Minimal Conflicting Sets for the Consecutive Ones Property in ancestral genome reconstruction

Cedric Chauve[1], Utz-Uwe Haus[2], Tamon Stephen[1], and Vivija P. You[1]

[1] Department of Mathematics, Simon Fraser University, Burnaby (BC), Canada
[cedric.chauve,tamon,vpy]@sfu.ca
[2] Institute for Mathematical Optimization, University of Magdeburg, Germany
haus@imo.math.uni-magdeburg.de

Abstract. A binary matrix has the Consecutive Ones Property (C1P) if its columns can be ordered in such a way that all 1's on each row are consecutive. A *Minimal Conflicting Set* is a set of rows that does not have the C1P, but every proper subset has the C1P. Such submatrices have been considered in comparative genomics applications, but very little is known about their combinatorial structure and efficient algorithms to compute them. We first describe an algorithm that detects rows that belong to Minimal Conflicting Sets. This algorithm has a polynomial time complexity when the number of 1s in each row of the considered matrix is bounded by a constant. Next, we show that the problem of computing all Minimal Conflicting Sets can be reduced to the joint generation of all minimal true clause and maximal false clauses for some monotone boolean function. We use these methods in preliminary experiments on simulated data related to ancestral genome reconstruction.

Publiched in the proceedings of the Seventh Annual RECOMB Satellite Workshop on Comparative Genomics, RECOMB-CG 2009, volume 5817 of *Lecture NOtes in Bioinformatics*, pp. 48–48, 2009. Version of October 9, 2009.

1 Introduction

A binary matrix M has the Consecutive Ones Property (C1P) if its columns can be ordered in such a way that all 1's on each row are consecutive. Deciding if a matrix has the C1P can be done in linear-time and space [4, 13, 15, 19, 20]. The problem of testing if a matrix has the C1P has been considered in genomics, for problems such as physical mapping [2, 6, 17] or ancestral genome reconstruction [1, 5, 18, 21].

If a binary matrix M does not have the C1P, a *Minimal Conflicting Set* (MCS) is a submatrix M' of M composed of a subset of the rows of M such that M' does not have the C1P, but every proper subset of rows of M' has the C1P. The Conflicting Index (CI) of a row of M is the number of MCS that contains this row. Minimal Conflicting Sets were introduced in [3]. In the context of ancestral genome reconstruction, MCS were used in [3] and in [23], where the CI was used to rank the rows of M and input into a branch-and-bound algorithm that computes parsimonious evolution scenarios for gene clusters. In both papers, the question of computing the CI of the rows of a non-C1P binary matrix M, or more generally to compute all MCS of M, was raised.

In the present paper, after some preliminaries on the C1P and MCS (Section 2), we attack two problems. First, in Section 3, we consider the problem of deciding if a given row of a binary matrix

M belongs to at least one MCS. We show that, when all rows of a matrix are constrained to have a bounded number of 1's, deciding if the CI of a row of a matrix is greater than 0 can be done in polynomial time. Next, in Section 4, we attack the problem of generating all MCS for a binary matrix M. We show that this problem, which is #P-hard, can be approached as a joint generation problem of minimal true clauses and maximal false clauses for monotone boolean functions. This can be done in quasi-polynomial time thanks to an oracle-based algorithm for the dualization of monotone boolean functions [8, 9, 12]. We implemented this algorithm [14] and applied it on simulated data (Section 5). We conclude by discussing several open problems.

2 Preliminaries

We briefly review here ancestral genome reconstruction and algorithmic results related to Minimal Conflicting Sets. Let M be a binary matrix with m rows and n columns, with e entries 1. We denote by $r_1, \ldots, r_m$ the rows of M and $c_1, \ldots, c_n$ its columns. We assume that M does not have two identical rows, nor two identical columns, nor a row with less than two entries 1 or a column with no entry 1. We denote by $\Delta(M)$ the maximum number of entries 1 found in a single row of M. In the following, we sometimes identify a row of M with the set of columns where it has entries 1.

Minimal Conflicting Sets, Maximal C1P Sets and Conflicting Index. A Minimal Conflicting Set (MCS) is a set R of rows of M that does not have the C1P but such that every proper subset of R has the C1P. The Conflicting Index (CI) of a row r_i of M is the number of MCS that contain r_i. A row r_i of M that belongs to at least one conflicting set is said to be a *conflicting row*.

