Finding All Significant Closed Connected Subgraphs

Anonymous Author(s)
Affiliation
Address
`email`

Abstract

In many applications, covariates describing the data are structured according to a known graph $\mathcal{G}$. Subgraphs of $\mathcal{G}$ can then serve to design new covariates, yielding an enriched representation of the data. Testing the association of these new covariates to an outcome of interest can provide more insight on critical biological processes. However, the number of subgraphs is often exponential in the number of original covariates. Therefore, a method testing all possible subgraphs would have very low power, due to multiple testing corrections and could quickly become computationally intractable. The concept of testable hypothesis has been used to simultaneously address both issues in similar contexts. Here, we introduce a method leveraging this concept to test all closed connected subgraphs, i.e., those which are not included in a larger one leading to the exact same covariate. We propose a novel enumeration scheme for these objects which fully exploits the pruning opportunity offered by testability, leading to drastic improvements in speed. We illustrate this improvement on both real and simulated datasets. This paves the way for numerous applications in biomedicine, especially for genome-wide association studies in bacterial genomes.

1 Introduction

Networks are pervasive in molecular biology, and can represent, for instance, gene regulations or interactions between proteins or metabolic pathways. They are also a major opportunity for statistical analysis, as many applications involve few samples and many descriptors, leading to high-dimensional problems. Bacterial Genome-wide association studies (GWAS) is an example. GWAS aim at finding genetic variants whose presence in a genome is associated with a phenotype. When studying bacterial genomes, *e.g.*, to identify genetic determinants of antibiotic resistances, the tested variants are often the presence or absence of k-mers, *i.e.*, words of length k, in the genomes of the samples. However, a gene or a plasmid whose presence in the genome makes bacteria resistant can be longer than k and exist in slightly different version, and therefore be represented by many different k-mers. Jaillard et al. [2018] proposed `DBGWAS`, a method using the De Bruijn graph that connects overlapping k-mers to help interpret the result of the GWAS: if several significant k-mers arise from a single polymorphic gene, they typically aggregate into linear subgraphs. `DBGWAS` exploits the De Bruijn graph for visualization but still relies on a separate statistical test for each k-mer, while testing the association between antibiotic resistance and the presence of any version of the resistance-causing gene could yield more power. This presence would correspond to a single covariate indicating the presence of any k-mer among those represented in the subgraph.

A systematic approach would therefore be to test each connected subgraphs of the De Bruijn graph, but this seems doomed for two reasons: (1) their number grows exponentially with the number of nodes in the network, making the task usually computationally intractable, and (2) adjusting for multiple testing over this very large number of tests leaves little to no power to detect associations. Here we propose a method addressing these two issues by using the concept of testability introduced by Tarone [1990]. Tarone's procedure allows to control the family-wise error rate (FWER) while disregarding a large number of (non-testable) hypotheses in the multiple testing correction. Intuitively, considering the presence of any k-mer among a growing set corresponding to larger and larger connected subgraphs quickly leads to all-one covariates, which cannot be associated to any phenotype, making the actual number of tests more manageable. Testability provides a well-grounded and quantitative version of this intuition. Furthermore, since adding nodes to a subgraph can only increase the number of ones in the tested covariate, we are able to rapidly prune non-testable subgraphs, which solves the computational problem.

Testability has been used in similar situations, but most existing procedures are restricted to complete [Terada et al., 2013, Minato et al., 2014] or linear graph [Llinares-López et al., 2015, 2017]. Sese et al. [2014] described an algorithm to test all connected closed subgraphs (CCSs), *i.e.*, connected subgraphs such that adding any neighbor does not affect the created covariate. They combined the testability-based procedure `LAMP` of Terada et al. [2013] with COIN [Sese et al., 2010], an enumeration method for CCSs. While no experiment was provided in Sese et al. [2014], we found that combining COIN with an improved version of LAMP [Minato et al., 2014, Llinares-López et al., 2015] could find all significant CCSs in graphs with up to 20,000 nodes in a day in favorable settings. However modern applications such as bacterial GWAS involve millions of nodes, so a more scalable method is necessary to make CCSs testing amenable.

Our contributions are the following: We introduce a novel, provably complete and non-redundant enumeration scheme for CCSs named `CALDERA`, that leads to faster exploration than `COIN`, and to more pruning when combined with Tarone's procedure. We show that this makes it possible to find all significant CCSs in a large graph, making it suited to applications such as bacterial GWAS, a critical and contemporary problem for human health. We provide—in the Supplementary material—the first implementation of a procedure finding all significant CCSs.

Notation and goal We consider a set of n samples, $(x_i, y_i)_{i=1}^n$, where $x_i \in \{0,1\}^p$ are p binary covariates describing sample i and $y_i \in \{0,1\}$ denotes a binary phenotype. Furthermore, we consider an undirected unweighted connected graph $\mathcal{G} = (\mathcal{V}, E)$, where $\mathcal{V} = \{v_1, \ldots, v_p\}$ and each vertex $v_j \in \mathcal{V}$ represents one of the p binary covariates represented in x. We denote by $\mathcal{I}(v_j) = \{i : x_i^j = 1\}$ the set of samples having a 1 for covariate represented by vertex v_j, and $\mathbb{V}_i = \{v \in \mathcal{V} : i \in \mathcal{I}(v)\}$ the set of vertices whose covariate is 1 for sample i. For any connected subgraph $\mathcal{S} = (\mathcal{V}', E')$, such that $\mathcal{V}' \subseteq \mathcal{V}$ and $E' \subseteq E$, we let $\mathcal{I}(\mathcal{S}) = \bigcup_{v \in \mathcal{V}'} \mathcal{I}(v)$. The set of all connected subgraphs of $\mathcal{G}$ is denoted by $\mathcal{A}$. Of note, this framework addresses both disjunctions and conjunctions, as the latter can simply be obtained by replacing each x_i by its complement. We now properly define the notion of closed connected subgraph. The validity of the corresponding closure operation is proved in Supplementary S-1.1.

Definition 1. *A connected subgraph $\mathcal{S} \in \mathcal{A}$ of $\mathcal{G} = (\mathcal{V}, E)$ is closed if and only if there exists no edge $(v_1, v_2) \in E$ such that $v_1 \in \mathcal{S}$, $v_2 \notin \mathcal{S}$, and $\mathcal{I}(\mathcal{S} \bigcup \{v_2\}) = \mathcal{I}(\mathcal{S})$. We denote by $\mathcal{C} \subseteq \mathcal{A}$ the set of all closed connected subgraphs of $\mathcal{G}$.*

Considering $(x_i, y_i)_{i=1}^n$ n i.i.d. realizations of random variables $\mathbf{X}, \mathbf{Y}$, we aim to test null hypotheses of the form $H_0^{\mathcal{S}}(\mathbf{X}, \mathbf{Y}) : \big(\mathcal{I}(S) \perp \mathbf{Y}\big)$ for all $\mathcal{S} \in \mathcal{C}$, while controlling the FWER at level α. Translated in the context of GWAS, we want to test the association between the pattern $\mathcal{I}(\mathcal{S})$ of each closed connected subgraph $\mathcal{S}$ with the phenotype $\mathbf{Y}$.

2 Speeding up the detection of all significant CCSs with `CALDERA`

2.1 Tarone's testability

The Bonferroni correction [Bonferroni, 1936] controls the family-wise error rate (FWER) at a level α. A null hypothesis is rejected if its p-value is smaller than $\frac{\alpha}{N}$, where N is the total number of tested null hypotheses. As described in Tarone [1990], discrete tests admit a deterministic minimal attainable p-value $p^\star$, which can be used to control the FWER with a substantially smaller correction factor than N. Defining $m(k)$ as the number of hypotheses such that $p^\star < \frac{\alpha}{k}$, the lowest threshold guaranteeing that the FWER is controlled at a level α is $\frac{\alpha}{k_0}$, where k_0 is the smallest k such that $m(k) \leq k$. Provided that enough closed connected subgraphs have sufficiently large $p^\star$, Tarone's procedure could therefore solve the multiple testing issue caused by exploring $\mathcal{C}$. Importantly, non-exhaustive strategies have been proposed to determine k_0, by exploiting a monotonicity property of $p^\star$, *i.e.*, $p^\star(\mathcal{S}) \leq p^\star(\mathcal{S}')$ for any $\mathcal{S} \subseteq \mathcal{S}'$. If $\mathcal{S}$ is non-testable, all $\mathcal{S}' \supseteq \mathcal{S}$ can be discarded without being processed, making the procedure tractable provided that subgraphs are explored in the right order. More precisely as highlighted by Minato et al. [2014], Llinares-López et al. [2015], all subgraphs $\mathcal{S}'$ explored from a subgraph $\mathcal{S}$ should be such that $\mathcal{S}' \supsetneq \mathcal{S}$.

