Invitation to Algorithmic Uses of Inclusion–Exclusion

Thore Husfeldt

IT University of Copenhagen, Denmark
Lund University, Sweden

Abstract. I give an introduction to algorithmic uses of the principle of inclusion–exclusion. The presentation is intended to be be concrete and accessible, at the expense of generality and comprehensiveness.

1 The principle of inclusion–exclusion. There are as many odd-sized as even-sized subsets sandwiched between two different sets: For $R \subseteq T$,

$$\sum_{R \subseteq S \subseteq T} (-1)^{|T \setminus S|} = [R = T]. \qquad (1)$$
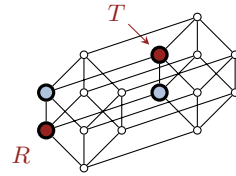
We use Iverson notation $[P]$ for proposition P, meaning $[P] = 1$ if P and $[P] = 0$ otherwise.

Proof of (1). If $R = T$ then there is exactly one sandwiched set, namely $S = T$. Otherwise we set up a bijection between the odd- and even-sized subsets as follows. Fix $t \in T \setminus R$. For every odd-sized subset S_1 with $R \subseteq S_1 \subseteq T$ let $S_0 = S_1 \oplus \{t\}$ denote the symmetric difference of S_1 with $\{t\}$. Note that the size of S_0 is even and that S_0 contains R. Furthermore, S_1 can be recovered from S_0 as $S_1 = S_0 \oplus \{t\}$. $\qquad \square$

Perspective. We will see the (perhaps more familiar) formulation of the principle of inclusion–exclusion in terms of intersecting sets in §6, and another equivalent formulation in §11.

2 Graph colouring. A *k-colouring* of a graph $G = (N, E)$ on $n = |N|$ nodes assigns one of k colours to every node such that neighbouring nodes have different colours. In any such colouring, the nodes of the same colour form a nonempty *independent set*, a set of nodes none of which are neighbours.
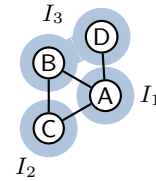
Let $g(S)$ denote the number of nonempty independent subsets in $S \subseteq N$. Then G can be k-coloured if and only if

$$\sum_{S \subseteq N} (-1)^{n - |S|} \big(g(S)\big)^k > 0. \qquad (2)$$

Proof. For every $S \subseteq N$, the term $g(S)^k$ counts the number of ways to pick k nonempty independent sets $I_1, \ldots, I_k$ in S. Thus, we can express the left hand side of (2) as

$$\sum_S \sum_{I_1} \cdots \sum_{I_k} [\forall i \colon I_i \subseteq S](-1)^{|N \setminus S|} = \sum_{I_1} \cdots \sum_{I_k} \sum_S [\forall i \colon I_i \subseteq S](-1)^{|N \setminus S|}.$$

The innermost sum has the form

$$\sum_{I_1 \cup \cdots \cup I_k \subseteq S \subseteq N} (-1)^{|N \setminus S|}.$$

By (1), the only contributions come from $I_1 \cup \cdots \cup I_k = N$. Every such choice indeed corresponds to a valid colouring: For $i = 1, \ldots, k$, let the nodes in I_i have colour i. (This may re-colour some nodes.) Conversely, every valid k-colouring corresponds to such a choice. (In fact, the colourings are the disjoint partitions). □
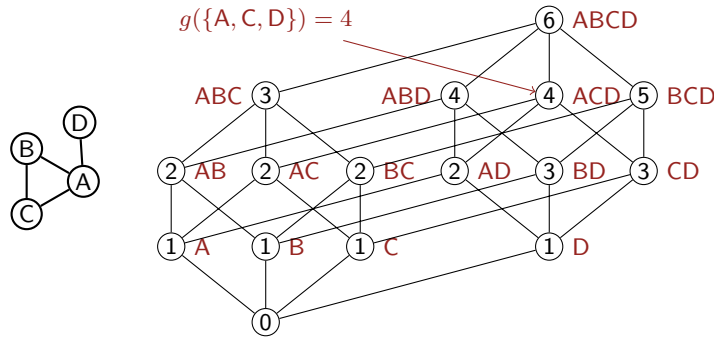

Fig. 1. The values of $g(S)$ for all S for the example graph to the left. Expression (2) evaluates an alternating sum of the cubes of these values, in this case $6^3 - (3^3 + 4^3 + 4^3 + 5^3) + (2^3 + 2^3 + 2^3 + 2^3 + 3^3 + 3^3) - (1^3 + 1^3 + 1^3 + 1^3) + 0 = 18$.

3 Counting the number of independent sets. Expression (2) can be evaluated in two ways:

For each $S \subseteq N$, the value $g(S)$ can be computed in time $O(2^{|S|}|E|)$ by constructing every nonempty subset of S and testing it for independence. Thus, the total running time for evaluating (2) is within a polynomial factor of

$$\sum_{S \subseteq N} 2^{|S|} = \sum_{i=1}^n \binom{n}{i} 2^i = 3^n.$$

The space requirement is polynomial.

Alternatively, we first build a table with 2^n entries containing $g(S)$ for all $S \subseteq N$, after which we can evaluate (2) in time and space $2^n n^{O(1)}$.

Such a table is easy to build given a recurrence for $g(S)$. We have $g(\varnothing) = 0$, and

$$g(S) = g(S \setminus \{v\}) + g(S \setminus N[v]) + 1 \qquad (v \in S), \qquad (3)$$

where $N[v] = \{v\} \cup \{u \in N : uv \in E\}$ denotes the closed neighbourhood of v.