A Maximal C1P Set (MC1P) is a set R of rows of M that has the C1P and such that adding any additional row from M to it results in a set of rows that does not have the C1P. For a subset $I = \{i_1, \ldots, i_k\}$ of $[n]$, we denote by R_I the set $\{r_{i_1}, \ldots, r_{i_k}\}$ of rows of M. If R_I is an MCS (resp. MC1P), we then say that I is an MCS (resp. MC1P).

Ancestral genome reconstruction. The present work is motivated by the problem of inferring an ancestral genome architecture given a set of extant genomes. An approach to this problem, described in [5], consists in defining an alphabet of genomic markers that are believed to appear uniquely in the extinct ancestral genome. An *ancestral synteny* is a set of markers that are believed to have been consecutive along a chromosome of the ancestor. A set of ancestral syntenies can then be represented by a binary matrix M: columns represent markers and the 1 entries of a given row define an ancestral synteny. If all ancestral syntenies are true positives (i.e. represent sets of markers that were consecutive in the ancestor), then M has the C1P. Otherwise, some ancestral syntenies are false positives that create MCS and the key problem is to remove them. This problem is generally attacked by removing from M the minimum number of rows such that the resulting matrix has the C1P; this optimization problem is NP-hard, even in the simpler case where every row of M contains exactly two entries 1, and is then often attacked using heuristics or branch-and-bound methods [1, 5, 18, 21].

Computing the conflicting index. In the case where each row of M has exactly two entries 1, M naturally defines a graph G_M with vertex set $\{c_1, \ldots, c_n\}$ and where there is an edge between c_i and c_j if and only if there is a row with entries 1 in columns c_i and c_j. A set of rows R of M is an MCS if and only if the subgraph induced by the corresponding edges is a star with four vertices (also called a claw) or a cycle. This implies immediately that the number of MCS can be exponential in n. Also, combined with the fact that counting the number of cycles that contain a given edge in an arbitrary graph is #P-hard [25], this leads to the following result.

Theorem 1. *The problem of computing the Conflicting Index of a row in a binary matrix is #P-hard.*

Testing for Minimal Conflicting Sets. Given a set of p rows R of M, deciding whether these rows form an MCS can be achieved in polynomial time by testing (1) whether they form a matrix that does not have the C1P and, (2) whether every maximal proper subset of R (obtained by removing exactly one row) forms a matrix that has the C1P. This requires only $p+1$ C1P tests and can then be done in time $O(p(n+m+e))$, using an efficient algorithm for testing the C1P [19].

Generating one Minimal Conflicting Set. It follows from the link between MCS and cycles in graphs that there can be an exponential number of MCS for a given binary matrix M, and that generating all of them is a hard problem. However, generating one MCS is easy and can be achieved in polynomial time by the following simple greedy algorithm:

1. let $R = \{r_1, \ldots, r_m\}$ be the full set of rows of M;
2. for i from 1 to m, if removing r_i from R results in a set of rows that has the C1P then keep r_i in R, otherwise remove r_i from R;
3. the subset of rows R obtained at the end of this loop is then an MCS.

Generating all Minimal Conflicting Sets. Given M and a list $C = \{R_1, \ldots, R_k\}$ of known MCS, the *sequential generation problem* $Gen_{MCS}(M, C)$ is the following: decide if C contains all MCS of M and, if not, compute one MCS that does not belong to C. Using the obvious property that, if R_i and R_j are two MCS then neither $R_i \subset R_j$ nor $R_j \subset R_i$, Stoye and Wittler [23] proposed the following backtracking algorithm for $Gen_{MCS}(M, C)$:

1. Let M' be defined by removing from M at least one row from each R_i in C by recursing on the elements of C.
2. If M' does not have the C1P then compute an MCS of M' and add it to C, else backtrack to step 1 using another set of rows to remove such that each $R_i \in C$ contains at least one of these rows.

This algorithm can require time $\Omega(n^k)$ to terminate, which, as k can be exponential in n, can be superexponential in n. As far as we know, this is the only previously proposed algorithm to compute all MCS.

3 Deciding if a row is a conflicting row.

We now describe our first result, an algorithm to decide if a row of M is a conflicting row (i.e. has a CI greater than 0). Detecting non-conflicting rows is important, for example to speed-up algorithms that compute an optimal C1P subset of rows of M, or in generating all MCS. Our algorithm has a complexity that is exponential in $\Delta(M)$. It is based on a combinatorial characterization of non-C1P matrices due to Tucker [24].