2.2 Critical properties for a fast, Tarone-aware enumeration of $\mathcal{C}$

The testing procedure based on testability relies on an exploration of the set of hypotheses—in our setting, one for each element of $\mathcal{C}$. The scalability of the testing procedure is affected by both the computational behavior—speed and memory footprint–of the exploration scheme itself, and its ability to take advantage of the pruning opportunity offered by the Tarone procedure. To provide a fast exploration, we ensure that it is non-redundant: each element of $\mathcal{C}$ is enumerated exactly once. To do this, we define a tree structure whose nodes are the elements of $\mathcal{C}$ and propose an algorithm to traverse this tree. Furthermore, the tree is directly built over $\mathcal{C}$, as opposed to the set $\mathcal{A} \supset \mathcal{C}$ of connected subgraphs. This latter option is found on the `COIN` algorithm described in Seki and Sese [2008], Sese et al. [2010], which builds a tree over the set of connected subgraphs. This yields a much larger object and results in a slower traversal. Furthermore in order to exploit the pruning opportunity offered by the testing procedure, our tree over $\mathcal{S}$ is such that the children of a node representing $\mathcal{S} \in \mathcal{C}$ always represent subgraphs $\mathcal{S}' \subsetneq \mathcal{S}$. While Haraguchi et al. [2019], Okuno et al. [2017]

define a tree on $\mathcal{C}$, the root of that tree corresponds to the entire graph $\mathcal{G}$: the inclusion relationship along edges of the tree is the opposite to the one we need, making this unsuited to our problem.

2.3 Defining and exploring the tree over $\mathcal{C}$

To build a tree over $\mathcal{C}$ rooted on the empty CCS, we use a reverse search [Avis and Fukuda, 1993]. Reverse search relies on a reduction operation, which takes one element of the set to be enumerated, and returns a unique, strictly smaller element of the same set called its parent. This operation necessarily defines a tree over the elements of the set, by ensuring a unique path between any element and the empty one—the root of the tree. In order to traverse the tree from the root, one needs to inverse the reduction operation: given a CCS $\mathcal{S}$, this would recover all CCSs that lead to $\mathcal{S}$ by reduction. Here we introduce a reduction operation over $\mathcal{C}$, as well as its inversion. We rely on an arbitrary numbering of the vertices in $\mathcal{V}$, and denote by abuse of notation $\max \mathcal{S}$ the vertex in $\mathcal{S}$ that received the largest number.

Definition 2. *For a subgraph $\mathcal{S} \in \mathcal{C}$, we denote $\mathcal{J}(\mathcal{S}) = \bigcap_{v \in \mathcal{S}} \mathcal{I}(v)$. We note $i_{\mathcal{S}} = \max(\mathcal{I}(\mathcal{S}) \setminus \mathcal{J}(\mathcal{S}))$. The parent $\mathcal{P}(\mathcal{S})$ of $\mathcal{S}$ is the connected subgraph of $\mathcal{S} \setminus \mathbb{V}_{i_{\mathcal{S}}}$ that contains $\max \mathcal{S} \setminus \mathbb{V}_{i_{\mathcal{S}}}$. (If $\mathcal{I}(\mathcal{S}) = \mathcal{J}(\mathcal{S})$, then the parent of $\mathcal{S}$, $\mathcal{P}(\mathcal{S})$ is $\emptyset$.)*

Lemma 1. *The function $\mathcal{P}$ defines a valid reduction over $\mathcal{C}$.*

Algorithm 1 Children of $\mathcal{S}$

```
 1: procedure CHILDREN($\mathcal{S}$, $\mathcal{S}_p$, $i$, $\mathcal{T}$)
 2:     children← $\emptyset$
 3:     for $k, G$ in enumerate(EqGroups($\mathcal{S}$)) do
 4:         $v \leftarrow G[0]$
 5:         $\mathcal{S}' \leftarrow cl(\mathcal{S} \bigcup \{v\})$
 6:         if $i$ is NULL then
 7:             if $(\mathcal{S}, \mathcal{S}')$ verify (C1-C3) then
 8:                 Add $\mathcal{S}'$ to siblings
 9:                 Add Children($\mathcal{S}', \mathcal{S}, i_{\mathcal{S}'}, \mathcal{T} = \emptyset$) to children
10:             end if
11:         else if $(\mathcal{S}_p, \mathcal{S}')$ verify (C1-C3) then
12:             if $i_{\mathcal{S}'} = i$ and $\{\mathcal{I} \in \mathcal{T} : \mathcal{I} \subset \mathcal{I}(\mathcal{S}')\} = \emptyset$ then
13:                 $\mathcal{T}' = \mathcal{T} \bigcup \{\mathcal{I}_1(\mathcal{S}'), \ldots, \mathcal{I}_{k-1}(\mathcal{S}')\}$
14:                 Add Children($\mathcal{S}', \mathcal{S}_p, i_{\mathcal{S}'}, \mathcal{T}'$) to children
15:             end if
16:         end if
17:     end for
18:     return children
19: end procedure
```

Note that we have $\mathcal{S} \supsetneq \mathcal{P}(\mathcal{S})$ for all $\mathcal{S}$ so this structure allows pruning. For any subgraph $\mathcal{S}$, we further note $Ne(\mathcal{S}) = \{v \in \mathcal{G} \setminus \mathcal{S} : \exists v_1 \in \mathcal{S}, (v, v_1) \in E\}$ the set of neighbouring nodes of $\mathcal{S}$. Lemma 2 provides necessary and sufficient conditions for $\mathcal{S}' \in \mathcal{C}$ to be a child of $\mathcal{S} \in \mathcal{C}$:

Lemma 2. *For $\mathcal{S}, \mathcal{S}' \in \mathcal{C}$ such that $\mathcal{S} \subset \mathcal{S}'$, $\mathcal{S} = \mathcal{P}(\mathcal{S}')$ if and only if the three following conditions are verified:*

(C1) $i_{\mathcal{S}'} \notin \mathcal{I}(\mathcal{S})$

(C2) $\max \mathcal{S}' \setminus \mathbb{V}_{i_{\mathcal{S}'}} = \max \mathcal{S}$

(C3) $\{v' \in \mathcal{S}' \setminus \mathbb{V}_{i_{\mathcal{S}'}} : v' \in Ne(\mathcal{S})\} = \emptyset$

Interestingly, the reduction itself is never used when exploring the tree from the root, only its inverse. Besides, using (C1–3) in Lemma 2 to check whether $\mathcal{S} = \mathcal{P}(\mathcal{S}')$ for any $\mathcal{S}'$ does not require to identify the connected components of $\mathcal{S}' \setminus \mathbb{V}_{i_{\mathcal{S}'}}$, even though the reduction $\mathcal{P}$ itself does rely on these connected components. This property of the inverse reduction is critical for the scalability of `CALDERA`, as repeatedly identifying or maintaining these components would be very costly. It results from the fact that the reduction operation $\mathcal{P}$ does not maintain connectivity—it only retains one of the components obtained by removing nodes with $i_{\mathcal{S}}$. Doing so comes at a price: finding the children of $\mathcal{S}$ is not straightforward, as we must identify and reconnect all the connected components involved. By Theorem 1, Algorithm 1 solves this problem and effectively inverts the reduction, therefore of building a tree over $\mathcal{C}$.

Theorem 1. *For any $\mathcal{S} \in \mathcal{C}$, Algorithm 1 returns the set $\{\mathcal{S}' \in \mathcal{C} : \mathcal{S} = \mathcal{P}(\mathcal{S}')\}$.*

Algorithm 1 exploits a partition of $Ne(\mathcal{S})$ into equivalence groups $G_k(\mathcal{S})$ with regard to the pattern, *i.e.*, $v_1, v_2 \in G_k(\mathcal{S}) \implies \mathcal{I}(\mathcal{S} \bigcup \{v_1\}) = \mathcal{I}(\mathcal{S} \bigcup \{v_2\})$. $\mathcal{I}_k(\mathcal{S})$ denotes the pattern of the equivalence group $G_k(\mathcal{S})$. Note that in practice, we do not need to store the full table $\mathcal{T}$ in order to verify the second condition of Algorithm 1, Line 12 (see Supplementary S-2).

2.4 A breadth-first-search enumeration

Exploring any tree structure on $\mathcal{C}$ in breadth first often allows for more pruning than in depth first. Previous work, including Llinares-López et al. [2017], have used BFS but did not specifically highlight its interest. Note that `COIN` performs a depth-first search [Sese et al., 2014]. Here, we implemented both versions of `CALDERA` to show the gains of BFS. This is evidenced in the simulation results. Supplementary section S-3 provides a general intuition for this result. Algorithm S-1 describes a general implementation of the BFS enumeration of all elements of $\mathcal{C}$ while implementing the pruning mechanism described above. Moreover, a search in breadth is also easily parallelized since the computation of the minimal p-value and the children of every CCS of a given level can be done in parallel, contrary to DFS.

