Computation of A244493:
The number of oriented (0, 1, 2)-factors of the Johnson graph J(n, 2)

Ronald Niles
BSE, Cooper Union School of Engineering, MA, Courant Institute of Mathematical Sciences
rniles@yahoo.com
June 25, 2017

Version 0.147

ABSTRACT

East and Gray [1] have discussed the sequence A244493 [2] and show that it is also the number of minimal idempotent generating sets of the singular part of the Brauer monoid. Although none of the authors was aware of a closed-end formula or recursion for this sequence (d_n), they were able to present software-calculated values for d_n through n=6 [1 p.105]. This paper discusses and implements a technique for calculating two additional members of the sequence using an ordinary PC.

To calculate d_7 and d_8, a technique is presented which includes multiple applications of the inclusion-exclusion principle, transformation of the resulting summations into summations over mappings for which $|m^{-1}(y)|>1$ for all y in the image, use of symmetry and orbits to reduce the number of mappings to be evaluated, and efficient implementation for solution by computer. This technique enables the calculation of A244493 on an ordinary laptop PC, nearly instantly for n<7, in about 6 minutes for n=7, and in 24 days for n=8.

The known values of the sequence A244493 are now:

$d_2 = 1$, $d_3 = 6$, $d_4 = 265$, $d_5 = 126140$, $d_6 = 855966441$,

$d_7 = 102526835994056$, $d_8 = 258081861682902430193$

DEDICATION

To the brilliant Professor Manolis Kondopirakis (1948-2016).

TABLE OF CONTENTS

INTRODUCTION

There are several ways to describe the sequence A244493. As East and Gray have shown, it is the number of minimal idempotent generating sets of the singular part of the Brauer monoid B_n. They were able to prove a one-to-one correspondence between such sets and the number of oriented (0,1,2)-factors of the Johnson graph J(n,2) [1, p.104]. Additionally, there is a one-to-one correspondence between such oriented (0-1-2)-factors of the graph J(n,2) and the set of permutations of vertices of J(n,2) which map each vertex either to itself or to a neighboring vertex. It is these permutations which will be counted in this paper to determine members of the sequence.

The sequence grows rapidly, due in part to the rapid growth of J(n,2). Let X be the set of all two-element subsets of {1,...,n}. There are, of course, $\binom{n}{2}$ elements in X. The graph J(n,2) has its vertices in one-to-one correspondence with elements of X, and is a regular graph which connects vertices if and only if their corresponding subsets are not disjoint. It is easy to see that there are $\binom{n}{2}$ vertices and $\binom{n}{2}(n-2)$edges in J(n,2).

In order to evaluate d_n, an initial application of inclusion-exclusion is performed, beginning with the set of all permutations on the elements of X, and subsequently excluding those permutations which map at least one element of X to an element with which it is disjoint. This results in an finite alternating series of the form:

$$\sum_{k=0}^{|X|} (-1)^k C_k (|X|-k)!$$

where C_0 is 1, and for k>0, C_k is the number of partial permutations p whose domain and image are k-element subsets of X, and for which p(x) is disjoint from x for all x in its domain.

The values of C_k are then evaluated by an additional application of inclusion-exclusion, this time starting with the set of all mappings m whose domain is a k-element subset of X and whose image is a subset of X, and for which m(x) is disjoint from x for all x in the domain of m. The set of partial permutations being sought can be obtained by excluding those mappings which are not partial permutations. These mappings are partial permutations unless they have at least one element y in the image for which the number of domain elements mapped to y is larger than 1, i.e. $|m^{-1}(y)|>1$. The value $|m^{-1}(y)|$ will be referred to as the *multiplicity* of y, abbreviated mult*(y),* and mappings for which all elements in the image have multiplicity greater than one will be referred to as *fully-multiple* mappings. When inclusion-exclusion is applied, the results can be transformed into a sum over fully-multiple mappings. The problem of finding d_n is thus transformed into one in which the major task is to count all fully-multiple mappings *m* where the domain and image are subsets of X, and where *m(x)* is disjoint from *x* for all *x* in the domain.

The various fully-multiple mappings are counted by first classifying the image of the mapping, i.e. specifying which elements of X are to be in the image, and the multiplicity for each element. Such a classification of a mapping (image and multiplicity) will be referred to as an *image configuration*.

Further applications of inclusion-exclusion were considered, for example assigning domain elements freely to each image configuration, and then excluding associations which were one-to-many and therefore not mappings. However, this did not adequately simplify the problem, which was already well-suited for computer evaluation.