Tucker patterns. The class of C1P matrices is closed under column and row deletion. Hence there exists a characterization of matrices which do not have the C1P by forbidden minors. Tucker [24] characterizes these forbidden submatrices, called M_I, M_{II} M_{III}, M_{IV} and M_V: if M is binary matrix that does not have the C1P, then it contains at least one of these matrices as a submatrix. We call these forbidden matrices the *Tucker patterns*. Patterns M_{IV} and M_V have each 4 rows, while the other three patterns do not have a fixed number of rows or columns; pattern M_I corresponds to the cycle when $\Delta(M) = 2$.

Bounded patterns. Let P be a set of p rows R of M, that defines a $p \times n$ binary matrix. P is said to *contain exactly* a Tucker pattern M_X if a subset of its columns defines a matrix equal to pattern M_X. The following properties are straightforward from the definitions of Tucker patterns and MCS:

Property 1. (1) If a subset of rows of M is an MCS, then the subset contains exactly a Tucker pattern.
(2) If a subset of p rows of M contains exactly a Tucker pattern M_{II} M_{III}, M_{IV} or M_V, then $4 \leq p \leq \max(4, \Delta(M) + 1)$.

From Property 1.(1), to decide if r_i belongs to at least one MCS, it suffices to decide if it belongs to a set R of p rows of M that contains exactly a Tucker pattern and is an MCS. Moreover, from Property 1.(2), if this Tucker pattern is of type M_{II} to M_V, it can be found by a brute-force approach that considers all sets of at most $\Delta(M)$ rows of R that contain r_i, and test, for each such set of rows, if it is an MCS. This brute-force approach has a worst-case time complexity of $O(\Delta(M)^2 m^{\Delta(M)+1}(n + m + e))$. It also allows to detect if r_i is in an MCS because of a Tucker pattern M_I containing at most $\Delta(M) + 1$ rows, which leaves only the case of patterns M_I (the cycle) with more than $\Delta(M) + 1$ rows.

Arbitrarily long cycles. Let B_M be the bipartite graph defined by M as follows: vertices are rows and columns of M, and every entry 1 in M defines an edge. Pattern M_I with p rows corresponds to a cycle of length $2p$ in B_M. Hence, if R contains M_I with p-row, the subgraph of B_M induced by R contains such a cycle and possibly other edges. Let $C = (r_{i_1}, c_{j_1} \ldots, r_{i_\ell}, c_{j_\ell})$ be a cycle in B_M. We say that a r_{i_q} belonging to C is *blocked in* C if there exists a vertex c_j such $M_{i_q,j} = 1$, $M_{i_{q-1},j} = 1$ ($M_{i_\ell,j} = 1$ if $q = 1$) and $M_{i_{q+1},j} = 1$ ($M_{1,j} = 1$ if $q = \ell$).

Proposition 1. *Let M be a binary matrix that does not have the C1P. Let r_i be a row of M that does not belong to any set R of rows of M that contains exactly a Tucker pattern M_{II} M_{III}, M_{IV} or M_V.*

A subset of p rows of M that contain r_i contains exactly the pattern M_I if and only if r_i belongs to a cycle $C = (r_{i_1}, c_{j_1}, \dots r_{i_\ell}, c_{j,\ell})$ in B_M and r_i is not blocked in C.

Proof. (Sketch) If r_i belongs to an MCS and is not included in any pattern M_{II} M_{III}, M_{IV} or M_V, then it belongs to a set of rows that contain exactly the pattern M_I where r_i is not blocked (otherwise, this would contradict the fact that these p rows form an MCS).

To show that if r_i is not blocked in a cycle C implies it belongs to an MCS, consider a minimal cycle C where r_i is not blocked. A *chord* in the cycle C is a set of two edges (r, c) and (r', c) such that r and r' belong to C but are not consecutive row vertices in this cycle. If C has no chord, then the vertices it contains define a Tucker pattern M_I and then an MCS. Otherwise, the chord defines two cycles that contain r_i and where r_i is blocked, which contradicts the minimality of C. □

The algorithm. To decide whether r_i belongs to an MCS, we can then (1) check all sets of at least 4 and at most $\Delta(M) + 1$ rows that contain r_i to see if they define an MCS, and, if this is not the case, (2) check whether r_i belongs to an MCS due to pattern M_I. For this second case, we only need to find a cycle where r_i is not blocked. This can be done in polynomial time by considering all pairs of possible rows r_{i_1} and r_{i_2} that each have an entry 1 in a column where r_i has an entry 1 (there are at most $O(m^2)$ such pairs of rows), exclude the cases where the three rows r_i, r_{i_1} and r_{i_2} have an entry 1 in the same column, and then check if there is a path in B_M between r_{i_1} and r_{i_2} that does not visit r_i. This leads to the main result of this section.