Proof of (3). Fix $v \in S$ and consider the nonempty independent sets $I \subseteq S$. They can be partitioned into two classes: either $v \in I$ or $v \notin I$. The latter sets are counted in $g(S \setminus \{v\})$. It remains to argue that the sets $I \ni v$ are counted in $g(S \setminus N[v]) + 1$. We will do this by counting the equipotent family of sets $I \setminus \{v\}$ instead. Since I contains v and is independent, it cannot contain other nodes in $N[v]$. Thus $I \setminus \{v\}$ is disjoint from $N[v]$ and contained in S. Now, either I is the singleton $\{v\}$ itself, accounted for by the '+1' term, or $I \setminus \{v\}$ is a nonempty independent set and therefore counted in $g(S \setminus N[v])$. $\qquad\square$
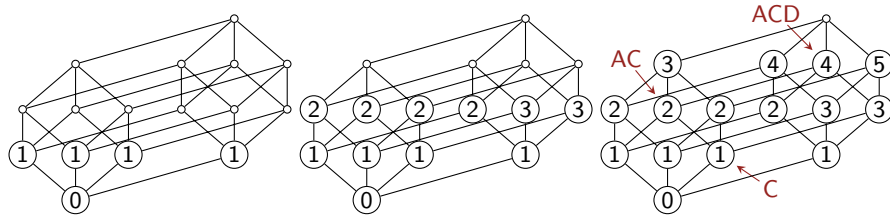

Fig. 2. Three stages in the tabulation of $g(S)$ for all $S \subseteq N$ bottom-up. For example, the value of $g(\{A, C, D\})$ is given by (3) with $v = D$ as $g(\{A, C\}) + g(\{C\}) + 1 = 4$.

Perspective. The *brute force* solution for graph colouring tries all k^n assignments of colours to the nodes, which is slower for $k \geq 4$. Another approach is *dynamic programming over the subsets* [15], based on the idea that G can be k-coloured if and only if $G[N \setminus S]$ can be $(k-1)$-coloured for some nonempty independent set S. That algorithm also runs within a polynomial factor of 3^n, but uses exponential space. In summary, the inclusion–exclusion approach is faster than brute force, and uses less space than dynamic programming over the subsets. The insight that this idea applies to a wide range of sequencing and packing problems goes back to Karp [12], the application to graph colouring is from [2].

We use a space–time trade-off to reducing the exponential running time factor from 3^n to 2^n, applying dynamic programming to tabulate the decrease-and-conquer recurrence (3), based on [8]. Recurrence (3) depends heavily on the structure of independent sets; a more general approach is shown in §10.

The two strategies for computing $g(S)$ represent extreme cases of a space–time tradeoff that can be balanced [4].

4 Perfect matchings in bipartite graphs. Consider a bipartite graph with bipartition (N, N), where $N = \{1, \ldots, n\}$, and edge set $E \subseteq N \times N$. A *perfect matching* is an edge subset $M \subseteq E$ that includes every node as an endpoint exactly once. See Fig. 3 for some interpretations.

Fig. 3. Row 1: A bipartite graph and its three perfect matchings. Row 2: In the graph's adjacency matrix A, every perfect matching corresponds to a permutation π for which $A_{i,\pi(i)} = 1$ for all $i \in [n]$. Row 3: In the directed n-node graph defined by A, every perfect matching corresponds to a directed cycle partition. Bottom row: an equivalent formulation in terms of non-attacking rooks on a chess board with forbidden positions.
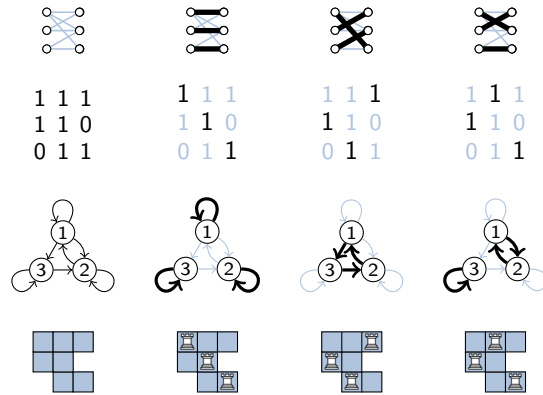
The *Ryser formula* for counting the perfect matchings in such a graph can be given as

$$\sum_{\pi \in S_n} \prod_{i=1}^{n} [i\pi(i) \in E] = \sum_{S \subseteq N} (-1)^{|N \setminus S|} \prod_{i=1}^{n} \sum_{j \in S} [ij \in E], \qquad (4)$$

where S_n denotes the set of permutations from N to N. The left hand side succinctly describes the problem as iterating over all permutations and checking if the corresponding edges (namely, $1\pi(1)$, $2\pi(2)$, …, $n\pi(n)$) are all in E. Direct evaluation would require $n!$ iterations. The right hand side provides an equivalent expression that can be evaluated in time $O(2^n n^2)$, see Fig. 4.

Proof of (4). For fixed $i \in N$, the value $\sum_{j \in S}[ij \in E]$ counts the number of i's neighbours in $S \subseteq N$. Thus the expression

$$\prod_{i=1}^{n} \sum_{j \in S} [ij \in E] \qquad (5)$$

is the number of ways every node $i \in N$ can choose a neighbour from S. (This allows some nodes to select the same neighbour.) Consider such a choice as a mapping $g \colon N \to N$, not necessarily onto, with image $R = g(N)$. The contribution of g to (5) is 1 for every $S \supseteq R$, and its total contribution to the right hand
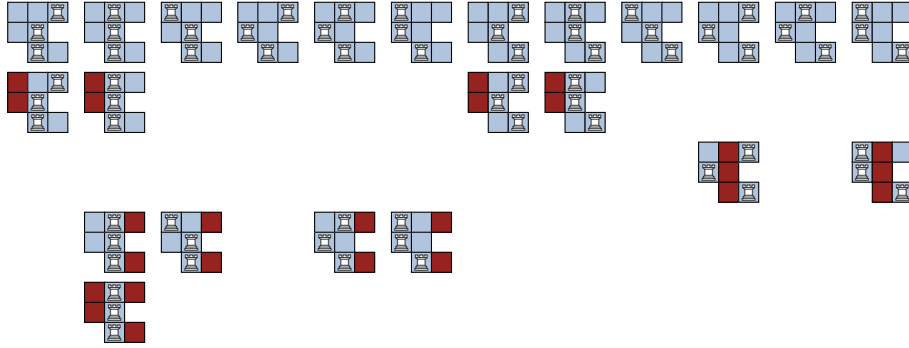
Fig. 4. Inclusion–exclusion for non-attacking rooks. The top row shows all $12 = 3 \cdot 2 \cdot 2$ ways to place exactly one rook in every board line. Every row shows the possible placements in the vertical lines given by $S \subseteq \{1, 2, 3\}$. We omit the rows whose contribution vanishes, namely $S = \{1\}$, $S = \{3\}$ and $S = \varnothing$. Of particular interest is the second column, which is subtracted twice and later added again. The entire calculation is $12 - 4 - 2 - 4 + 1 + 0 + 0 - 0 = 3$.