The graph J(n,2) is symmetric with respect to permutations of {1,...,n}, and this symmetry can be used to reduce the number of image configurations which need to be evaluated. Because of this symmetry, for any given image configuration, if the values {1,...,n} are permuted in the image, the new image has

the same number of possible mappings as the old, because the mappings being counted require *m(x)* to be disjoint from *x* and that relationship is invariant under permutation of {1,...,n}. If this permutation group acts on the set of all possible images, the resulting set of orbits can be evaluated by computing the number of mappings possible for a representative of each orbit and then multiplying the result by the orbit size. This will greatly reduce the number of mappings to be evaluated. The enumeration of these orbits is well-suited for computer evaluation.

At this point, a computer is programmed to determine all possible image configurations and a representative from each orbit. For each representative image configuration, the computer counts all possible assignments of domain elements to the specified image configuration. Various optimizations were made during the implementation process and these will be described. The results of the computations will be given, along with a consistency check of the computed results.

BIJECTION WITH CERTAIN PERMUTATIONS OF VERTICES OF J(n,2)

An oriented (0,1,2)-factor of J(n,2) is a subgraph consisting of all of the vertices of J(n,2) and some of the edges. These vertices are partitioned into three categories: one or more isolated vertices (the 0-factor), one or more pairs of vertices joined only to each other (the 1-factor) and one or more directed cycles of vertices (the oriented 2-factor). An oriented (0,1,2)-factor may be put into one-to-one correspondence with permutations p of vertices of J(n,2) for which p(x) is a neighbor of x as follows. The vertices in the 0-factor are fixed points of the permutation, the pairs of vertices in the 1-factor are 2-cycles of the permutation, and the directed cycles in the oriented 2-factor are 3- and higher- cycles of the permutation. The 3- and higher cycles of permutations can be reversed in direction resulting in a different permutation, thus it is necessary to specify *oriented* (0-1-2)-factors.

Let X be the set of all two-element subsets of {1,...,n}. The permutations in one-to-one correspondence with the various oriented (0,1,2)-factors of J(n,2) are those permutations of X for which p(x) is not disjoint from x for any x in X. These are the permutations which will be counted to determine d_n.

INCLUSION-EXCLUSION IN GENERAL

Inclusion-exclusion [3] is a principle from set theory which states that for sets A and B with finite cardinality:

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

It is called inclusion-exclusion because the sizes of both sets are included, then an adjustment is made for the overlap in the sets by excluding the intersection. Inclusion-exclusion can be extended to as many sets as desired, e.g. three sets:

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

or four sets:

$$\begin{aligned} |A \cup B \cup C \cup D| = & \\ & + (|A| + |B| + |C| + |D|) \\ & - (|A \cap B| + |A \cap C| + |A \cap D| + |B \cap C| + |B \cap D| + |C \cap D|) \\ & + (|A \cap B \cap C| + |A \cap B \cap D| + |A \cap C \cap D| + |B \cap C \cap D|) \\ & - |A \cap B \cap C \cap D| \end{aligned}$$

In words, the size of the union is equal to the sum of the sizes of all sets in the union, minus the sum of the sizes of all pairwise intersections of such sets, plus the sum of the sizes of all 3-wise intersections of such sets, minus the sum of the sizes of all 4-wise intersections, and so on. The number of terms gets large quickly but generalizations are often possible, based on large classes of intersections being empty, or large classes of intersections having the same size. Inclusion-exclusion is good for “at least one”

type of problems, where the sets A,B,C... represent different ways that “at least one” can be achieved. Additionally, inclusion-exclusion can be used to solve “no” or “none” type of problems by subtracting a union of “at least one” sets from the set of all.

Finding the number of derangements is a well-known application of inclusion-exclusion [3]. Derangements are permutations of n objects that have no fixed points. For the sake of example, let n=6, and let P_6 be the set of all permutations on {1..6}, and let D_6 be the number of derangements in P_6. The number of derangements can be calculated by subtracting the non-derangements, i.e. the number of permutations where at least one element maps to itself. Let A1 be the subset of P_6 for which 1 is fixed. Let A2 be the subset of P_6 for which 2 is fixed, and so on. The number of derangements is then $|D_6| = |P_6| - |A1 \cup A2 \cup A3 \cup A4 \cup A5 \cup A6|$, and by inclusion-exclusion:

$$|D_6| = |P_6| - |A1 \cup A2 \cup A3 \cup A4 \cup A5 \cup A6| = \begin{aligned} & 6! \\ & - \sum_{i \leq 6} |Ai| \\ & + \sum_{i<j \leq 6} |Ai \cap Aj| \\ & - \sum_{i<j<k \leq 6} |Ai \cap Aj \cap Ak| \\ & + \sum_{i<j<k<l \leq 6} |Ai \cap Aj \cap Ak \cap Al| \\ & - \sum_{i<j<k<l<m \leq 6} |Ai \cap Aj \cap Ak \cap Al \cap Am| \\ & + \sum_{i<j<k<l<m<n \leq 6} |Ai \cap Aj \cap Ak \cap Al \cap Am \cap An| \end{aligned}$$