Theorem 2. *Let M be an $m \times n$ binary matrix that does not have the C1P, and r_i be a row of M. Deciding if r_i belongs to at least one MCS can be done in $O(\Delta(M)^2 m^{\max(4,\Delta(M)+1)}(n+m+e))$ time.*

4 Generating all MCS with monotone boolean functions

Let $[m] = \{1, 2, \dots, m\}$. For a set $I = \{i_1, \dots, i_k\} \subseteq [m]$, we denote by X_I the boolean vector $(x_1, \dots, x_m)$ such that $x_j = 1$ if and only if $i_j \in I$. A *boolean function* $f : \{0,1\}^m \to \{0,1\}$ is said to be *monotone* if for every $I, J \subseteq [m]$, $I \subseteq J \Rightarrow f(X_I) \leq f(X_J)$.

Given a boolean function, a boolean vector X is said to be a *Minimal True Clause* (MTC) if $f(X_I) = 1$ and $f(X_J) = 0$ for every $J \subset I$. Symmetrically, X_I is said to be a *Maximal False Clause* (MFC) if $f(X_I) = 0$ and $f(X_J) = 1$ for every $I \subset J$. We denote by $MTC(f)$ (resp. $MFC(f)$) the set of all MTC (resp. MFC) of f.

For a given $m \times n$ binary matrix M, let $f_M : \{0,1\}^m \to \{0,1\}$ be the boolean function defined by $f_M(X_I) = 1$ if and only if R_I does not have the C1P, where $I \subseteq [m]$. This boolean function is obviously monotone and the following proposition is immediate.

Proposition 2. *Let $I = \{i_1, \dots, i_k\} \subseteq [m]$. R_I is an MCS (resp. MC1P) of M if and only if X_I is an MTC (resp. MFC) for f_M.*

Generating Minimal True Clauses for monotone boolean functions. It follows from Proposition 2 that generating all MCS reduces to generating all MTC for a monotone boolean function. This very general problem has been the subject of intense research, and we describe briefly below some important properties.

Theorem 3. [12] *Let $C = \{X_1, \ldots, X_k\}$ be a set of MTC (resp. MFC) of a monotone boolean function f. The problem of deciding if C contains all MTC (resp. MFC) of f is coNP-complete.*

Theorem 4. [11] *The problem of generating all MTC of a monotone boolean function f using an oracle to evaluate this function can require up to $|MTC(f) + MFC(f)|$ calls to this oracle.*

The second property suggests that, in general, to generate all MTC, it is necessary to generate all MFC, and vice-versa. For example, the algorithm of Stoye and Wittler [23] described in Section 2 is a satisfiability oracle based algorithm – it uses a polynomial-time oracle to decide if a given submatrix has the C1P, but it doesn't use this structure any further. Once it has found the complete list C of MCS, it will proceed to check all MC1P sets as candidate conflicting sets before terminating. Since this does not keep the MC1P sets explicitly, but instead uses backtracking, it may generate the same candidates repeatedly resulting in a substantial duplication of effort. In fact, this algorithm can easily be modified to produce *any* monotone boolean function given by a truth oracle.

Joint Generation of MTC and MFC for monotone boolean functions. One of the major results on generating MTC for monotone boolean functions, is due to Fredman and Khachiyan. It states that generating both sets together can be achieved in time quasi-polynomial in the number of MTC plus the number of MFC.

Theorem 5. [9] *Let $f : \{0,1\}^m \to \{0,1\}$ be a monotone boolean function whose value at any point $x \in \{0,1\}^m$ can be determined in time t, and let C and D be respectively the sets of the MTC and MFC of f. Given two subsets $C' \subseteq C$ and $D' \subseteq D$ of total size $s = |C'| + |D'|$, deciding if $C \cup D = C' \cup D'$, and if $C \cup D \neq C' \cup D'$ finding an element in $(C \backslash C') \cup (D \backslash D')$ can be done in time $O(m(m+t) + s^{o(\log s)})$.*

The key element to achieve this result is an algorithm that tests if two monotone boolean functions are duals of each other (see [8] for a recent survey on this topic). As a consequence, we can then use the algorithm of Fredman and Khachiyan to generate all MCS and MC1P.

