation that every input has a solution matrix with at most n^2 columns. Moreover, we have no (non-trivial) provable guarantee on the quality of the solutions returned by the heuristic; that is, we do not know how far the produced letter display is away from an optimal one. For these reasons, we introduce two alternative approaches to compute letter displays.

2.4 Clique-Growing Heuristic

In this section, we review a novel heuristic for CLD-C that is based on the equivalence between CLD-C and Clique Cover.

While the heuristic, which we call "Clique-Growing", was originally described in terms of graph theory (Kellerman, 1973; Kou et al., 1978), we present it here as a method of constructing letter displays.

Algorithm. Like the Insert-Absorb heuristic described before, the Clique-Growing heuristic starts with a trivial letter display and then successively takes the input information into account and adapts the matrix accordingly: it processes the treatments in some fixed order and accounts for all differences of a treatment t to already processed treatments by either adding t to existing columns, or opening new columns (recall that, for simplicity, we occasionally treat a column as a set of treatments).

We start with a matrix with no columns and successively, for $i = 1, \ldots, n$, update the matrix to account for the pairwise differences between treatment i and treatments j with $j < i$. More specifically, for each treatment j with $j < i$ and $\{i, j\} \notin H$, the updated matrix contains a column C with $\{i, j\} \subseteq C$. If i is significantly different from every treatment in T, the only possibility is to add a new column containing only i. Otherwise, we add i to every column where this is possible, that is, where all treatments contained in this column are nsd to i. There might remain a set of treatments $W \subseteq T$ that are nsd to i and for which this fact is still not represented in the matrix. For these, we need to create new columns containing i and some elements of W. To cover as many elements of W at a time as possible, we examine the intersection of W with each column. Since these intersections form sets of pairwise nsd treatments, they are good candidates for new columns to be added together with i. We repeatedly choose the largest of these intersections to create new columns, until all elements in W are accounted for.

Analysis. The runtime of the Clique-Growing heuristic is polynomial: With a straight-forward implementation, it works in $O(n^5)$ steps for n treatments, and with some modifications and a careful analysis, an upper bound of $O(n^3)$ can be established for the algorithm including Sweeping (Gramm et al., 2006). Sweeping is described in Section 2.6.

2.5 Search-Tree Algorithm

In this section, we review a further algorithm for CLD-C, which, like Clique-Growing, is based on the equivalence between CLD-C and Clique Cover (see Figure 2). This algorithm has an exponential runtime but, in contrast to the previous two approaches, guarantees an optimal solution with respect to CLD-C, that is, a letter display with a minimum number of columns. Here, we give a brief overview on the algorithm which is, in terms of graph theory, described in more detail by Gramm et al. (2006).

Algorithm. Search trees are a popular means of exactly solving hard problems. The basic method is to identify for a given instance a small set of simplified instances such that the given instance has a solution if at least one of the simplified instances has one. The algorithm "branches" recursively into several subcases, each subcase corresponding to a simplified instance, until a stopping criterion is met. In this way, it finds a solution by an exhaustive search that is realized as a recursive procedure. During branching, new opportunities for data reduction can arise and, therefore, the search tree is interleaved with data reduction.

Focusing on the objective of CLD-C, an important observation is that, without loss of generality, we can assume that every column contains a *maximally-nsd* subset of the treatments. Here, a set T' of treatments is called maximally-nsd when all treatments in T' are pairwise nsd and no treatment from T can be added to T' while maintaining this property. This observation can, on real data sets, drastically decrease the number of choices we have, as columns only need to be generated for maximally-nsd subsets of treatments.

We branch on maximally-nsd subsets of treatments. To this end, we choose an uncovered pair of nsd treatments, enumerate all maximally-nsd subsets of treatments containing this pair, and then branch according to the maximally-nsd subset for which we add a column to the solution matrix. Details of the algorithm, e.g., how to choose the treatment pair to branch on and how to enumerate maximally-nsd subsets of treatments containing the pair are explained by Gramm et al. (2006).