side of (4) is, using (1),

$$\sum_{R \subseteq S \subseteq N} (-1)^{|N \setminus S|} \cdot 1 = [g(N) = N].$$

Thus g contributes if and only if it is a permutation. □

Perspective. Bipartite matching is an example of a sequencing problem, where inclusion–exclusion replaces an enumeration over permutations, $\sum_{\pi \in S_n}$ by an alternating enumeration over subsets $\sum_{S \subseteq N} (-1)^{|N \setminus S|}$ of functions with restricted range. Typically, this reduces a factor $n!$ in the running time to 2^n. One can express the idea algebraically like this:

$$\begin{aligned}
\sum_{\substack{f: N \to N \\ f(N) = N}} [\cdots] &= \sum_R [R = N] \sum_{\substack{f: N \to N \\ f(N) = R}} [\cdots] \\
&= \sum_R \sum_S [R \subseteq S] (-1)^{|N \setminus S|} \sum_{\substack{f: N \to N \\ f(N) = R}} [\cdots] \\
&= \sum_S (-1)^{|N \setminus S|} \sum_R [R \subseteq S] \sum_{\substack{f: N \to N \\ f(N) = R}} [\cdots] \\
&= \sum_S (-1)^{|N \setminus S|} \sum_{f: N \to S} [\cdots].
\end{aligned} \qquad (6)$$

Ryser's formula is normally given in a more general form, for the *permanent* $\sum_\pi \prod_i A_{i\pi(i)}$ of a matrix, where the entries can be other than just 0 and 1. The

running time can be improved to $O(2^n n)$ arithmetic operations by iterating over N in Gray code order.

Ryser's formula [17] is a very well-known result in combinatorics and appears in many textbooks. However, it is easy to achieve running time $O(2^n n)$ using dynamic programming over the subsets, at the expense of space $O(2^n)$. This is the standard approach to sequencing problems [1,11], and appears as an exercise in Knuth [13, pp. 515-516], but usually not in the combinatorics literature. We will witness the opposite methodological preferences in §7. Inclusion–exclusion-based algorithms for the permanent of non-square matrices in semirings are described in [5].

5 Perfect matchings in general graphs. We turn to graphs that are not necessarily bipartite. In general, the number of perfect matchings in a graph with an even number n of nodes N is

$$\sum_{S \subseteq N} (-1)^{|N \setminus S|} \binom{e[S]}{n/2},\tag{7}$$

where $e[S]$ denotes the number of edges between nodes in $S \subseteq N$.

Proof. The term $\binom{e[S]}{n/2}$ counts the number of ways to select $n/2$ distinct edges with endpoints in S. (The edges are distinct, but may share nodes.) Consider such a selection $F \subseteq E$ and let $R = \bigcup_{uv \in F} \{u, v\}$ denote the nodes covered by the selected edges. The total contribution of F to the right hand side of (7) is

$$\sum_{R \subseteq S \subseteq N} (-1)^{|N \setminus S|} \binom{e[S]}{n/2} = [R = N],$$

using (1). Thus, F contributes 1 if and only if it covers all nodes. Since F contains $n/2$ edges, F it must be a perfect matching. □

The running time is within a polynomial factor of 2^n, and the space is polynomial; see Fig. 5.

Perspective. Perfect matchings in general graphs is a packing or partitioning problem, while the bipartite case was a a sequencing problem and the graph colouring example in §2 was a covering problem. (Admittedly, the distinction between these things is not very clear.) The application is form [2], which also contains another space–time trade-off based on matrix multiplication.

The point of the large in example in Fig. 5 is to illustrate the intuition that inclusion–exclusion is a *sieve*. We start with a large collection of easy-to-compute objects (the top row in Fig. 5), and let the alternating sum perform a cancellation that sifts through the objects and keeps only the interesting ones in the sieve.
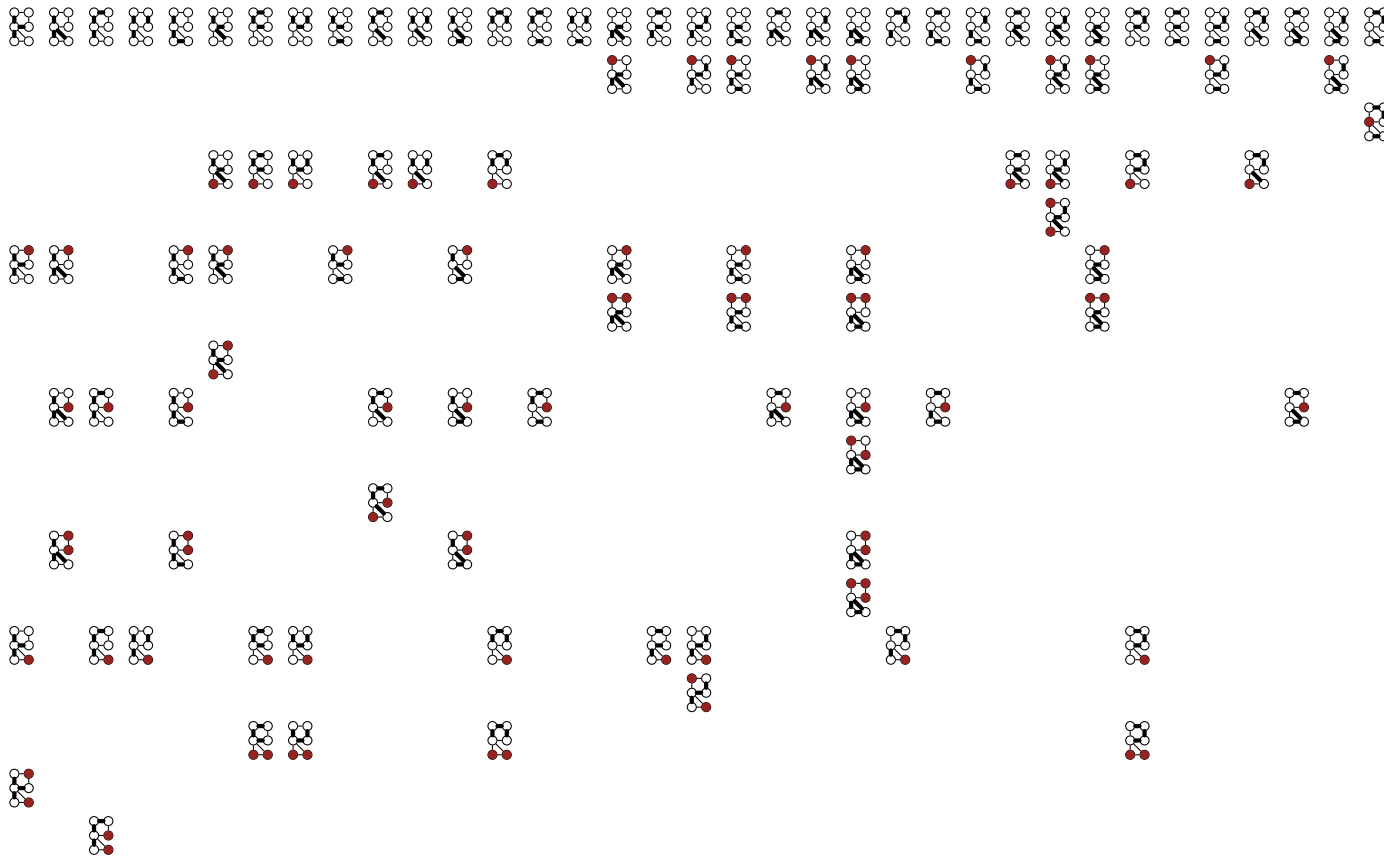
Fig. 5. The perfect matching algorithm for a graph with $n = 6$ and $m = 7$. There are $\binom{7}{3} = 35$ ways to pick 3 edges out of 7, shown in the top row. The triangle appears in 7 other terms (4 negative, 3 positive), the two perfect matchings appear only once.