5 Experimental results

We used the `cl-jointgen` implementation of the joint generation method which is publicly available [14] with an oracle to test the C1P property based on the algorithm described in [19]. All datasets and results are available at the following URL: `http://www.cecm.sfu.ca/~cchauve/SUPP/RCG09`.

We generated 10 datasets of adjacencies (the rows of the binary matrices contain each two entries 1) with $n = 40$ and $m = 45$. Each dataset contains exactly 39 true positive (rows $\{i, i+1\}$

for $i = 1, \ldots, 39$) and 6 random false positives (rows $\{i, j\}$ with $j > i + 1$). These parameters were chosen to simulate moderately large datasets that resemble real datasets. For a given dataset, the *conflicting ratio* (CR) of a row is the ratio between the CI of this row and the number of MCS. Similarly, the *MC1P ratio* (MR) of a row is the ratio between the number of MC1P that contain the row and the total number of MC1P. The *MCS rank* of a row is its ranking (between 1 and 45) when rows are ranked by increasing CR. The *MC1P rank* of a row is its ranking when rows are ranked by increasing MR. Table 1 shows some statistics on these experiments.

Dataset	Number of MCS	Number of MC1P	Average FP_CR	Average TP_CR	Average FP_MR	Average TP_MR	Average FP MCS rank	Average FP MC1P rank
1	55	8379	0.41	0.37	0.34	0.77	18.83	6.83
2	43	4761	0.36	0.32	0.3	0.84	20.33	6
3	38	9917	0.4	0.22	0.34	0.79	33.17	7
4	46	4435	0.5	0.35	0.41	0.8	33.33	9
5	59	6209	0.44	0.3	0.36	0.76	28.33	6
6	45	13791	0,47	0.2	0.39	0.8	32.67	4.67
7	61	2644	0.44	0.31	0.37	0.8	28.83	5.83
8	50	3783	0.43	0.28	0.36	0.81	34.5	6.83
9	57	2575	0.51	0.37	0.43	0.81	32.83	5.17
10	60	3641	0.45	0.31	0.38	0.83	26.33	7.83

Table 1. Statistics on MCS and MC1P on simulated adjacencies datasets. FP_CR is the Conflicting Ratio for False Positives, TP_CR is for CR the True Positives, FP_MR is the MC1P ratio for False Positives and TP_MR is the MR for True Positives.

First, we can notice the large difference between the number of MCS and the number of MC1P. This shows that most computation time, in the joint generation, is spent generating MC1P. However if, as expected, false positives have, on average, a higher conflicting ratio than true positives, and conversely a lower MC1P ratio than true positives, it is interesting that the MC1P ratio discriminates much better between false positives and true positives than the conflicting ratio. This is seen in the MCS and MC1P ranks: the false positives have an average MCS rank of 28.91, well below the rank that would be expected if they were the rows that have the highest CI (42.17), while they have an average MC1P rank of 6.52, quite close of the 3.5 rank expected if they belonged to the fewest MC1P. To get a better understanding of the usefulness of the MCS ratio and MC1P ratio, Table 2 shows the rough distribution of these ratios.

These result suggest that the increased computation required by generating MC1P brings valuable information in discriminating false positives from true positives, and that the MC1P ratio is a better information to rank rows when trying to compute a maximal MC1P subset of rows.

Dataset	[0, .1]	(.1, .2]	(.2, .3]	(.3, .4]	(.4, .5]	(.5, .6]	(.6, .7]	(.7, .8]	(.8, .9]	(.9, 1]
MCS ALL	52	16	75	205	87	14	1	0	0	0
MCS FP	0	0	14	31	13	2	0	0	0	0
MCS TP	52	16	61	174	74	12	1	0	0	0
MC1P ALL	10	2	7	18	32	50	73	44	20	194
MC1P FP	0	0	0	6	22	44	64	40	20	194
MC1P TP	10	2	7	12	10	6	9	4	0	0

Table 2. Distribution of the MCS and MC1P ratios for all rows (ALL), false positives (FP) and true positives (TP). Each cell of the table contains the number of rows whose ratio is in the interval for the column.