Table 1
Overview on the theoretical characteristics of the three algorithms solving Compact Letter Display, with respect to their runtime and their guarantees of producing a solution with a minimum number of columns or a minimum number of 1s.

Algorithm	runtime	optimality	
		no. of columns	no. of 1s
Insert-Absorb	exponential	no	no
Clique-Growing	polynomial	no	no
Search-Tree	exponential	guaranteed	no

As shown by Gramm et al. (2006), this algorithm is guaranteed to find a solution with a minimum number of columns. It is not obvious how a non-trivial upper bound on the exponential runtime can be given. Note that Gramm et al. (2006) additionally introduced four useful data reduction rules for CLD-C which replace, in polynomial time, a given CLD-C instance by a simpler instance such that a solution to the simplified instance is also a solution to the original one. In this way these data reduction rules greatly reduce the search tree size and enhance the performance of the algorithm.

2.6 Sweeping

We now describe the "Sweeping" method given by Piepho (2004) to reduce the number of 1s in a given solution, the objective of CLD-Σ.

Algorithm. Given a letter display for H, the goal is now to minimize its number of 1s while still retaining a correct letter display. We check for every treatment i in a column C whether i can be omitted from C. We can omit i if for all other treatments j in C there is another column containing both i and j.

Notably, the order of processing the 1s can influence the resulting matrix. Piepho (2004) does not give a strategy for processing the entries. A reasonable strategy is to examine the entries in a random order, repeating several times, and picking the best result.

A straightforward implementation of Sweeping requires $O(n^3)$ steps.

Ideally, a user interested in solving an optimization problem would like to compute an *optimal* solution within a *short runtime*, say within a second. For CLD-C we have a proof of its NP-hardness, i.e., a proof of its combinatorial difficulty. This means that it is very unlikely that one can achieve both user goals at the same time, unless the instances to be solved are of small size. Moreover, to complicate the situation, Compact Letter Display has two optimization criteria, namely to minimize the number of columns in a computed letter display, and to minimize the number of 1s.

In Table 1, we give an overview of the characteristics of the algorithms derived from their *theoretical* analysis. Section 3.3 will discuss to what extent the theoretical properties of the algorithms are confirmed by their *empirical* evaluation.

Regarding the runtime, Table 1 shows that both the Insert-Absorb heuristic and the Search-Tree algorithm have *exponential* runtime, which implies that runtime may grow quickly with growing instance sizes and is likely to become intolerably large even for instances of moderate size. In contrast, the Clique-Growing algorithm has *polynomial* runtime, meaning that we have a guarantee that the runtime grows only moderately with growing instance size, also promising small runtimes for large instances.

Regarding the number of columns, only the Search-Tree algorithm gives a guarantee that the number of columns in the produced letter display is as small as possible. For the other two approaches, we cannot be sure whether the produced letter display could not be replaced by one with fewer columns. Note that for the user of letter displays it is highly desirable to have these as compact as possible, so it is desirable to find solutions with a minimum number of columns. Regarding the number of 1s, none of the three algorithms gives a guarantee. In fact it is an open problem to find a competitive algorithm computing a letter display with a minimum number of 1s. It is even open whether it is possible to achieve both optimization goals at the same time, i.e., minimizing the number of columns and the number of 1s simultaneously.

3 Experimental Evaluation

In this section, we report on the application of the three algorithms (all including Sweeping) discussed in this paper. All algorithms were implemented in the Objective Caml programming language (Leroy et al., 1996), which is available for all major platforms. The source code is available from the au-

Table 2
Comparison of the performance of the Search-Tree algorithm, the Clique-Growing heuristic, and the Insert-Absorb heuristic on five datasets from real-world statistical analyses. For each dataset and each heuristic we report the number of columns of the computed letter display (cols), the number of 1s (1s), and the runtime in seconds (time).