6 Inclusion–exclusion for sets. If two sets A and B have no elements in common, then we have the principle of *addition*: $|A \cup B| = |A| + |B|$. In general, the equality does not hold, and all we have is $|A \cup B| \le |A| + |B|$. Observing that every element of $A \cap B$ is counted exactly twice on the right hand side allows us subtract the error term:
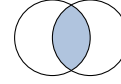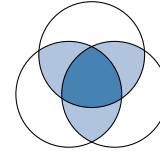
$$|A \cup B| = |A| + |B| - |A \cap B|,$$

often called the *principle of inclusion–exclusion*.

Actually, that's just a special case, the formula is elevated to a principle by generalising to more sets. For three sets, the formula

$$|A \cup B \cup C| = |A| + |B| + |C| - |A \cap B| - |A \cap C| - |B \cap C| + |A \cap B \cap C|$$

can be verified by staring at a Venn diagram. The right-hand side contains all the possible intersections of A, B, and C, with signs depending on how many sets intersect. Generalising this leads us to

$$|A_1 \cup \cdots \cup A_n| = \sum_{\varnothing \ne S \subseteq N} (-1)^{|S|+1} \Big| \bigcap_{i \in S} A_i \Big|, \tag{8}$$

where $N = \{1, \dots, n\}$. Equivalently, the number of elements not in any A_i is

$$\big| \overline{A_1 \cup \cdots \cup A_n} \big| = \sum_{S \subseteq N} (-1)^{|S|} \Big| \bigcap_{i \in S} A_i \Big|, \tag{9}$$

with the usual convention that the 'empty' intersection $\bigcap_{i \in \varnothing} A_i$ equals the universe from which the sets are taken.

Proof of (9). We consider the contribution of every element a.

Let $T = \{\, i \in N \colon a \in A_i \,\}$ denote the index set of the sets containing a. The contribution of a to the left hand side of (9) is $[T = \varnothing]$. To determine its contribution to the right hand side, we observe that a belongs to the intersection $\bigcap_{i \in T} A_i$ and all its sub-intersections, so it contributes 1 to all corresponding terms. More precisely, the total contribution of a is given by

$$\sum_{S \subseteq T} (-1)^{|S|} = (-1)^{|T|} \sum_{S \subseteq T} (-1)^{|T \setminus S|} = (-1)^{|T|} [T = \varnothing] = [T = \varnothing],$$

using (1) with $R = \varnothing$. $\qquad\square$

Perspective. Expressions (8) and (9) are the standard textbook presentation of inclusion–exclusion. We derived them from (1) with $R = \varnothing$. Let us show the opposite derivation to see that the two formulations are equivalent.

Let T be a nonempty, finite set and write $T = \{1, \dots, n\}$. Consider the family of identical subsets $A_i = \{1\}$ for all $i \in T$. Their union and every nonempty intersection is $\{1\}$. Thus, from (8),

$$1 = \sum_{\varnothing \ne S \subseteq T} (-1)^{|S|+1} \cdot 1,$$

which gives (1) for $R = \varnothing$ after subtracting 1 from both sides.

7   Hamiltonian paths.   A *walk* in a graph is a sequence of neighbouring nodes $v_1, \ldots, v_k$. Such a walk is a *path* if every node appears at most once, and a path is *Hamiltonian* if it includes every vertex in G. For ease of notation we also assume that all Hamiltonian paths start in node $v_1 = 1$.

Given a graph G on n nodes N let $a(X)$ denote the number of walks of length n that start in 1 and 'avoid' the nodes in $X \subseteq V$, i.e., walks of the form $1 = v_1, \ldots, v_n$ with $v_i \notin X$ for all $1 \le i \le n$. Then the number of Hamiltonian paths in G is $a(\varnothing)$.

Let A_i denote the walks that avoid $\{i\}$. Then $a(\varnothing) = |\bigcup_{i \in N} A_i|$ and $a(X) = |\bigcap_{i \in X} A_i|$. Thus, from (9), we have

$$a(\varnothing) = \sum_{X \subseteq N} (-1)^{|X|} a(X) \,.$$

For every X, the value $a(X)$ can be computed in polynomial time using dynamic programming (over the lengths and endpoints, not over the subsets). For $t \in V$ and $k = 1, \ldots, n$ let for a moment $a^k(X, t)$ denote the number of walks of the form $1 = v_1, \ldots, v_k = t$ with $v_i \notin X$. Then we can set $a^1(X, v) = [v = 1]$ and

$$a^{k+1}(X, t) = \sum_{v \in V} a^k(X, v)[vt \in E] \,.$$

The total time to compute $a(X) = \sum_{t \in V} a(X, t)$ becomes $O(n^2|E|)$, using polynomial space. It follows that Hamiltonicity in an n-node graph can be decided (in fact, counted) in time $O(2^n n^2 |E|)$ and linear space.