6 Conclusion and perspectives

This paper describes preliminary theoretical and experimental results on Minimal Conflicting Sets and Maximal C1P Sets. In particular, we suggested that Tucker patterns are fundamental in understanding the combinatorics of MCS, and that the generation of all MCS is a hard problem, related to monotone boolean functions. From an experimental point of view it appears, at least on datasets of adjacencies, that MC1P offer a better way to detect false positive ancestral syntenies than MCS and the CI. This leaves several open problems to attack.

Detecting non-conflicting rows. The complexity of detecting rows of a matrix that do not belong to any MCS when rows can have an arbitrary number of entries 1 is still open. Solving this problem probably requires a better understanding of the combinatorial structure of MCS and Tucker patterns. Tucker patterns have also be considered in [7, Chapter 3], where polynomial time algorithms are given to compute a Tucker pattern of a given type for a matrix that does not have the C1P. Even if these algorithms can not obviously be modified to decide if a given row belongs to a given Tucker pattern, they provide useful insight on Tucker patterns.

It follows from the dual structure of monotone boolean functions that the question of whether a row belongs to any MCS is equivalent to the question of whether it belongs to any MC1P. Indeed, for an arbitrary oracle-given function, testing if a variable appears in any MTC is as difficult as deciding if a list of MTC is complete. Consider an oracle-given f and a list of its MTC which define a (possibly different) function f'. We can build a new oracle function g with an additional variable x_0, such that $g(x_0, x) = 1$ if and only if $x_0 = 0$ and $f'(x) = 1$ or $x_0 = 1$ and $f(x) = 1$.

Generating all MCS and MC1P. Right now, this can be approached using the joint generation method, but the number of MCS and MC1P makes this approach impractical for large matrices. A natural way to deal with such problem would be to generate at random and uniformly MCS and MC1P. For MCS, this problem is at least as hard as generating random cycles of a graph, which is known to be a hard problem [16]. We are not aware of any work on the random generation of MC1P. An alternative to random generation would be to abort the joint generation after it generates a large number of MCS and MC1P, but the quality of the approximation of the MCS ratio and MC1P ratio so obtained would not be guaranteed. Another approach for the generation of all MCS could

be based on the remark that, for adjacencies, it can be reduced to generating all claws and cycles of the graph G_M. Generating all cycles of a graph can be done in time that is polynomial in the number of cycles, using backtracking [22]. It is then tempting to use this approach in conjunction with dynamic partition refinement [13] for example or the graph-theoretical properties of Tucker patterns described in [7].

Combinatorial characterization of false positives ancestral syntenies. It is interesting to remark that, with adjacencies datasets, detecting most false positives can be attacked in a simple way. True positive rows define a set of paths in the graph G_M, representing ancestral genome segments, while false positive rows $\{i, j\}$, unless i or j is an extremity of such a path (in which case it does not exhibit any combinatorial sign of being a false positive), both the vertices i and j belong to a claw in the graph G_M. And it is easy to detect all edges in this graph with both ends belonging to a claw. In order to extend this approach to more general datasets, where $\Delta(M) > 2$, it would be helpful to understand better the impact of adding a false positive row in M. The most promising approach would be to start from the *partition refinement* [13] obtained from all true positive rows and form a better understanding of the combinatorial structure of connected components of the overlap graph that do not have the C1P.

Computation speed. The experiments in this paper took at most a few minutes to complete. We are currently running experiments on larger simulated datasets, as well as real data taken from [21], whose results will be made available on the companion Website of this paper. On larger datasets, especially with matrices with an arbitrary number of entries 1 per row, some connected components of the overlap graph can be very large (see the data in [21] for example). In order to speed up the computations, algorithmic design and engineering developments are required, both in the joint generation algorithm and in the problem of testing the C1P for matrices after rows are added or removed.

7 Acknowledgments

Cedric Chauve and Tamon Stephen were partially supported by NSERC Discovery Grants. Utz-Uwe Haus was supported by the Magdeburg Center for Systems Biology, funded by a FORSYS grant of the German Ministry of Education and Research.

References

1. Z. Adam, M. Turmel, C. Lemieux, and D. Sankoff. Common intervals and symmetric difference in a model-free phylogenomics, with an application to streptophyte evolution. *J. Comput. Biol.*, 14, pp. 436–445. 2007.
2. F. Alizadeh, R. Karp, D. Weisser and G. Zweig. Physical mapping of chromosomes using unique probes. *J. Comput. Biol.* 2, pp. 159–184. 1995.
3. A. Bergeron, M. Blanchette, A. Chateau and C. Chauve. Reconstructing ancestral gene orders using conserved intervals. In *WABI 2004*, vol. 3240 of LNCS/LNBI, pp. 14–25. Springer. 2004.
4. K.S. Booth and G.S. Lueker. Testing for the consecutive ones property, interval graphs, and grap h planarity. *J. Comput. Syst. Sci.* 13, pp. 335–379. 1976.