			Insert-Absorb			Clique-Growing			Search-Tree		
Dataset	n	$\|H\|$	cols	1s	time	cols	1s	time	cols	1s	time
Triticale 1	13	55	4	20	0.00	4	20	0.00	4	20	0.00
Triticale 2	17	86	5	32	0.00	5	33	0.00	5	32	0.00
Wheat 1	124	4847	56	986	1.93	50	711	0.20	49	663	4.00
Wheat 2	121	4706	50	902	1.66	48	605	0.17	48	656	3.69
Wheat 3	97	3559	39	691	1.03	32	484	0.15	31	487	2.08
Rapeseed 1	47	576	20	176	0.02	20	149	0.00	20	175	0.04
Rapeseed 2	57	1040	20	244	0.05	20	202	0.01	20	205	0.12
Rapeseed 3	64	1260	24	288	0.08	24	237	0.01	24	232	0.17
Rapeseed 4	62	1085	19	222	0.04	19	207	0.02	19	204	0.11
Rapeseed 5	64	1456	19	259	0.09	19	231	0.03	19	232	0.27
Rapeseed 6	70	1416	27	332	0.12	27	293	0.02	27	301	0.27
Rapeseed 7	74	1758	29	387	0.15	27	356	0.03	25	344	0.35
Rapeseed 8	59	1128	17	215	0.04	17	186	0.01	17	229	0.12
Rapeseed 9	76	1835	30	424	0.21	30	327	0.04	30	332	0.64

thors upon request. The testing machine is an AMD Athlon 64 3400+ with 2.4 GHz, 512 KB cache, and 1 GB main memory, running under the Debian GNU/Linux 3.1 operating system.

3.1 Real-World Data

In the following, we discuss results on three sets of data, each set containing several examples. We use real-world data with many treatments. This requires some justification. Letter displays are most frequently used and needed when the number of treatments is smaller than 20. It should be emphasized, however, that multiple comparisons are used also when the number of treatments is large. For example, plant breeders often perform such tests when there are many genotypes (> 20), though they often only report a mean least significant difference (e.g. Miedaner et al., 2004) or the mean standard error of a difference (Kempton and Fox, 1997). The reason for this is the lack of any procedures to display the results of individual pairwise comparisons for large unbalanced datasets. Thus, there is a need for informative displays of multiple comparisons for a larger number of treatments. While letter displays might not be appropriate in these cases, advanced visualization methods (for example utilizing colors, additional visual elements, or software interactivity) would

probably still require the structure discovered by our algorithms. This being said, we would like to stress that our main reason for emphasizing examples with many treatments is to put the different algorithms to the hardest possible test, as it turned out that differences between competing methods become most prominent in this case. Moreover, we also want to have a sound empirical evaluation of to what extent the solution quality of the heuristic methods (Insert-Absorb and Clique-Growing) typically deviates from the quality of the exact Search-Tree algorithm. We measure the algorithm performance with respect to three criteria: (1) the numbers of columns in the produced letter displays, (2) the numbers of 1s, and (3) the runtime. The results are displayed in Table 2.

Triticale yield trials. The first two examples refer to the comparison of means in a mixed model analysis of series of yield trials with triticale in Germany. Details of the data and the mixed model analysis for the first example are described by Piepho (2000).

These examples are relatively small, containing 13 and 17 treatments. On these data, the algorithms perform almost equally: their runtime is negligible and the approaches produce letter displays with the same number of columns. However, with respect to the numbers of 1s, the Clique-Growing heuristic is outperformed by the other two approaches by one entry.

Winter wheat yield trials. These data are from a series of regional cultivar trials with winter wheat in the federal state Mecklenburg–Western Pomerania (Germany). Trials were performed at several locations between 1999 and 2004, and they involved a total of 124 cultivars (treatments). The data were analyzed according to a standard mixed linear model as described by Piepho and Michel (2000). Multiple comparisons among all pairs of treatments were performed using a t-test with degrees of freedom determined by the Kenward–Roger method (Kenward and Roger, 1997). The locations fell into two ecologically distinct subregions. The data in dataset "Wheat 1" were analyzed ignoring the subdivision into subregions, datasets "Wheat 2" and "Wheat 3" contain data separately for subregions.