3 Experiments

3.1 Speed benchmark on simulated data

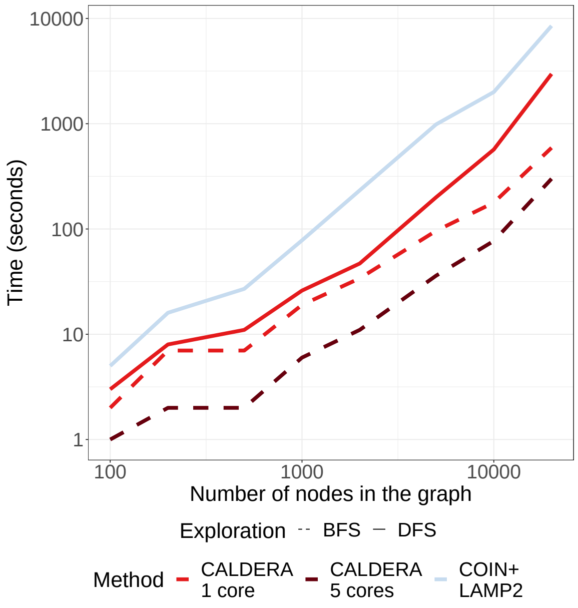


Figure 1: Runtimes for `CALDERA` and `COIN+LAMP` on graphs with various values of covariates p.

Benefit of `CALDERA`'s exploration scheme We generate datasets with $n = 50$ samples represented by $p \in [100 : 20000]$ covariates, and a graph connecting these covariates, to test the speed of our algorithm. As a baseline, we include an improvement on Sese et al. [2014], by combining `COIN` with the improved `LAMP` algorithm of [Minato et al., 2014]. Since `CALDERA` and `COIN+LAMP2` both rely on the same statistical procedures (the identification of testable hypotheses with Fisher's test), the set of significant hypotheses is the same regardless of the method. We provide more details on the simulation procedure in Supplementary S-4. In addition to `COIN+LAMP2`, we benchmark 3 versions of `CALDERA`. The first one, closest to `COIN+LAMP2`, is the DFS implementation. The second one is the BFS implementation, where we modify the enumeration order of the elements of $\mathcal{C}$ to promote pruning. The last is a parallelized BFS implementation, using 5 cores. The ranking in speed is uniform over all value of p, with `COIN+LAMP2` being the slowest, followed by the DFS and BFS implementation, and finally the parallelized version of `CALDERA`. For $p = 20000$, `COIN+LAMP2` takes 2h20 to run while the parallelized version of `CALDERA` took 5 minutes. Others simulation settings (see Supplementary S-4) provide the same speed ranking. For example, if $n = 100$, `COIN+LAMP2` times out (two days threshold) before finishing while the parallelized version of `CALDERA` runs in 6 hours. Over all parameter values, the average ratio of runtime for `COIN+LAMP2` over `CALDERA BFS` with 5 cores is 76. More details on memory usage and simulations settings can be found in section S-4.

3.2 Bacterial GWAS

We consider the $n = 280$ *Pseudomonas Aeruginosa* genomes used in Jaillard et al. [2018], along with their amikacin resistance phenotype. The De Bruijn graph is constructed using $k = 31$-mers, leading to a graph with over 2.3 million nodes. The full exploration of $\mathcal{C}$ is not computationally feasible, even for `CALDERA`. We therefore limited our search to the first 5 stages of the tree constructed on $\mathcal{C}$. Exploring that space took approximately 5 hours to `CALDERA` with 4 cores. This search identified $k_0 = 2.8 \times 10^6$ testable subgraphs for an FWER level $\alpha = 10^{-8}$. 35 of the testable subgraphs were significantly associated to amikacin resistance at this FWER level. We restricted ourselves to the 17 that were not fully included in another significant subgraph, and annotated the corresponding k-mers using blast [Altschul et al., 1990] against both the NCBI database and a resistance database provided with `DBGWAS`. The two subgraphs with lowest p-values are the only two confirmed resistance determinants identified by `DBGWAS`. `DBGWAS` identified these determinants—as its first and third hits—by testing individual k-mers and heuristically adding their neighbors. `CALDERA`, on the other hand, allows inference on the subgraph itself—corresponding to an entire gene or plasmid, paving the way for more powerful and principled bacterial GWAS. `COIN+LAMP2` would return the same result as `CALDERA`, but was still exploring the tree structure with a value of $k = 2.8 \times 10^5$ (a tenth of the final value) after running for 9 days.

4 Discussion

This article presented `CALDERA`, an algorithm to enumerate all significant closed connected subgraphs. `CALDERA` scales to large datasets, relying on an efficient structure on $\mathcal{C}$ and an exploration scheme that leverages the pruning opportunity offered by discrete statistics. Future work will focus on incorporating pre-processing schemes before `CALDERA` that could compact the graph to both reduce its size and facilitate pruning by increasing the average $|\mathcal{I}(v_j)|$.

References

Magali Jaillard, Leandro Lima, Maud Tournoud, Pierre Mahé, Alex van Belkum, Vincent Lacroix, and Laurent Jacob. A fast and agnostic method for bacterial genome-wide association studies: Bridging the gap between k-mers and genetic events. *PLoS genetics*, 14(11):e1007758, 2018. ISSN 1553-7404. doi: 10.1371/journal.pgen.1007758. URL `http://www.ncbi.nlm.nih.gov/pubmed/30419019`.

R. E. Tarone. A Modified Bonferroni Method for Discrete Data. *Biometrics*, 46(2):515, jun 1990. ISSN 0006341X. doi: 10.2307/2531456.

Aika Terada, Mariko Okada-Hatakeyama, Koji Tsuda, and Jun Sese. Statistical significance of combinatorial regulations. *Proceedings of the National Academy of Sciences of the United States of America*, 110(32): 12996–13001, aug 2013. doi: 10.1073/pnas.90.1.203.

Shin Ichi Minato, Takeaki Uno, Koji Tsuda, Aika Terada, and Jun Sese. A fast method of statistical assessment for combinatorial hypotheses based on frequent itemset enumeration. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 8725 LNAI, pages 422–436. Springer Verlag, 2014. ISBN 9783662448502. doi: 10.1007/978-3-662-44851-9_27.

Felipe Llinares-López, Dominik G. Grimm, Dean A. Bodenham, Udo Gieraths, Mahito Sugiyama, Beth Rowan, and Karsten Borgwardt. Genome-wide detection of intervals of genetic heterogeneity associated with complex traits. *Bioinformatics*, 31(12):i240–i249, 2015. ISSN 14602059. doi: 10.1093/bioinformatics/btv263.

Felipe Llinares-López, Laetitia Papaxanthos, Dean Bodenham, Damian Roqueiro, and Karsten Borgwardt. Genome-wide genetic heterogeneity discovery with categorical covariates. *Bioinformatics*, 33(12):1820–1828, 2017. ISSN 14602059. doi: 10.1093/bioinformatics/btx071.

Jun Sese, Aika Terada, Yuki Saito, and Koji Tsuda. Statistically significant subgraphs for genome-wide association study. *SDM*, 47:1–7, 2014.

Jun Sese, Mio Seki, and Mutsumi Fukuzaki. Mining networks with shared items. In *International Conference on Information and Knowledge Management, Proceedings*, pages 1681–1684, New York, New York, USA, 2010. ACM Press. ISBN 9781450300995. doi: 10.1145/1871437.1871703. URL `http://portal.acm.org/citation.cfm?doid=1871437.1871703`.

CE Bonferroni. Teoria Statistica Delle Classi e Calcolo Delle Probabilità. *Pubblicazioni del R Istituto Superiore di Scienze Economiche e Commerciali di Firenze*, 8:3–62, 1936. doi: 10.4135/9781412961288.n455.

Mio Seki and Jun Sese. Identification of active biological networks and common expression conditions. In *8th IEEE International Conference on BioInformatics and BioEngineering, BIBE 2008*, 2008. ISBN 9781424428458. doi: 10.1109/BIBE.2008.4696746.

Kazuya Haraguchi, Yusuke Momoi, Aleksandar Shurbevski, and Hiroshi Nagamochi. COOMA: A components overlaid mining algorithm for enumerating connected subgraphs with common itemsets. *Journal of Graph Algorithms and Applications*, 23(2):434–458, 2019. ISSN 15261719. doi: 10.7155/jgaa.00497. URL `http://jgaa.info/vol`.

Shingo Okuno, Tasuku Hiraishi, Hiroshi Nakashima, Masahiro Yasugi, and Sese Jun. Parallelization of extracting connected subgraphs with common itemsets in distributed memory environments. *Journal of Information Processing*, 25(3):256–267, 2017. ISSN 18826652. doi: 10.2197/ipsjjip.25.256.

David Avis and Komei Fukuda. Reverse search for enumeration. *Discrete Applied Mathematics*, 65:21–46, 1993.