To evaluate the sums, note that $|Ai|$ are all 5! because they have one constrained point, $|Ai \cap Aj|$ are all 4! because they have two constrained points, $|Ai \cap Aj \cap Ak|$ are all 3! because they have three constrained points, and so on. Now note that the number of such sets is

$\binom{6}{1} Ai$, $\binom{6}{2} Ai \cap Aj$, $\binom{6}{3} Ai \cap Aj \cap Ak$, $\binom{6}{4} Ai \cap Aj \cap Ak \cap Al$, $\binom{6}{5} Ai \cap Aj \cap Ak \cap Al \cap Am$ and $\binom{6}{6} Ai \cap Aj \cap Ak \cap Al \cap Am \cap An$. Putting this information together and doing some algebra with the binomials yields D_6 = 6! - 6!/1! + 6!/2! - 6!/3! + 6!/4! - 6!/5! + 6!/6! which evaluates to 720 – 720 + 360 – 120 + 30 - 6 + 1 = 265.

Note that the method used generalizes to $D_n = n! \sum_{i=0}^{n} \frac{-1^n}{i!}$ and that the summation is the beginning of the Maclaurin expansion for 1/e. Remarkably, this series converges so quickly that the number of derangements may be found for all n>0 by rounding n!/e to the nearest integer [4].

INCLUSION-EXCLUSION APPLIED TO A244493

First, a change of notation. For clarity in the following discussion, letters will be used instead of numbers for the elements of {1,...n}, and also for elements of X. Each number k in {1,...,n} will be substituted with the k-th capital letter of the alphabet. Moreover, elements of X, which are two-element subsets of {1...n}, will be denoted in shorthand by simply concatenating the two set elements in lexicographic order, e.g. {2,4} will be denoted BD (but never DB). Let P be the set of all permutations of X.

The application of inclusion-exclusion to the set of all permutations of X where p(x) is not disjoint from x for any x in X somewhat parallels the method in which it is used to calculate derangements in the previous section. For example, d_4 can be determined by considering permutations of X={AB,

AC,AD,BC,BD,CD} for which p(x) is not disjoint from x for any x in X. The number of such permutations is determined by excluding from the set of all permutations those which have at least one p(x) disjoint from x. That set can be described as the union:

$$|\{P \mid AB \to CD\} \cup \{P \mid AC \to BD\} \cup \{P \mid AD \to BC\} \cup \{P \mid BC \to AD\} \cup \{P \mid BD \to AC\} \cup \{P \mid CD \to AB\}|$$

Note that any intersection of these sets can be described by combining restrictions, e.g.

$$\{P \mid AB \to CD\} \cap \{P \mid AC \to BD\} = \{P \mid AB \to CD \wedge AC \to BD\} .$$

Note also that for n=4, the restrictions taken together always describe a partial permutation, since each element of X appears exactly once in the domain or image of any restriction. This will not be the case for larger n. Let P1 be the subsets of P which have 1 restriction, P2 the subsets of P which have two restrictions, and so on up to P6. Then, the logic proceeds similarly as to determining the number of derangements:

$$d_4 = 6! - \sum_{x \in P1} |x| + \sum_{x \in P2} |x| - \sum_{x \in P3} |x| + \sum_{x \in P4} |x| - \sum_{x \in P5} |x| + \sum_{x \in P6} |x|$$