These examples are considerably larger than the examples in the first set, containing 97–124 treatments. Here, we can observe differences in the algorithms' performance: In all three examples, the Search-Tree algorithm produces letter displays with less columns than the two heuristics while requiring the largest runtime. The Clique-Growing heuristic produces letter displays of almost-optimal size. We observe large deviations in the number of 1s with the Clique-Growing heuristic requiring the fewest 1s in two of three examples.

Oilseed rape yield trials. The third set of examples concerns a series of yield trials with oilseed rape conducted by the Bundessortenamt in Germany from 1991 to 2000. The data fall into nine series of trials, each replicated over three years at several locations. The design in each trial was a randomized complete block design. The series had between 47 and 76 cultivars, and it was highly unbalanced with respect to years, location, and cultivars tested. We analyzed cultivar means per trial according to a three-way linear model. The three main effects and the location-by-year interaction were taken as fixed, while the remaining effects were considered as random. The inverse of the squared standard error was used as a weight in order to partition error from cultivar-by-location-by-year interaction. For more details see Piepho and Möhring (2005).

These examples are of medium size, containing 47–76 treatments. The three algorithms produce letter displays of the same size in all except one example in which the Search-Tree algorithm outperforms the heuristics. The runtime of all algorithms clearly remains below one second for all examples. With respect to the number of 1s, the Search-Tree algorithm and the Clique-Growing heuristic yield the best results, with the heuristic (slightly) outperforming the Search-Tree algorithm in six out of nine examples.

3.2 Simulated Data

To get a broader view of the performance of the three algorithms, we generated synthetic problem instances.

Model. We simulated significance tests for the pairwise comparison of means. Our simulation procedure is such that the number of treatments (n) can be chosen arbitrarily. Treatment means m_i were generated from a normal distribution with mean ϕ and variance θ^2. These were parameterized based on an analysis of the oilseed rape data as $\phi = 42.6$ and $\theta^2 = 12.7$. The variance-covariance matrix $\Sigma = \{\sigma_{ii'}\}$ of simulated sample means was generated according to $\Sigma = D^{1/2}RD^{1/2}$, where R is the correlation matrix and D is a diagonal matrix containing the diagonal elements of Σ, i.e. the variances, and $D^{1/2}$ is the square root of D. Diagonal elements of D were generated from a gamma distribution with shape parameter $\alpha = 4.12$ and scale parameter $\beta = 0.429$, as determined from the oilseed rape data. The off-diagonal elements $r_{ii'}$ $(= r_{i'i})$ of R were simulated as $r_{ii'} = (\exp(2z_{ii'}) - 1)/(\exp(2z_{ii'}) + 1)$ where $z_{ii'}$ are i.i.d. normal deviates with mean $\lambda = 0.00625$ and variance $\eta = 0.00182$. In order to make sure that R was positive definite, we used its spectral decomposition $R = \sum_{k=1}^{K} d_k q_k q_k'$, where d_k are the eigenvalues of R and q_k the corresponding eigenvectors (Harville, 2000, p. 537). We then set any non-positive d_k