S. Altschul, W. Gish, W. Miller, E. Myers, and D. Lipman. Basic local alignment search tool. *Journal of Molecular Biology*, 215:403–410, 1990.

T Uno, M Kiyomi, and H Arimura. Efficient mining algorithms for frequent/closed/maximal itemsets. In *IEEE ICDM Workshop on Frequent Itemset Mining Implementations*, 2004.

R3: Announcing the next generation of Amazon EC2 Memory-optimized instances, 2020. URL `https://aws.amazon.com/about-aws/whats-new/2014/04/10/r3-announcing-the-next-generation-of-amazon-ec2-memory-optimized-instances/`.

S-1 Proofs

S-1.1 Lemma 3: correctness of the closure

Lemma 3 provides that the operator cl is well defined on connected subgraphs.

Lemma 3. *For any connected subgraph $\mathcal{S}$ of $\mathcal{G}$, there exists a unique subgraph $\mathcal{S}' \in \mathcal{C}$ such that $\mathcal{I}(\mathcal{S}) = \mathcal{I}(\mathcal{S}')$ and $\mathcal{S} \subseteq \mathcal{S}'$.*

Proof of Lemma 3 First let's show that there exists $\mathcal{S}' \in \mathcal{C}$ such that $\mathcal{I}(\mathcal{S}) = \mathcal{I}(\mathcal{S}')$ and $\mathcal{S} \subseteq \mathcal{S}'$. Let $\mathcal{S}'$ be a (inclusionwise) maximal connected subgraph containing $\mathcal{S}$ and such that $\mathcal{I}(\mathcal{S}) = \mathcal{I}(\mathcal{S}')$. By maximality of $\mathcal{S}'$, for every edge $(v_1, v_2) \in E$ with $v_1 \in \mathcal{S}'$ and $v_2 \notin \mathcal{S}'$, we have $\mathcal{I}(\mathcal{S}' \cup \{v_2\}) \neq \mathcal{I}(\mathcal{S}) = \mathcal{I}(\mathcal{S}')$, thus $\mathcal{S}' \in \mathcal{C}$.

Now let's show that such a subgraph is unique. Assume that there exits two different subgraphs $\mathcal{S}_1$ and $\mathcal{S}_2$ in $\mathcal{C}$ such that $\mathcal{S} \subseteq \mathcal{S}_1$ and $\mathcal{S} \subseteq \mathcal{S}_2$ with $\mathcal{I}(\mathcal{S}) = \mathcal{I}(\mathcal{S}_1) = \mathcal{I}(\mathcal{S}_2)$. Since $\mathcal{S}_1 \neq \mathcal{S}_2$, at least one of the subgraphs $\mathcal{S}_1 \setminus \mathcal{S}_2$ and $\mathcal{S}_2 \setminus \mathcal{S}_1$ is not empty. Assume without loss of generality that $\mathcal{S}_1 \setminus \mathcal{S}_2 \neq \emptyset$. Since $\mathcal{S}_1$ is connected and since $\mathcal{S}_1 \cap \mathcal{S}_2 \supseteq \mathcal{S} \neq \emptyset$, there is at least one edge (u, v) with $u \in \mathcal{S}_1 \cap \mathcal{S}_2$ and $v \in \mathcal{S}_1 \setminus \mathcal{S}_2$. This leads to a contradiction since the edge (u, v) is such that $u \in \mathcal{S}_2$, $v \notin \mathcal{S}_2$ and $\mathcal{I}(\mathcal{S}_2 \cup v) = \mathcal{I}(\mathcal{S}) = \mathcal{I}(\mathcal{S}_2)$, which is in contradiction with $\mathcal{S}_2 \in \mathcal{C}$.

S-1.2 Lemma 1: $\mathcal{P}$ is a valid reduction

Case if $\mathcal{I}(\mathcal{S}) = \mathcal{J}(\mathcal{S})$: Then, either $\mathcal{S} = \emptyset$ which has trivially no parent by this reduction. Or all nodes of $\mathcal{S}$ contain exactly the same pattern. For any $v \in \mathcal{S}$, $\mathcal{S} = cl(v)$. $\mathcal{S}$ is a root of our exploration. Its parent is $\emptyset \subseteq \mathcal{S}$. Note that, to avoid enumerating those roots more than once, we only start from $v_{\max} = \max \mathcal{S}$.

Case if $i_{\mathcal{S}}$ is defined: Then, $i_{\mathcal{S}} \in \mathcal{I}(\mathcal{S})$ so $\mathcal{S} \bigcap \mathbb{V}_{i_{\mathcal{S}}} \neq \emptyset$ and $i_{\mathcal{S}} \notin \mathcal{I}(\mathcal{S})$ so $\mathcal{S} \setminus \mathbb{V}_i \neq \emptyset$. Therefore, there is at least one connected component in $\mathcal{S} \setminus \mathbb{V}_i$. Moreover, any connected component of $\mathcal{S} \setminus \mathbb{V}_i$ is included but not equal to $\mathcal{S}$. From [Haraguchi et al., 2019], Lemma 1, we know that, if $\mathcal{S} \in \mathcal{C}$, any connected component of $\mathcal{S} \setminus \mathbb{V}_i$ is also in $\mathcal{C}$. So any connected component of $\mathcal{S} \setminus \mathbb{V}_i$ can be defined as a parent of $\mathcal{S}$. To identify a unique parent, we select the one with the highest node number, $\mathcal{S}_p$. This proves that reduction defines a unique parent. It is a strictly smaller subgraph by inclusion. Indeed, note that since $\mathcal{S} \setminus \mathbb{V}_i \neq \emptyset$ and $\mathcal{S}_p \subset \left(\mathcal{S} \bigcap \mathbb{V}_i\right)$, then $\mathcal{S}_p \subsetneq \mathcal{S}$.

S-1.3 Lemma 2: conditions (C1-3) are necessary and sufficient for $\mathcal{S} = \mathcal{P}(\mathcal{S}')$

S-1.3.1 Proof that for any $\mathcal{S}'$, $(\mathcal{S} = \mathcal{P}(\mathcal{S}'), \mathcal{S}')$ verify $(C1-3)$

$\mathcal{S} \subset \mathcal{S}' \setminus \mathbb{V}_{i_{\mathcal{S}'}}$ so $i_{\mathcal{S}'} \notin \mathcal{I}(\mathcal{S})$. This proves (1). $\max\{v' \in \mathcal{S}' \setminus \mathbb{V}_{i_{\mathcal{S}'}}\} \in \mathcal{S}$ by construction of the parent so $\max\{v' \in \mathcal{S}' \setminus \mathbb{V}_{i_{\mathcal{S}'}}\} \leq \max \mathcal{S}$. Moreover, $\mathcal{S} \subset \mathcal{S}' \setminus \mathbb{V}_{i_{\mathcal{S}'}}$ so $\max \mathcal{S} \leq \max\{v' \in \mathcal{S}' \setminus \mathbb{V}_{i_{\mathcal{S}'}}\}$. So $\max\{v' \in \mathcal{S}' \setminus \mathbb{V}_{i_{\mathcal{S}'}}\} = \max \mathcal{S}$, this proves (2).

Suppose (3) is false. Then, we have $v \in Ne(\mathcal{S}) \bigcap (\mathcal{S}' \setminus \mathbb{V}_{i_{\mathcal{S}'}})$. $\mathcal{S}_2 = cl(\mathcal{S} \bigcup \{v\}) \subset \mathcal{S}'$, $\max \mathcal{S}_2 = \max\{v' \in \mathcal{S}' \setminus \mathbb{V}_{i_{\mathcal{S}'}}\}$ since $\mathcal{S} \subset \mathcal{S}_2$ and $\mathcal{S}_2 \bigcap \mathbb{V}_{i_{\mathcal{S}'}} = \emptyset$ so $\mathcal{S}_2 \subset \mathcal{P}(\mathcal{S}')$. But $\mathcal{S}2 \supsetneq \mathcal{S} = \mathcal{P}(\mathcal{S})$. This is not possible. So (3) is true.

This proves the implication in the first sense.

S-1.3.2 Proof that for any $(\mathcal{S}, \mathcal{S}')$ that verify $(C1-3)$, $\mathcal{S} = \mathcal{P}(\mathcal{S}')$

We consider two closed connected subgraph $\mathcal{S}, \mathcal{S}' \in \mathcal{C}$ that verify (1-3). We want to prove that $\mathcal{P}(\mathcal{S}') = \mathcal{S}$. Point (1) insures that $\mathcal{S} \subseteq (\mathcal{S}' \setminus \mathbb{V}_{i_{\mathcal{S}'}})$. Since $\mathcal{S} \in \mathcal{C}$ and contains the maximal node (from (2)), this ensures that $\mathcal{S} \subseteq \mathcal{P}(\mathcal{S}')$.