Perspective. Hamiltonicity is one of the earliest explicitly algorithmic applications of inclusion–exclusion. It appears in [12], but implicitly already in [14], where it is described for the traveling salesman problem with bounded integer weights. Both these papers have lived in relative obscurity until recently, for example the TSP result has been both reproved and called 'open' in a number of places.

Hamiltonicity is also the canonical application of another algorithmic technique, *dynamic programming over the subsets* [1,11], which yields an algorithm with similar time bounds but exponential space. Thus, we can observe a curious cultural difference in the default approach to hard sequencing problems: Dynamic programming is the well-known solution to Hamiltonicity, while the inclusion–exclusion formulation is often overlooked. For the permanent (§4), the situation is reversed.

8   Steiner tree.   For a graph $G = (N, E)$ and a subset $\{t_1, \ldots, t_k\} \subseteq N$ of nodes called *terminals*, a *Steiner tree* is a tree in G that contains all terminals. We want to determine the smallest size of such a tree.

We consider a related structure that is to a tree what a walk is to a path. A *willow* W consists of a multiset of nodes $S(W)$ from N and a parent function $p \colon S(W) \to S(W)$, such that repeated applications of p end in a specified *root*

node $r \in S(W)$. The size of W is the number of nodes in $S(W)$, counted with repetition.

Every tree can be turned into a willow, by selecting an arbitrary root and orienting the edges towards it, but the converse is not true. However, a minimum size willow over a set of nodes is a tree: Assume a node appears twice in W. Remove one of its occurrences $u \in S(W)$, not the root node, and change the parent $p(v)$ of all v with $p(v) = u$ to $p(v) = p(u)$. The resulting willow is smaller than W but spans the same nodes. Finally, when all repeated nodes are removed, p defines a tree.

Thus, it suffices to look for a size-l willow W that includes all terminals, for increasing $l = k, \ldots, n$. Set A_i to be the set of willows of size l that avoid t_i. Then, from (9), the number of willows of size l that include all terminals is

$$\sum_{X \subseteq K} (-1)^{|X|} a^l(X) \,,$$

where $a^l(X) = |\bigcap_{i \in X} A_i|$ is the number of willows of size l that avoid the terminals in X.

Again, we can use dynamic programming to compute $a^l(X)$ for given X. For all $X \subseteq V$ and $u \notin X$ let $a^l(X, u)$ denote the number willows of size l that avoid X and whose root is $u \notin X$. Then $a^1(X, u) = 1$ and

$$a^k(X, u) = \sum_{uv \in E} \sum_{i=1}^{k-1} a^i(X, u) a^{k-i}(X, v) \,.$$

Perspective. This application is from [16]. The role of inclusion–exclusion is slightly different from the Hamiltonian path construction in the previous subsection, because we have no control over the size of the objects we are sieving for.

There, we sifted through all walks of length n. What was left in the sieve were the walks of length n that visit all n nodes. Thus, every node appears exactly once, so that sieve contained exactly the desired solutions, i.e., the Hamiltonian paths.

Here, we sift through all willows of size l. What is left in the sieve are the willows that visit all terminals. For given l, these are not necessarily trees. Instead, correctness hinges on the fact that we already sifted through willows of smaller size. To strain the metaphor, we use increasingly fine sieves until we find something.

9 Long paths.
Consider a graph $G = (N, E)$ and integer $k \leq n$. We want to detect if G has a path of length k. Inspired by the Hamiltonicity construction in §7 we look at all walks $(w_1, \ldots, w_k)$ on k nodes. For expository reasons we again stipulate that all walks begin in a fixed node $w_1 = 1$. Write $K = \{1, \ldots, k\}$.

For every edge e pick a random value $r(e)$. For every vertex v and integer $k \in K$ pick a random value $r(v, k)$. With foresight, the values are chosen uniformly
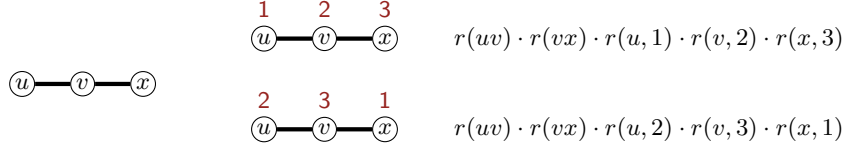
$$r(uv) \cdot r(vx) \cdot r(u,1) \cdot r(v,2) \cdot r(x,3)$$

$$r(uv) \cdot r(vx) \cdot r(u,2) \cdot r(v,3) \cdot r(x,1)$$

Fig. 6. Left: The path $W = (u, v, x)$. Middle: The nodes of W labelled with two permutations. Right: The terms associated with W and the two permutations.

at random form a finite field F of characteristic 2 and size at least $2k(k-1)$. All computation is in this field. For every walk $W = (w_1, \ldots, w_k)$ starting in $w_1 = 1$ and every function $\phi \colon K \to K$ define the term

$$p(W, \phi) = \left(\prod_{i=1}^{k-1} r(w_i w_{i+1}) \right) \left(\prod_{i=1}^{k} r(w_i, \phi(i)) \right), \tag{10}$$

see Fig. 6.

Consider the sum over all walks W in G and all permutations $\pi \in S_k$,

$$p(G) = \sum_{\pi \in S_k} \sum_{W} p(W, \pi). \tag{11}$$

We will show below that

$$\Pr(\, p(G) = 0 \,) \begin{cases} < \frac{1}{2}, & \text{if } G \text{ contains a } k\text{-path}; \\ = 0, & \text{otherwise}. \end{cases} \tag{12}$$

where the probability is taken over the random choices of r.

To compute (11), we first recognise a summation over a permutation and replace it by an alternating sum over functions with restricted range, as in (6):

$$\sum_{\pi \in S_k} \sum_{W} p(W, \pi) = \sum_{S \subseteq K} (-1)^{|K \setminus S|} \sum_{\phi \colon K \to S} \sum_{W} p(W, \phi).$$

For each $S \subseteq K$, the value of the two inner sums can be computed efficiently using dynamic programming; we omit the details. The total running time is within a polynomial (in n) factor of 2^k.

Proof of (12). Consider the contribution of every walk W to (11).