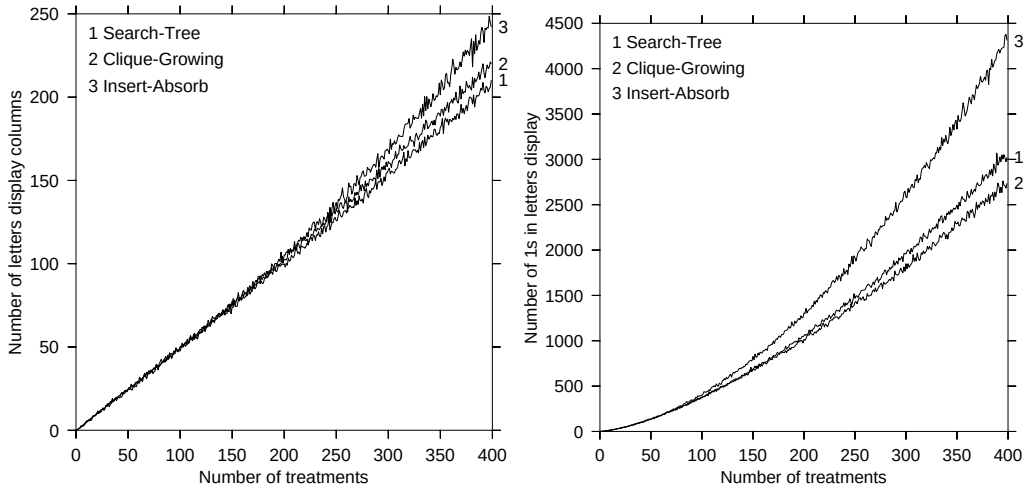


Fig. 3. Comparison of the output quality of the three algorithms. Averaged over 10 random instances.

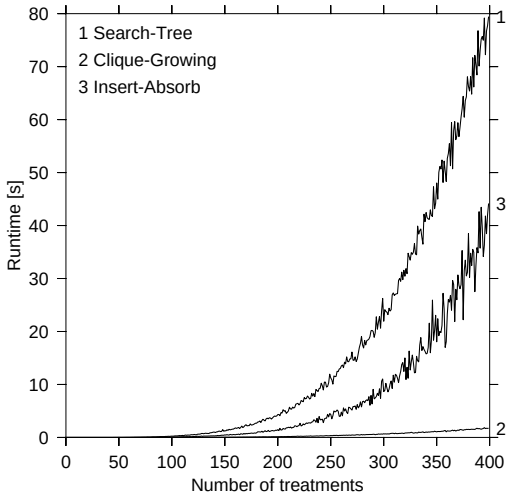


Fig. 4. Comparison of the runtime of the three algorithms. Averaged over 10 random instances.

equal to the smallest positive eigenvalue and re-computed R from the spectral decomposition. It should be mentioned that the resulting R may have diagonal elements < 1 such that it is not a true correlation matrix. The resulting matrix Σ, however, will always have the properties of a variance-covariance matrix and so will be valid for our purposes. The tests statistic for a pairwise comparison among treatments i and i' was $t_{ii'} = (m_i - m_{i'})/\sqrt{\sigma_{ii} + \sigma_{i'i'} - 2\sigma_{ii'}}$. The difference of means for treatments i and i' was declared significant at the 5% level when $|t_{ii'}| > 1.96$.

Results. Using the described model, we generated random instances of varying size. Randomized sweeping, repeated 10 times, was applied for each algorithm. The results are shown in Figures 3 and 4.

Table 3
Overview on the empirical analysis of the three algorithms solving Compact Letter Display, with respect to their runtime and their performance in producing a solution with a small number of columns or a small number of 1s. We indicate the ranking (first, second, third) of the algorithms with respect to each criterion.

Rank	Runtime	No. of columns	No. of 1s
1.	Clique-Growing	Search-Tree	Clique-Growing
2.	Insert-Absorb	Clique-Growing	Search-Tree
3.	Search-Tree	Insert-Absorb	Insert-Absorb

3.3 Evaluations and Recommendations

In Table 3, we show the summarized ranking of the three algorithms with respect to the three optimization criteria, namely runtime, number of columns, and number of 1s. In the following, we summarize our insights from the results on real-world (Section 3.1) and simulated (Section 3.2) data, and conclude with recommendations.