Suppose $\mathcal{S} \subsetneq \mathcal{P}(\mathcal{S}')$. Then, $\mathcal{P}(\mathcal{S}') \setminus \mathcal{S} \neq \emptyset$. In particular, since $\mathcal{S}$ and $\mathcal{P}(\mathcal{S}')$ are both connected subgraphs, there exists $v' \in \left(\mathcal{P}(\mathcal{S}') \setminus \mathcal{S}\right) \bigcap Ne(\mathcal{S})$. Since this neighbour is in $\mathcal{P}(\mathcal{S}')$, it is also in $\mathcal{S}' \setminus \mathbb{V}_{i_{\mathcal{S}'}}\}$. That is impossible from (3). So $\mathcal{S} = \mathcal{P}(\mathcal{S}')$. (Note that point (3) includes the fact that $i_{\mathcal{S}'} \in \mathcal{I}(v)$).

This proves the converse implication.

S-1.4 Theorem 1: Algorithm 1 correctly inverts the reduction

We consider a subgraph $\mathcal{S}' \in \mathcal{S}$ and its parent $\mathcal{S} = \mathcal{P}(\mathcal{S}')$.

We first show two lemmas

Lemma 4. *For two subgraphs $\mathcal{S}_1, \mathcal{S}_2 \in \mathcal{C}$, if $\mathcal{S}_1 \subset \mathcal{S}_2$, then $i_{\mathcal{S}_1} \leq i_{\mathcal{S}_2}$.*

Proof:

$$\begin{aligned}
\mathcal{S}_1 \subset \mathcal{S}_2 &\implies \mathcal{I}(\mathcal{S}_1) \subset \mathcal{I}(\mathcal{S}_2) && \text{and} \quad (1)\\
\mathcal{S}_1 \subset \mathcal{S}_2 &\implies \mathcal{J}(\mathcal{S}_2) \subset \mathcal{J}(\mathcal{S}_1) && (2)\\
\text{(1) and (2)} &\implies (\mathcal{I}(\mathcal{S}_1) \setminus \mathcal{J}(\mathcal{S}_1)) \subset (\mathcal{I}(\mathcal{S}_2) \setminus \mathcal{J}(\mathcal{S}_2)) && (3)\\
&\implies i_{\mathcal{S}_1} \leq i_{\mathcal{S}_2} && (4)
\end{aligned}$$

Lemma 5. *For a subgraph $\mathcal{S}' \in \mathcal{C}$ such that $\mathcal{S} = \mathcal{P}(\mathcal{S}') \neq \emptyset$, any subgraph $\mathcal{S}_2 \in \mathcal{C}$ that verifies:*

- $\mathcal{S} \subsetneq \mathcal{S}_2$
- $\mathcal{S}_2 \subset \mathcal{S}'$

is a child of $\mathcal{S}$, that is $\mathcal{P}(\mathcal{S}_2) = \mathcal{S}$

Proof: We know that $\mathcal{S} \subsetneq \mathcal{S}_2$ so $Ne(\mathcal{S}) \bigcap \mathcal{S}_2 \neq \emptyset$. Since $\mathcal{S}_2 \subset \mathcal{S}'$, $Ne(\mathcal{S}) \bigcap \mathcal{S}_2 \subset \mathcal{S}'$ so, from (3) for $\mathcal{S}, \mathcal{S}'$, we have $Ne(\mathcal{S}) \bigcap \mathcal{S}_2 \subset \mathbb{V}_{i_{\mathcal{S}'}}$. So $i_{\mathcal{S}'} \in \mathcal{I}(\mathcal{S}_2)$. $i_{\mathcal{S}'} \notin \mathcal{I}(\mathcal{S})$ so $i_{\mathcal{S}'} \notin \mathcal{J}(\mathcal{S}_2)$. Therefore, $i_{\mathcal{S}'} \leq i_{\mathcal{S}_2}$. But since $\mathcal{S}_2 \subset \mathcal{S}'$, $i_{\mathcal{S}'} \geq i_{\mathcal{S}_2}$. So $i_{\mathcal{S}'} = i_{\mathcal{S}_2}$. Then, we know that $\mathcal{S}, \mathcal{S}_2$ verifies (1). Since $\mathcal{S} \subsetneq \mathcal{S}_2$, we also have (2). Finally $\{v' \in \mathcal{S}_2 \setminus \mathbb{V}_{i_{\mathcal{S}_2}} : v' \in Ne(\mathcal{S})\} = \{v' \in \mathcal{S}_2 \setminus \mathbb{V}_{i_{\mathcal{S}'}} : v' \in Ne(\mathcal{S})\} \subset \{v' \in \mathcal{S}' \setminus \mathbb{V}_{i_{\mathcal{S}'}} : v' \in Ne(\mathcal{S})\} = \emptyset$. This proves (3). Since we have (1-3), we know that $\mathcal{P}(\mathcal{S}_2) = \mathcal{S}$.

Main proof: Now let us prove the main result: Assume that we cannot generate $\mathcal{S}'$ with the procedure from algorithm 1. Let's then consider the largest $\mathcal{S}'' \subsetneq \mathcal{S}'$ generated with the algorithm 1, that is the one with the largest number of nodes. Since $\mathcal{S} = \mathcal{P}(\mathcal{S}')$, we at least have $\mathcal{S}_d \subset \mathcal{S}'$ so at minimum we can take $\mathcal{S}'' = \mathcal{S}_d$.

By assumption, $\mathcal{S}'' \subsetneq \mathcal{S}'$. Therefore, there exists a neighbour $v \in Ne(\mathcal{S}'') \bigcap \mathcal{S}'$ since $\mathcal{S}'$ and $\mathcal{S}''$ are connected subgraphs. Note that we know that $i_{\mathcal{S}''} = i_{\mathcal{S}'}$ by construction. Moreover, $(\mathcal{S}, \mathcal{S}_2 = cl(\mathcal{S}'' \bigcup \{v\}))$ verify (1-3) by Lemma 2.

Case 1: $\mathcal{I}(cl(\mathcal{S}'' \bigcup \{v\}))$ does not include any pattern of `forbidden` $(\mathcal{S}'')$: Since $\mathcal{S}_2 \subset \mathcal{S}'$, $i_{\mathcal{S}_2} \leq i_{\mathcal{S}'}$. However, since $\mathcal{S}'' \subset \mathcal{S}_2$, $i_{\mathcal{S}_2} \geq i_{\mathcal{S}''} = i_{\mathcal{S}'}$. So $i_{\mathcal{S}_2} = i_{\mathcal{S}'}$. Moreover, we already know that $(\mathcal{S}, \mathcal{S}_2)$ verify (1-3). That means we can create $\mathcal{S}_2 \supsetneq \mathcal{S}''$ which contradicts our assumption that $\mathcal{S}''$ is the largest closed subgraph strictly included in $\mathcal{S}'$ that could be generated.

Case 2: $\mathcal{I}(cl(\mathcal{S}'' \bigcup \{v\}))$ includes one of the pattern of `forbidden` $(\mathcal{S}'')$: We note $v_1, \ldots, v_l$ the sequence that created $\mathcal{S}''$ from $\mathcal{S}_d$. At one point in that process, we added a pattern to `forbidden`$(\mathcal{S}'')$ that is now contained in $\mathcal{I}(cl(\mathcal{S}'' \bigcup \{v\}))$, let's say when adding v_k. This pattern was linked to another equivalence group. If we consider v' a node from that group, we will then construct a subgraph using the sequence $v_1, \ldots, v_{k-1}, v', v_k, \ldots, ...v_l$. Note that since at each new addition, the constructed graph is included in $\mathcal{S}_2$, it's also included in $\mathcal{S}'$. Moreover, each one contains $\mathcal{S}$ and a node from $\mathbb{V}_{i_{\mathcal{S}'}}$ by construction (since it contains $\mathcal{S}_d$). So, using Lemma 2, we know that those additions all respect (1-3), i.e they are valid additions according to our algorithm. This way, we can create a subgraph that contains $\mathcal{S}''$ and v' with our procedure. This contradicts our assumption that $\mathcal{S}''$ is the largest closed subgraph strictly included in $\mathcal{S}'$ that could be generated.

This proves that all $\mathcal{S}'$ will be generated from $\mathcal{S}$ and therefore that we have properly inverted the reduction.

S-2 Efficient implementation of `CALDERA`

S-2.1 Full algorithm

Algorithm S-1 explores $\mathcal{C}$ through a BFS traversal of the tree defined by the reduction $\mathcal{P}$, exploiting Algorithm 1 (L.15) to invert the reduction and using this exploration to apply the Tarone testing procedure described in Section 2.1 (L7-12, 14), before finally testing the testable CCS>(L21-25).