First assume that W is non-simple and let π be a permutation. We will construct anther permutation ρ such that $\pi \neq \rho$ but $p(W, \pi) = -p(W, \rho)$. Thus, summing over all permutations, the contribution of W is even and therefore vanishes in F. To construct ρ, let (i, j) be the first self-intersection on W, i.e., the lexicographically minimal pair with $w_i = w_j$ and $i < j$. Set ρ equal to π except for $\rho(i) = \pi(j)$ and $\rho(j) = \pi(i)$.

Now assume that W is a path. It is useful to view (11) as a polynomial in variables $x(e), x(v, k)$, evaluated at random points $x(e) = r(e)$, $x(v, k) = r(v, k)$. For every permutation π, the monomial

$$\left(\prod_{i=1}^{k-1} x(w_i w_{i+1}) \right) \left(\prod_{i=1}^{k-1} x(w_i, \pi(i)) \right)$$

is unique. To see this, both W and π can be recovered from $p(W, \pi)$ by first reconstructing the nodes $w_1, \ldots, w_k$ in order, starting at $w_1 = 1$ and following the unique incident edge described by the terms $x(e)$, and then reconstructing π from the terms $x(w_i, \pi(i))$. Thus, (11) can be viewed as a nonzero polynomial of degree $k(k-1)$ evaluated at $m + nk$ random points from F. By the DeMillo–Lipton–Schwarz–Zippel lemma [9,18], it evaluates to zero with probability less than $k(k-1)/|F| \leq \frac{1}{2}$. □

Perspective. The construction is implicit in [6] and not optimal. Again, the starting point is the same as for Hamiltonicity in §7: to sieve for paths among walks, whose contribution is computed by dynamic programming.

However, instead of counting the number of walks, we define an associated family of algebraic objects (namely, a multinomial defined by the walk and a permutation) and work with these objects instead. Strictly speaking, we did associate algebraic objects to walks even before, but the object was somewhat innocent: the integer 1.

There are two filtering processes at work: The sifting for paths among walks is performed by the the cancellation of non-simple, permutation-labelled walks in characteristic 2, rather than the by inclusion–exclusion sieve. At the danger of overtaxing the sieving metaphor, the permutation-labelling plays the role of *mercury* in gold mining; inclusion–exclusion ensures that the 'mercury' can be added and removed in time 2^k instead of the straightforward $k!$.

10 Yates's algorithm.   Let $f \colon 2^N \to \{0, 1\}$ be the the the indicator function of the nonempty independent sets in a graph. We will revisit the task of §3, computing

$$g(S) = \sum_{R \subseteq S} f(R), \tag{13}$$

for all $S \subseteq N$.

The computation proceeds in rounds $i = 1, \ldots, n$. Initially, set $g_0(S) = f(S)$ for all $S \subseteq N$. Then we have, for $i = 1, \ldots, n$,

$$g_i(S) = g_{i-1}(S) + [i \in S] \cdot g_{i-1}(S \setminus \{i\}) \qquad (S \subseteq N). \tag{14}$$

Finally, $g(S) = g_n(S)$.

Proof of (14). The intuition is that $g_0(S), \ldots, g_n(S)$ approach $g(S)$ 'coordinate-wise' by fixing fewer and fewer bits of S. To be precise, for $i = 1, \ldots, n$,

$$g_i(S) = \sum_{R \subseteq S} [S \cap \{i+1, \ldots, n\} = R \cap \{i+1, \ldots, n\}] \cdot f(R). \tag{15}$$
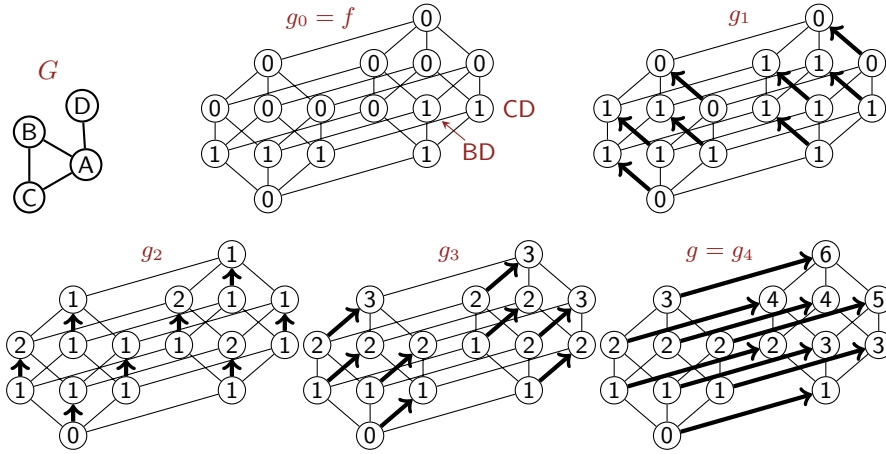
Fig. 7. Yates's algorithm on the indicator function of the nonempty independent sets of the graph G. Arrows indicate how the value of $g_i(S)$ for $i \in S$ is computed by adding $g_{i-1}(S \setminus \{i\})$ to the 'previous' value $g_{i-1}(S)$.

In particular, $g_n(S) = \sum_{R \subseteq S} f(R)$. Correctness of (14) is established by a straightforward but tedious induction argument for (15). The base case $g_0 = f$ is immediate. For the inductive step, adopt the notation $S(i)$ for $S \cap \{i+1, \ldots, n\}$. Then the right hand side of (15) can be written as

$$\sum_{R \subseteq S} [S(i) = R(i)] f(R)$$

$$= \sum_{\substack{R \subseteq S \\ i \in R}} [S(i) = R(i)] f(R) + \sum_{\substack{R \subseteq S \\ i \notin R}} [S(i) = R(i)] f(R).$$

If $i \notin S$ then the first sum vanishes and the second sum simplifies to

$$\sum_{R \subseteq S} [S(i-1) = R(i-1)] f(R) = g_{i-1}(S)$$

by induction. If $i \in S$ then we can rewrite both sums to

$$\sum_{R \subseteq S} [S(i-1) = R(i-1)] f(R) + \sum_{\substack{R \subseteq S \\ i \notin S}} [S(i-1) = R(i-1)] f(R)$$

$$= g_{i-1}(S) + g_{i-1}(S \setminus \{i\})$$

by induction. Finally, by (14) the entire expression equals $g_i(S)$. $\qquad \square$

Perspective. As before, our approach is basically dynamic programming for a decrease-and-conquer recurrence. The time and space requirements are within a linear factor of the ones given in §2. However, the expression (14) is more general and does not depend on the structure of independent sets. It applies to any function $f\colon 2^N \to R$ from subsets to a ring, extending the algorithm to many other covering problems than graph colouring.