With respect to runtime, our results for simulated data clearly show that the Search-Tree algorithm requires the longest runtime and that the Clique-Growing heuristic is fastest. However, results for real-world data show that for these instances, it is still tolerable to run the Search-Tree algorithm.

With respect to the number of columns, our results for simulated data consistently show that the Search-Tree algorithm yields the best results. However, our results for real-world data reveal that for instances of small size the other algorithms also yield optimal or near-optimal results in many (but not all) cases.

With respect to the number of 1s, our results for simulated data consistently show that overall the Clique-Growing approach yields the best results. Results for real-world data confirm this in many cases, while in some cases the Search-Tree algorithm is best.

For a concluding recommendation, note that the Insert-Absorb heuristic performs poorest for both quality criteria; as, in addition, it has no guarantees for its runtime, it seems always advisable to prefer one of the other two algorithms. Between the Search-Tree algorithm and the Clique-Growing algorithm, the choice depends on user priorities. The Search-Tree algorithm always produces letter displays with a minimum number of columns; the Clique-Growing algorithm typically produces letter displays with fewer 1s. However, in the range of input sizes most commonly encountered in practice, neither difference is very pronounced. The algorithms differ in the guarantees they offer: The Search-Tree algorithm guarantees an optimal number of columns for the result, and

the Clique-Growing algorithm guarantees a not exceedingly long runtime even in extreme cases.

4 Conclusion

This is an interdisciplinary piece of work based on a fruitful interaction between applied statistics and theoretical computer science, contrasting a theoretical analysis of algorithmic approaches with their practical evaluation. Two of these approaches, derived through the equivalence between the problems Compact Letter Display from applied statistics and Clique Cover from graph theory, are newly presented to a statistical audience. In particular, we shed new light on a previous heuristic for Compact Letter Display (Piepho, 2004) which is generally slower and less effective than the new algorithms.

To compare the proposed algorithms, we focussed on examples with many treatments, because differences among methods tend to be most pronounced in these cases. Most applications we see in practice, however, involve only a small number of treatments. When the number of treatments increases, reading letter displays may become tedious. Also, with a large number of treatments, selection of the best treatment is often the primary aim, while all pairwise comparisons among treatments are of less importance. Thus, letter displays are probably of most value when the number of treatments is not very large. Reducing the number of symbols needed to the minimum enhances readability of letter displays, and this paper has proposed and compared efficient algorithms for this purpose.

One question left open in this work is to find an efficient algorithm that computes letter displays with a minimum number of 1s. It is also a striking open question whether there is always a letter display that minimizes both the number of columns and the number of 1s at the same time.

Acknowledgements

The authors thank Benny Chor (Tel Aviv University) for fruitful discussions during an early phase of working on the CLD problem. We thank the Bundessortenamt (Hannover, Germany) for providing the oilseed rape data. Marianne Karpenstein-Machan (Universität Göttingen, Germany) is thanked for providing the triticale data, which she collected from various variety testing authorities in Germany. The winter wheat data were kindly made available

by Volker Michel (Landesforschungsanstalt für Landwirtschaft und Fischerei Mecklenburg-Vorpommern, Gülzow, Germany).

Jens Gramm was supported by the Deutsche Forschungsgemeinschaft, project OPAL (optimal solutions for hard problems in computational biology), NI 369/2. Jiong Guo and Falk Hüffner were supported by the Deutsche Forschungsgemeinschaft, Emmy Noether research group PIAF (fixed-parameter algorithms), NI 369/4.

References

Ausiello, G., Crescenzi, P., Gambosi, G., Kann, V., Marchetti-Spaccamela, A., Protasi, M., 1999. Complexity and Approximation. Springer.

Clever, A. G., Scarisbrick, D. H., 2001. Practical Statistics and Experimental Design for Plant and Crop Science. John Wiley&Sons.

Conforti, M., Hochberg, Y., 1987. Sequentially rejective pairwise testing procedures. Journal of Statistical Planning and Inference 17, 193–208.