S-2.2 Memory footpring

We consider a subgraph $\mathcal{S}'$ created following Algorithm 1, in the second case. We therefore have $\mathcal{S}, \mathcal{S}_p$ such that: $\mathcal{P}(\mathcal{S}') = \mathcal{P}(\mathcal{S}) = \mathcal{S}_p$ and $i_{\mathcal{S}'} = i_{\mathcal{S}}$. After creating $\mathcal{S}'$, we explore its children, with an itemtable $\mathcal{T}$. All elements of `Children`$(\mathcal{S}', \mathcal{S}_p, i_{\mathcal{S}'}, \mathcal{T})$ will have a pattern which includes $\mathcal{I}(\mathcal{S}')$. Moreover, by definition of the equivalence groups, we already know that $\{\mathcal{I} \in \mathcal{T} : \mathcal{I} \subset \mathcal{I}(\mathcal{S}')\} = \emptyset$. Therefore, when constructing $\mathcal{S}'' \in$ `Children`$(\mathcal{S}', \mathcal{S}_p, i_{\mathcal{S}'}, \mathcal{T})$, only the elements in $\mathcal{I}(\mathcal{S}'') \setminus \mathcal{I}(\mathcal{S}')$ need to be considered.

Algorithm S-1 List significant closed connected subgraphs

```
1: procedure LIST_SIG_CLOSED_SUBGRAPHS($\mathcal{G}, \alpha$)
2:     $Q \leftarrow Children(\emptyset, \emptyset, \text{NULL}, \emptyset)$
3:     $\mathcal{R} \leftarrow \emptyset$
4:     $k \leftarrow 1$
5:     while $Q \neq \emptyset$ do
6:         $\mathcal{S} \leftarrow \texttt{Dequeue}(Q)$
7:         if $p^\star(\mathcal{S}) \leq \alpha/k$ then
8:             $\mathcal{R} \leftarrow \mathcal{R} \cup \{\mathcal{S}\}$
9:         end if
10:        if $|\mathcal{R}| > k$ then
11:            $k \leftarrow k + 1$
12:            $\mathcal{R} \leftarrow \{\mathcal{S} \in \mathcal{R} : p^\star(\mathcal{S}) \leq \alpha/k\}$
13:        end if
14:        if $\tilde{p}^\star(\mathcal{S}) \leq \alpha/k$ then
15:            for $\mathcal{S}' \in \texttt{Children}(\mathcal{S}, \mathcal{S}, \text{NULL}, \emptyset)$ do
16:                $\texttt{Enqueue}(\mathcal{S}', Q)$
17:            end for
18:        end if
19:    end while
20:    Solutions$\leftarrow \emptyset$
21:    for $\mathcal{S} \in \mathcal{R}$ do
22:        if $p(\mathcal{S}) \leq \alpha/k$ then
23:            Add $\mathcal{S}$ to Solutions
24:        end if
25:    end for
26:    return Solutions
27: end procedure
```

Following Uno et al. [2004], we store $\mathcal{T}$ as a matrix of binary patterns. Therefore, some columns can be deleted without loss of information: in Line 13 of algorithm 1, we only keep the columns that are not in $\mathcal{I}(\mathcal{S})$. As the `Children` function is called recursively, the itemtable $\mathcal{T}$ will grow in the number of patterns saved (*i.e* number of rows) but the memory footprint of each pattern will be smaller (*i.e* fewer columns).

S-3 Benefit of breadth-first search

S-3.1 General intuition

In the DFS search, at any level, even if the CCSs visited along a branch do increase k and therefore lower the testability threshold, all the other CCSs of the level will need to be visited regardless of their testability.

By contrast, in BFS, the increase of k gained by visiting all CCSs of the same level in the tree will lower the threshold α/k for all CCS at the next level, making more branches prunable.

Therefore, the breath-first search does prune the subgraph much more efficiently.

S-3.2 A simplified scenario

We consider a very simple graph with $p = 3$ nodes, $J = 1$ population and $n = 12$ samples. The graph is displayed in Fig S1a. Using the reduction from `CALDERA`, we generate a tree structure on $\mathcal{C}$, displayed in Fig S1b.

Then we can explore this structure in depth-first or breadth-first, while pruning using $\alpha = 1$. The order resulting from an exploration in depth-first can be found in Table S1 and the order fom the exploration in breadth-first can be found in Table S2. In this simple setting, exploring in breadth only visits 4 subgraphs while exploring in depth visits 7. This is because the BFS enumerates testable subgraphs more quickly, thereby increasing k and lowering the threshold, which means that the branch starting at $\{v_1\}$ is pruned earlier in the exploration.

Subgraph explored	Number of subgraphs explored	Value of the threshold	Testable subgraphs
$\{v_1\}$	1	.15	$\{\{v_1\}\}$
$\{v_1, v_2\}$	2	.15/2	$\{\{v_1\}, \{v_1, v_2\}\}$
$\{v_1, v_2, v_3\}$	3	.15/2	$\{\{v_1\}, \{v_1, v_2\}\}$
$\{v_1, v_3\}$	4	.15/3	$\emptyset$
$\{v_2\}$	5	.15/3	$\{\{v_2\}\}$
$\{v_2, v_3\}$	6	.15/3	$\{\{v_2\}, \{v_2, v_3\}\}$
$\{v_3\}$	7	.15/3	$\{\{v_2\}, \{v_2, v_3\}, \{v_3\}\}$

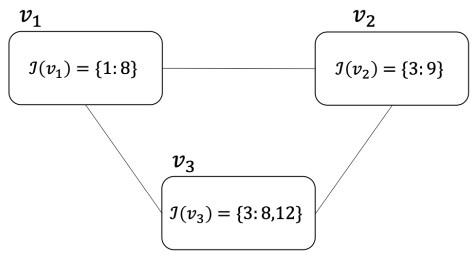
Table S1: Order of exploration of the elements of $\mathcal{C}$ while exploring depth-first

S-3.3 A more general setting to understand why BFS is more efficient than DFS

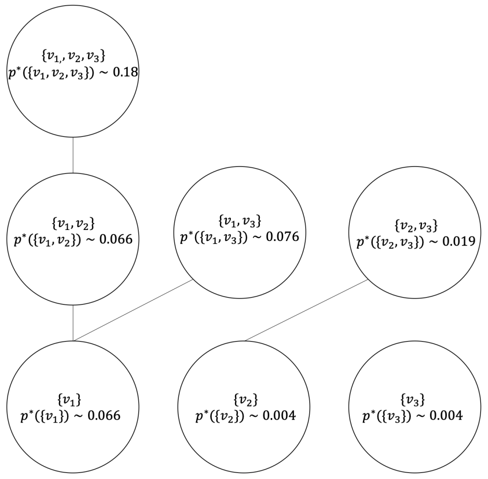
We consider a very simple graph model where, for $v \in \mathcal{V}$ and $i \in \{1, \ldots, n\}$, $i \in \mathcal{I}(v) \sim$ Binom(prop) and the patterns are independent across nodes . We have no population structure, which means that we consider Fisher's exact test. For a given level α, we want to compute $f(\alpha, \text{prop}) = \mathbb{P}(p^\star(\{v\}) > \alpha, \forall v \in \mathcal{V})$, that is the probability that no subgraph is testable at the first stage of our tree on $\mathcal{C}$.

Subgraph explored	Number of subgraphs explored	Value of the threshold	Testable subgraphs
$\{v_1\}$	1	.15	$\{\{v_1\}\}$
$\{v_2\}$	2	.15/2	$\{\{v_1\},\{v_2\}\}$
$\{v_3\}$	3	.15/3	$\{\{v_2\},\{v_3\}\}$
$\{v_2,v_3\}$	4	.15/3	$\{\{v_2\},\{v_2,v_3\},\{v_3\}\}$

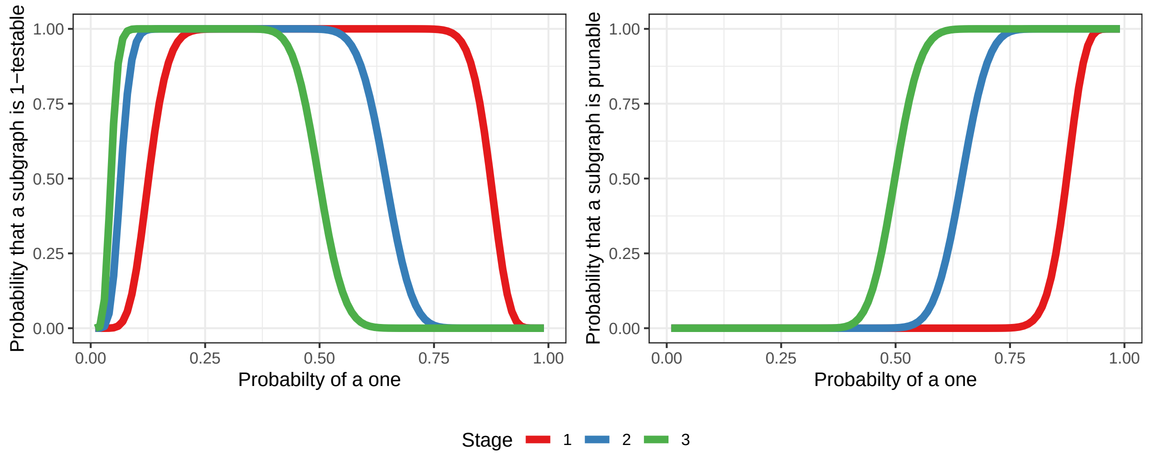
Table S2: Order of exploration of the elements of $\mathcal{C}$ while exploring breadth-first


v_1
$\mathcal{I}(v_1) = \{1:8\}$
v_2
$\mathcal{I}(v_2) = \{3:9\}$
v_3
$\mathcal{I}(v_3) = \{3:8,12\}$