Yates's algorithm has much in common with the fast Fourier transform. We can illustrate its operation using a butterfly-like network, see Fig. 8.
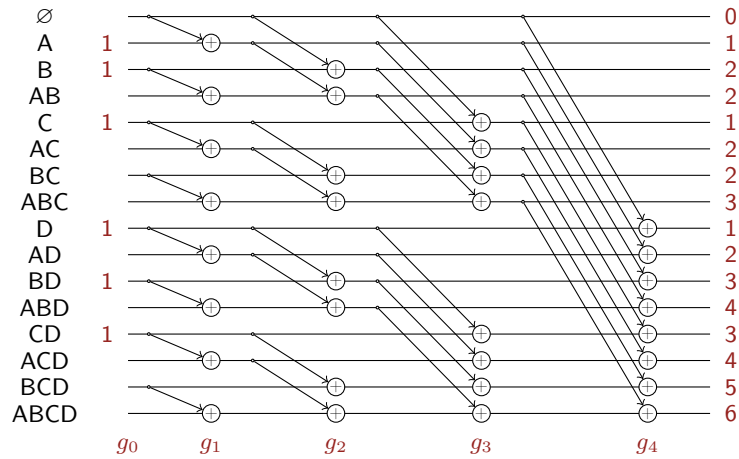

Fig. 8. Yates's algorithm for the zeta transform.

Here we used Yates's algorithm to compute (13), but the method is more general than that. For example, it computes the *Möbius transform*, see (17) below, and many others. A classical treatment of the algorithm appears in [13], recent applications and modifications are in [7] and the forthcoming journal version of [3].

11 Möbius inversion. Let $f\colon 2^N \to \{0,1\}$ be a function of subsets of N to $\{0,1\}$ (indeed, any ring would do). To connect to the graph colouring example from §2, think of f as the indicator function of the nonempty independent sets in a graph. The *zeta transform* of f is the function $(f\zeta)\colon 2^N \to \{0,1\}$ defined point-wise by

$$(f\zeta)(T) = \sum_{S \subseteq T} f(S)\,. \tag{16}$$

The brackets around $(f\zeta)$ are usually omitted. The *Möbius transform* of f is the function $(f\mu)\colon 2^N \to \{0,1\}$ defined point-wise by

$$(f\mu)(T) = \sum_{S \subseteq T} (-1)^{|T \setminus S|} f(S), \tag{17}$$

This allows us to state the principle of inclusion–exclusion in yet another way:

$$f\zeta\mu = f\mu\zeta = f. \tag{18}$$

Proof. We show $f\zeta\mu = f$, the other argument is similar.

$$f\zeta\mu(T) = \sum_{S \subseteq T} (-1)^{|T \setminus S|} \sum_{R \subseteq S} f(R)$$

$$= \sum_{S} \sum_{R} [S \subseteq T][R \subseteq S](-1)^{|T \setminus S|} f(R)$$

$$= \sum_{R} f(R) \sum_{S} [R \subseteq S \subseteq T](-1)^{|T \setminus S|}.$$

By (1), the inner sum equals $[R = T]$, so the expression simplifies to $f(T)$. $\quad\square$
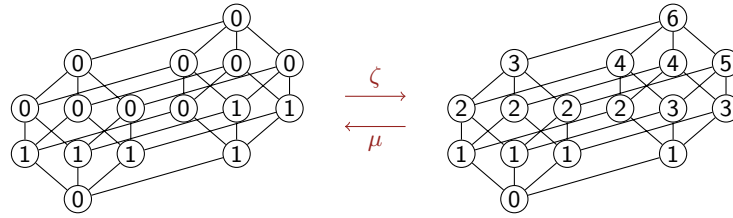

Fig. 9. Möbius inversion.

Perspective. For completeness, let us also derive (1) from (18), to see that the two claims are equivalent. Consider two sets R and T. Define $f(Q) = [Q = R]$. Then, expanding (18),

$$[R = T] = f(T) = \sum_{S \subseteq T} (-1)^{|T \setminus S|} \sum_{Q \subseteq S} f(Q) = \sum_{S \subseteq T} (-1)^{|T \setminus S|} [R \subseteq S]$$

$$= \sum_{R \subseteq S \subseteq T} (-1)^{|T \setminus S|}.$$

12   Covering by Möbius inversion.   We now give alternative argument for the graph colouring expression (2).
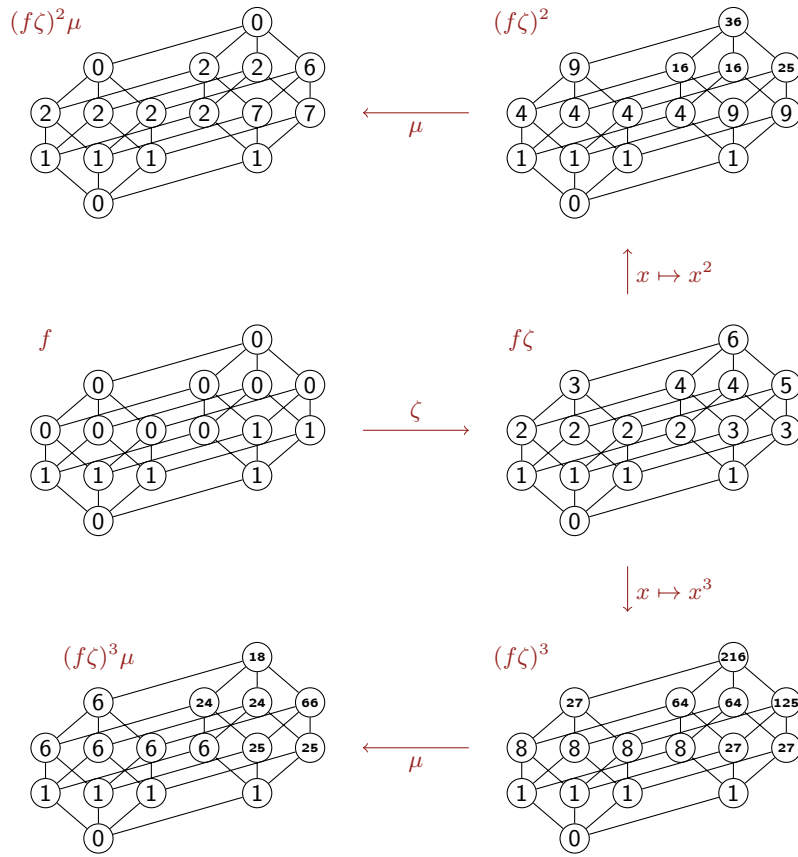
Fig. 10. Covering by Möbius inversion for $k = 2$ and $k = 3$.