Garey, M. R., Johnson, D. S., 1979. Computers and Intractability: A Guide to the Theory of NP-Completeness. W. H. Freeman.

Gramm, J., Guo, J., Hüffner, F., Niedermeier, R., 2006. Data reduction, exact, and heuristic algorithms for clique cover. In: Proc. 8th Workshop on Algorithm Engineering and Experiments (ALENEX'06). SIAM Press, pp. 86–94.

Harville, D. A., 2000. Matrix Algebra from a Statisticians Perspective. Springer.

Hochberg, Y., Tamhane, A. C., 1987. Multiple Comparison Procedures. Wiley.

Hsu, J. C., 1996. Multiple Comparisons: Theory and Methods. Chapman&Hall.

Kellerman, E., 1973. Determination of keyword conflict. IBM Technical Disclosure Bulletin 16 (2), 544–546.

Kempton, R. A., Fox, P. N. (Eds.), 1997. Statistical Methods for Plant Variety Evaluation. Chapman and Hall.

Kenward, M. G., Roger, J. H., 1997. Small sample inference for fixed effects from restricted maximum likelihood. Biometrics 53 (3), 983–997.

Kou, L. T., Stockmeyer, L. J., Wong, C. K., 1978. Covering edges by cliques with regard to keyword conflicts and intersection graphs. Communications of the ACM 21 (2), 135–139.

Leroy, X., Vouillon, J., Doligez, D., et al., 1996. The Objective Caml system. Available on the web, `http://caml.inria.fr/ocaml/`.

Michalewicz, Z., Fogel, D. B., 2005. How to Solve It: Modern Heuristics. Springer.

Miedaner, T., Heinrich, N., Schneider, B., Oettler, G., Rohde, S., Rabenstein, F., 2004. Estimation of deoxynivalenol (DON) content by symptom rat-

ing and exoantigen content for resistance selection in wheat and triticale. Euphytica 139 (2), 123–132.
Niedermeier, R., 2006. Invitation to Fixed-Parameter Algorithms. Oxford University Press.
Papadimitriou, C. H., 1994. Computational Complexity. Addison-Wesley.
Papadimitriou, C. H., 1997. NP-completeness: A retrospective. In: Proc. 24th International Colloquium on Automata, Languages and Programming (ICALP '97). Vol. 1256 of LNCS. Springer, pp. 2–6.
Pearl, J., 1984. Heuristics: Intelligent Search Strategies for Computer Problem Solving. Addison-Wesley.
Piepho, H.-P., 2000. Multiple treatment comparisons in linear models when the standard error of a difference is not constant. Biometrical Journal 42 (7), 823–835.
Piepho, H.-P., 2004. An algorithm for a letter-based representation of all-pairwise comparisons. Journal of Computational and Graphical Statistics 13, 456–466.
Piepho, H.-P., Michel, V., 2000. Considerations on regional evaluation of cultivar trials (in German: Überlegungen zur regionalen Auswertung von Landessortenversuchen). Informatik, Biometrie und Epidemiologie in Medizin und Biologie 31, 123–136.
Piepho, H.-P., Möhring, J., 2005. Best linear unbiased prediction for subdivided target regions. Crop Science 45, 1151–1159.
Rogers, J. A., Hsu, J. C., 2001. Multiple comparisons of biodiversity. Biometrical Journal 43 (5), 617–625.
Schmid, R., 2005. On an application of edge clique cover in applied statistics. Studienarbeit, Wilhelm-Schickard-Institut für Informatik, Tübingen.
Shaffer, J. P., 1995. Multiple hypothesis testing. Annual Review of Psychology 46, 561–584.
Sokal, R. R., Rohlf, F. J., 1995. Biometry. W. H. Freeman.
Steel, R. G. D., Torrie, J. H., 1980. Principles and Procedures of Statistics. McGraw-Hill.
Vazirani, V. V., 2001. Approximation Algorithms. Springer.