(a) A simple graph example with $p = 3$, $J = 1$ and $N = 12$ which we explore using `CALDERA` with pruning, and $\alpha = .15$


$\{v_1,v_2,v_3\}$
$p^*(\{v_1,v_2,v_3\}) \sim 0.18$
$\{v_1,v_2\}$
$p^*(\{v_1,v_2\}) \sim 0.066$
$\{v_1,v_3\}$
$p^*(\{v_1,v_3\}) \sim 0.076$
$\{v_2,v_3\}$
$p^*(\{v_2,v_3\}) \sim 0.019$
$\{v_1\}$
$p^*(\{v_1\}) \sim 0.066$
$\{v_2\}$
$p^*(\{v_2\}) \sim 0.004$
$\{v_3\}$
$p^*(\{v_3\}) \sim 0.004$

(b) Order on elements of $\mathcal{C}$ from the graph in a), according to the reduction of definition 2



(c) $n = 100, p = 100, \alpha = 10^{-4}$. For a simple model described in S-3.3 where $i \in \mathcal{I}(v) \sim$ Binom(prop), we plot the probability that any subgraph is 1-testable or prunable as a function of prop

Figure S1: Simple examples where the search in breadth-first is much more efficient that depth-first

Since we consider Fisher's exact test, there is a bijection between $p^\star(\{v\})$ and $x_{\{v\}}$ so $p^\star(\{v\}) > \alpha \implies x_{\{v\}} \geq \sigma(\alpha)$. Moreover, $x_{\{v\}} \sim \mathcal{B}(\text{prop}, n)$, so $f(\alpha, \text{prop}) = 1 - \left(\mathbf{F}_{\mathcal{B}\rangle\backslash\wr\Updownarrow(\text{prop},n)}(\sigma_\alpha)\right)^p$ with $\mathbf{F}_{\mathcal{B}\rangle\backslash\wr\Updownarrow(\text{prop},n)}$ the cumulative distribution function of the binomial (prop, n). Since the nodes are independent, the distribution of $x_{\mathcal{S}}$ at any stage of the tree can be computed by recursion. We furthermore assume that the graph structure is such that the number of closed subgraphs is $s \times p$ at stage s.

In Fig S1c, we display the probability that any subgraph is 1-testable or prunable at stage s, for $s \in \{1, 2, 3\}$, $p = 100$ and $\alpha = 10^{-4}$.

For most of the range of values, there is at least one testable subgraph in the first stage. So, by exploring in a BFS manner, we start the second stage with a much lower threshold (i.e., a much higher value of k) which leads to more pruning. For very low values of `prob`, there might be no testable subgraphs at the first stage but there will be at the second stage, which still justifies an exploration in depth. Note that for large p, we can see that there is no testable subgraph at the stages 2 and 3. That is because all such subgraphs have a pattern that is too large. While there may be not testable subgraphs, there are many prunable ones. In that case, an exploration in breadth-first or depth-first would be identical.

This example simplifies two aspects which have opposite effects. The first is that, in practice, the probability of $i \in \mathcal{I}(v)$ is of course not uniform across the graph. It is a distribution with much heavier tails which means that, even if the average number of 1 might be small, it is still quite likely that at least one subgraph is testable. The second is that the patterns of neighbouring nodes are correlated. As such, the patterns cannot increase by as much between stages, which limits both the increase in testable pattern discovery, and the pruning.

S-4 Simulations

S-4.1 General simulation settings

For given values of n (number of samples) and p (number of nodes), we first generate n samples with phenotype $y_i \in \{0,1\}$ such that $\mathbf{P}(y_i = 0) = prop$ (user defined parameter). Then, we generate p nodes. 10% of the nodes will be associated with the phenotype. For each node in the remaining 90%, we randomly generate 3 edges between this node and another in the 90%. The average degree is therefore 6. For those nodes v_j, the associated pattern $\mathcal{I}(v_j)$ is a random vector such that $\mathbf{P}(i \in \mathcal{I}(v_j)) = 0.5$.

Then, we generate the remaining 10% of the nodes associated with the phenotype. We first generate associated patterns $\mathcal{I}_{sig}$ such that $\mathbf{P}(i \in \mathcal{I}_{sig}|y_i = 1) = 0.95$ and $\mathbf{P}(i \in \mathcal{I}_{sig}|y_i = 0) = 0.05$. Then, those patterns are split into 10 significant nodes sig_j such that $\mathbf{P}(i \in \mathcal{I}(\text{sig}_j)|i \in \mathcal{I}_{sig}) = 0.9$ and $\mathcal{I}(\bigcup_{j\in[1\ldots10]}) = \mathcal{I}_{sig}$.

S-4.2 Parameters for various scenarios

We increase p until COIN+LAMP2 times out (sometimes we went a little further to continue investigation the behaviors).

Scenario	1	2	3	4
n	100	50	100	100
$prop$	.5	.5	.2	.2
α	.05	.05	.05	10^{-4}
timeout (days)	2	1	1	1
max value of p	2×10^4	2×10^4	2×10^3	5×10^4

Table S3: Parameter values for the simulations

S-4.3 Results on all scenarios

All computations were run on a r3.4xlarge AWS machine with 16 vCPUs (8 physical ones) and 122GiB[AWS, 2020]. We stop every method once it runs for more than timeout. We also stopped running the entire scenario once we have reached timeout for `COIN+LAMP2` (expect for scenario 3 where we continued further to study the behavior of the various modes of `CALDERA`.

S-4.4 Memory requirements

We also launched scenario 2 while monitoring memory usage for `COIN+LAMP2`, `CALDERA BFS` and `CALDERA DFS`. `CALDERA DFS` uses $1/3$ of the peak memory of `CALDERA BFS`. This is expected since the tree structure that is explored scales in p in breadth but in $n << p$ in depth. Memory-wise, `CALDERA BFS` is on par with `COIN+LAMP2`, which relies on a DFS search. This shows that the use of local itemset tables offers memory gains that are enough to offset the exploration in breath, while providing large speed gains. This also suggests that hybrid explorations might be even better at navigating the memory-speed trade-off.

S-5 Background on the minimal p-value

S-5.1 Minimal p-value

Variable	$i \in \mathcal{I}(\mathcal{S})$	$i \notin \mathcal{I}(\mathcal{S})$	**Rows totals**
$\mathbf{y_i = 1}$	$a_{\mathcal{S},j}$	$n_{1,j} - a_{\mathcal{S},j}$	$n_{1,j}$
$\mathbf{y_i = 0}$	$x_{\mathcal{S},j} - a_{\mathcal{S},j}$	$n_{2,j} - x_{\mathcal{S},j} + a_{\mathcal{S},j}$	$n_{2,j}$
Cols Totals	$x_{\mathcal{S},j}$	$n_j - x_{\mathcal{S},j}$	n_j

Table S4: Association table in community j for subgraph $\mathcal{S}$, used for the CMH test.