Let $f\colon 2^N \to \{0, 1\}$ be the indicator function of the nonempty independent sets of a graph $G = (N, E)$. We want to count the number of ways so cover N with k nonempty independent sets. Define $g(S)$ to be the number of ways to choose k nonempty independent sets whose union is S. Then we claim

$$g\zeta = (f\zeta)^k.$$

To see this, for every $T \subseteq V$, view $g\zeta(T)$ and $\big(f\zeta(T)\big)^k$ as two different ways of counting the number of ways so select k nonempty independent subsets of T. Now, by Möbius inversion (18), we have

$$g = (f\zeta)^k \mu,$$

which is the left hand side of (2). In fact, we can now rewrite and understand the left hand side of (2) as

$$\underbrace{\sum_{S \subseteq N} (-1)^{|N \setminus S|}}_{\mu} \left(\underbrace{\sum_{R \subseteq S} \boxed{f(R)}}_{\zeta} \right)^{\boxed{k}} \quad \begin{array}{l} \textit{operation in the transformed domain} \\ \textit{function in the original domain} \end{array}$$

Perspective. The fact that f was the indicator function of the *independent sets* played no role in this argument. It works for a many covering problems, and with some work also for packing and partitioning problems.

Taxonomically, we can view inclusion–exclusion as a *transform-and-conquer* technique, like the Fourier transform. This can be expressed succinctly in terms of Möbius inversion, illustrated in Fig. 10. The zeta transform translates the original problem into a different domain, where the computation is often easier, and the Möbius transform translates back. In the covering example, the operation in the transformed domain, exponentiation, is particularly simple. The idea extends to many other operations in the transformed domain, see [3].

Concluding remarks A comprehensive presentation of many of the ideas mentioned here appears in a recent monograph [10], with many additional examples. In particular a whole chapter is devoted to subset convolution, the most glaring omission from the present introduction.

I owe much of my understanding of this topic to illuminating conversations with my collaborators, Andreas Björklund, Petteri Kaski, and Mikko Koivisto.

References

1. R. Bellman. Dynamic programming treatment of the travelling salesman problem. *J. Assoc. Comput. Mach.*, 9(1):61–63, 1962.
2. A. Björklund and T. Husfeldt. Exact algorithms for exact satisfiability and number of perfect matchings. *Algorithmica*, 52(2):226–249, 2008.
3. A. Björklund, T. Husfeldt, P. Kaski, and M. Koivisto. Fourier meets Möbius: fast subset convolution. In *Proceedings of the 39th Annual ACM Symposium on Theory of Computing (San Diego, CA, June 11–13, 2007)*, pages 67–74, New York, 2007. ACM.
4. A. Björklund, T. Husfeldt, P. Kaski, and M. Koivisto. Covering and packing in linear space. In *Automata, Languages and Programming, 37th International Colloquium, ICALP 2010 (Bordeaux, France, July 6–10)*, number 6198 in Lecture Notes in Computer Science, pages 727–737. Springer, 2010.
5. A. Björklund, T. Husfeldt, P. Kaski, and M. Koivisto. Evaluation of permanents in rings and semirings. *Inf. Process. Lett.*, 110(20):867–870, 2010.
6. A. Björklund, T. Husfeldt, P. Kaski, and M. Koivisto. Narrow sieves for parameterized paths and packings. arXiv:1007.1161, 2010.
7. A. Björklund, T. Husfeldt, P. Kaski, and M. Koivisto. Trimmed Moebius inversion and graphs of bounded degree. *Theory Comput. Syst.*, 47(3):637–654, 2010.

8. A. Björklund, T. Husfeldt, and M. Koivisto. Set partitioning via inclusion–exclusion. *SIAM J. Comput.*, 39(2):546–563, 2009.

9. R. A. DeMillo and R. J. Lipton. A probabilistic remark on algebraic program testing. *Inform. Process Lett.*, 7:193–195, 1978.

10. F. Fomin and D. Kratsch. *Exact Exponential Algorithms*. Texts in Theoretical Computer Science. Springer, 2010.

11. M. Held and R. M. Karp. A dynamic programming approach to sequencing problems. *J. Soc. Indust. Appl. Math.*, 10:196–210, 1962.

12. R. M. Karp. Dynamic programming meets the principle of inclusion and exclusion. *Oper. Res. Lett.*, 1(2):49–51, 1981/82.

13. D. E. Knuth. *The Art of Computer Programming*, volume 2, Seminumerical Algorithms. Addison–Wesley, Upper Saddle River, N.J., 3rd edition, 1998.

14. S. Kohn, A. Gottlieb, and M. Kohn. A generating function approach to the traveling salesman problem. In *Proceedings of the 1977 ACM Annual Conference (Seattle, WA, October 17–19, 1977)*, pages 294–300, New York, 1977. ACM.

15. E. L. Lawler. A note on the complexity of the chromatic number problem. *Inf. Process. Lett.*, 5(3):66–67, 1976.

16. J. Nederlof. Fast polynomial-space algorithms using Möbius inversion: Improving on Steiner tree and related problems. In *Automata, Languages and Programming, 36th International Colloquium, ICALP 2009, Rhodes, Greece, July 5–12, 2009, Proceedings, Part I*, number 5555 in Lecture Notes in Computer Science, pages 713–725. Springer, 2009.

17. H. J. Ryser. *Combinatorial Mathematics*. Number 14 in The Carus Mathematical Monographs. The Mathematical Association of America, Washington, D.C., 1963.

18. J. T. Schwartz. Fast probabilistic algorithms for verification of polynomial identities. *J. Assoc. Comput. Mach.*, 27:701–717, 1980.