5. C. Chauve and E. Tannier. A methodological framework for the reconstruction of contiguous regions of ancestral genomes and its application to mammalian genome. *PLoS Comput. Biol.* 4, paper e1000234. 2008.
6. T. Christof, M. Jünger, J. Kececioglu, P. Mutzel and G. Reinelt. A branch-and-cut approach to physical mapping of chromosome by unique end-probes. *J. Comput. Biol.* 4, pp. 433–447. 1997.
7. M. Dom. Recognition, Generation, and Application of Binary Matrices with the Consecutive-Ones Property. Dissertation, Institut für Informatik, Friedrich-Schiller-Universität, Jena. 2008.
8. T. Eiter, K. Makino and G. Gottlob. Computational aspects of monotone dualization: A brief survey. *Disc. Appl. Math.* 156, pp. 2035–2049. 2008.
9. M.L. Fredman and L. Khachiyan. On the complexity of dualization of monotone disjunctive normal forms. *J. Algorithms* 21, pp. 618–628. 1996.
10. P.W. Goldberg, M.C. Golumbic, H. Kaplan and R. Shamir Four strikes againts physical mapping of DNA *J. Comput. Biol.* 2, pp. 139–152. 1995.
11. D. Gunopulos, R. Khardon, H. Mannila and H. Toivonen. Data mining, hypergraph transversals and machine learning. In *PODS 1997*, pp. 209–216. ACM. 1997.
12. V. Gurvich and L. Khachiyan. On generating the ireedundant conjunctive and disjunctive normal forms of monotone Boolean functions. *Disc. Appl. Math.* 96–97, pp. 363–373. 1999.
13. M. Habib, R.M. McConnell, C. Paul and L. Viennot. Lex-BFS and partition refinement, with applications to transitive orientation, interval graph recognition and consecutive ones testi ng. *Theoret. Comput. Sci.* 234, pp. 59–84. 2000.
14. U.-U. Haus and T. Stephen. `CL-JOINTGEN`: A Common Lisp Implementation of the Joint Generation Method. available at `http://primaldual.de/cl-jointgen/`. 2008.
15. W.-L. Hsu. A simple test for the Consecutive Ones Porperty. *J. Algorithms* 43, pp. 1–16. 2002.
16. M.R. Jerrum, L.G. Valiant and V.Y. Vazirani. Random generation of combinatorial structures from a uniform distribution. *Theoret. Comput. Sci.* 43, pp. 169–188. 1986.
17. W.-F. Lu and W.-L. Hsu. A test for the Consecutive Ones Property on noisy data – application to physical mapping and sequence assembly. *J. Comp. Biol.* 10, pp. 709–735. 2003.
18. J. Ma *et al.*. Reconstructing contiguous regions of an ancestral genome. *Genome Res.*, 16, pp.1557–1565. 2006.
19. R.M. McConnell. A certifying algorithm for the consecutive-ones property. In *SODA 2004*, pp. 761-770. ACM. 2004.
20. J. Meidanis, O. Porto and G.P. Telle. On the consecutive ones property. *Discrete Appl. Math.* 88, pp. 325–354. 1998.
21. A. Ouangraoua, F. Boyer, A. McPherson, E. Tannier and C. Chauve. Prediction of contiguous ancestral regions in the amniote ancestral genome In *ISBRA 2009*, vol. 5542 of LNCS/LNBI, to appear. Springer. 2009.
22. R.C. Read and R.E. Tarjan. Bounds on backtrack algorithms for listing cycles, paths, and spanning trees. *Networks* 5. pp. 237–252. 1975.
23. J. Stoye and R. Wittler A unified approach for reconstructing ancient gene clusters. To appear in *IEEE/ACM Trans. Comput. Biol. Bioinfo.* 2009.
24. A.C. Tucker. A structure theorem for the consecutive 1's property. *J. Combinat. Theory (B)* 12, pp. 153–162. 1972.
25. L.G. Valiant. The complexity of enumeration and reliability problems. *SIAM J. Comput.* 8, pp. 410–421. 1979.