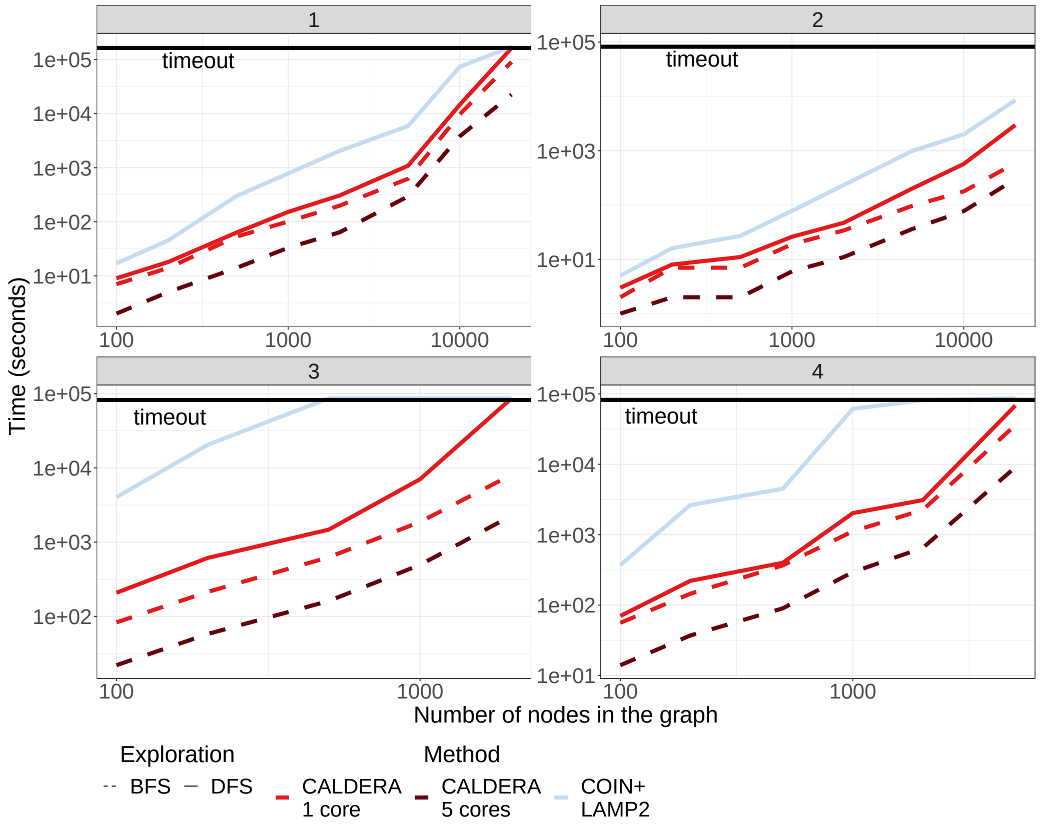


Figure S2: Runtimes for `CALDERA` and `COIN+LAMP` on graphs with various values of covariates p and various values of the simulation parameters.

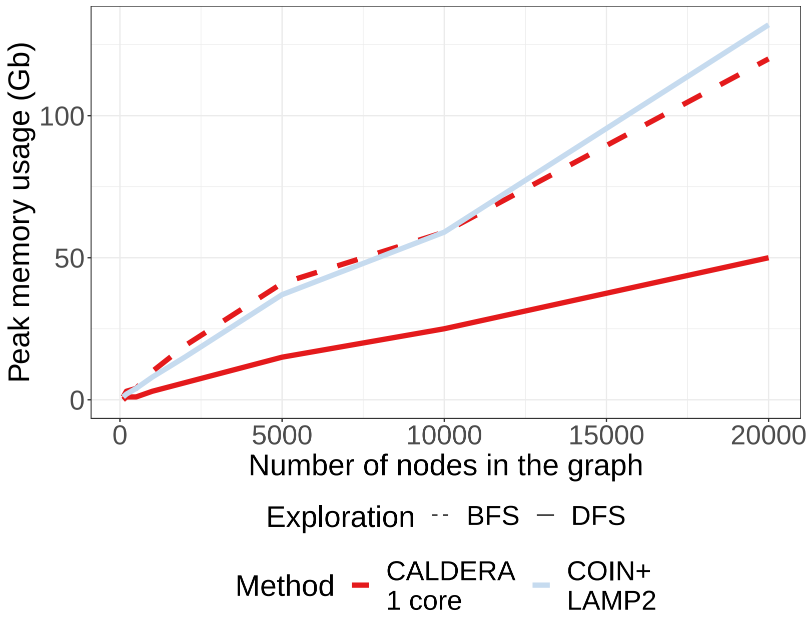


Figure S3: Peak memory usage for `CALDERA` and `COIN+LAMP` on graphs with various values of covariates p.

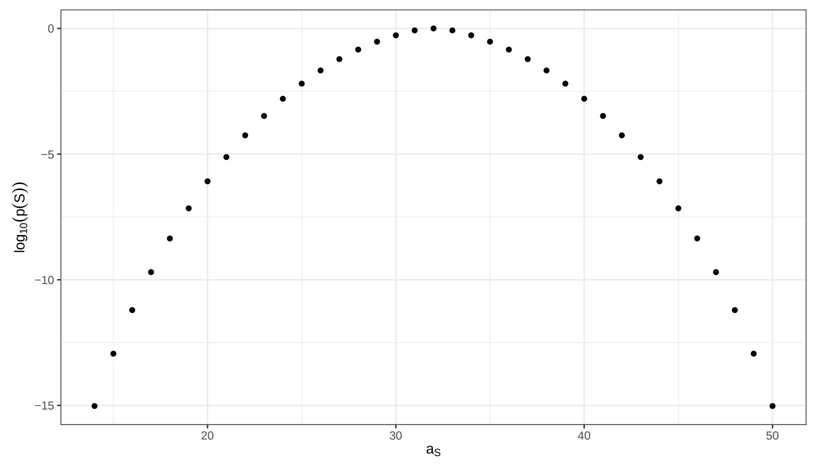


Figure S4: *Finite numbers of possible p-values (log scale) for a fixed value of $n1 = 50$ and $x_{\mathcal{S}} = 64$. Using the notation from table S4, with $J = 1$, $n_1 = 50$, $n = 100$ and $x_{\mathcal{S}} = 64$, the p-value of the χ^2 test is computed for all possible values of $a_{\mathcal{S}}$. Since there are only a finite number of possible $a_{\mathcal{S}}$ values, there are a finite number of possible p-values, and therefore a smallest one. This minimal p-value can be computed from $x_{\mathcal{S}}$, $n1$ and n alone and is $\sim 10^{-15}$*

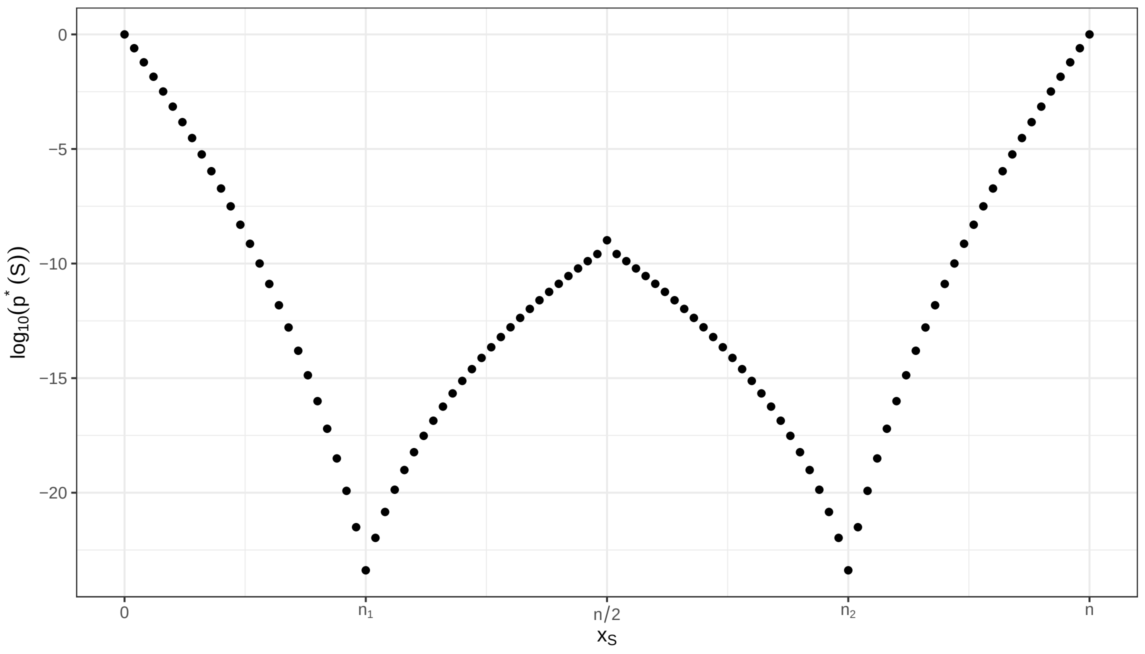


Figure S5: *Minimum p-value as a function of $x_{\mathcal{S}}$ for fixed values of $n_1 = 25$ and $n = 100$. Using the notation from table S4, with $J = 1$, $n_1 = 25$, $n = 100$, the minimal p-value $p^*(\mathcal{S})$ of the χ^2 test is computed for all possible values of $x_{\mathcal{S}}$. For $x_{\mathcal{S}} \geq \max(n_1, n_2)$, the minimal p-value is strictly increasing. If we reach that stage, we can prune the graph and stop the exploration in that direction. Indeed, if $\mathcal{S}' \supseteq \mathcal{S}$ then $x_{\mathcal{S}'} \geq x_{\mathcal{S}}$. So if $p^*(\mathcal{S}) > \frac{\alpha}{k}$, we know that $p^*(\mathcal{S}') > \frac{\alpha}{k}$ without computing it.*