In P1, the subsets have size 5! and there are 6 such subsets

In P2, the subsets have size 4! and there are $\binom{6}{2}$ such subsets

In P3, the subsets have size 3! and there are $\binom{6}{3}$ such subsets

In P4, the subsets have size 2! and there are $\binom{6}{4}$ such subsets

In P5, the subsets have size 1! and there are $\binom{6}{5}$ such subsets

In P6, the subsets have size 0! and there are $\binom{6}{6}$ such subsets

$$d_4 = 6! - 6 \times 5! + 15 \times 4! - 20 \times 3! + 15 \times 2! - 6 \times 1! + 1 \times 0!$$

$$d_4 = 720 - 720 + 360 - 120 + 30 - 6 + 1 = 265$$

The result and the reasoning are identical to those used in calculating the number of derangements on permutations of six objects described in the previous section, because both problems require a subset of the permutations of six items, each of which item has a single and unique prohibited element in the domain. For n>4, each element has $\binom{n-2}{2}$ prohibited elements, e.g. for n=5, there are 3 destinations prohibited for each of the 10 elements of X, and the 10 x 3 base set for inclusion-exclusion becomes:

$$\{P \mid AB \to CD\}, \{P \mid AB \to CE\}, \{P \mid AB \to DE\}, \{P \mid AC \to BD\}, \{P \mid AC \to BE\}, \{P \mid AC \to DE\}$$

$$\{P \mid AD \to BC\}, \{P \mid AD \to BE\}, \{P \mid AD \to CE\}, \{P \mid AE \to BC\}, \{P \mid AE \to BD\}, \{P \mid AE \to CD\}$$

$$\{P \mid BC \to AD\}, \{P \mid BC \to AE\}, \{P \mid BC \to DE\}, \{P \mid BD \to AC\}, \{P \mid BD \to AE\}, \{P \mid BD \to CE\}$$

$$\{P \mid BE \to AC\}, \{P \mid BE \to AD\}, \{P \mid BE \to CD\}, \{P \mid CD \to AB\}, \{P \mid CD \to AE\}, \{P \mid CD \to BE\}$$

$$\{P \mid CE \to AB\}, \{P \mid CE \to AD\}, \{P \mid CE \to BD\}, \{P \mid DE \to AB\}, \{P \mid DE \to AC\}, \{P \mid DE \to BC\}$$

P1, the set of permutation sets with a single restriction is the set of base sets just above. Since each base set has a single restriction, there are (10-1) free elements so each base set has size 9! Taking pairwise

intersections of these sets to form P2, notice that there are some sets in P2 where the combined restrictions describe a partial permutation, e.g. $\{P \mid AB \to CD \wedge AC \to CE\}$, all of which have size 8!, and other pairwise intersections which are empty sets, such as $\{P \mid AB \to CD \wedge AB \to CE\}$, since no permutation can map AB to two different elements, or $\{P \mid AB \to CD \wedge AE \to CD\}$, since no permutation can map two different elements to the same element. Similar logic applies for P3, P4, …,P10. By definition, it follows that the intersections of any number of elements in the base set are non-empty if and only if the combined restrictions of the intersecting sets all have unique domain elements and unique image elements, i.e. they must form a *partial permutation*. Let us introduce some concepts and definitions to formalize the argument.

- Let the *size of a mapping* be defined as the size of its domain, i.e. $|m| = |dom(m)|$.
- Let the concept of a *map extension* be defined as follows. If m' and m are mappings, then m' is an *extension* of m if the domain of m is a subset of the domain of m' and m'(x) = m(x) for any x in the domain of m. For the purposes of this definition, a map is considered to be an extension of itself.
- Let the concept of a *map union* be defined as follows. If m1 and m2 are maps with domains d1 and d2 respectively, and m1(x) = m2(x) for all x in the intersection of d1 and d2, then the *union* m of m1 and m2 is the mapping defined on the union of d1 and d2 such that m(x) = m1(x) if x is in d1, m(x) = m2(x) otherwise.

Let x and y be elements of X, and let m_{xy} be the mapping of size 1 which maps x to y. For any mapping m whose domain and image are subsets of X, Let E(m) be the set of all elements of P which are extensions of m.

Let the base set $B=\{E(m_{xy}) | x, y \in X \wedge x \cap y = \emptyset\}$ In general, there will be $|X| = \binom{n}{2}$ ways to choose x coupled with $\binom{n-2}{2}$ ways to choose y which is disjoint from x, thus $\binom{n}{2} \times \binom{n-2}{2}$ elements in the base set.

Lemma 1: The union of all elements in the base set B is precisely the set of permutations of X which map at least one element to an element with which it is disjoint.

Proof: If p is in the union of base sets, it must be in at least one of those base sets, and each base set consists of permutations which are extensions of a mapping which maps an element to a disjoint element. Conversely, if p is a permutation for which p(x)=y and x is disjoint from y, then p is an extension of the mapping m_{xy}, therefore $p \in \{E(m_{xy})\}$ and $\{E(m_{xy})\}$ is a member of the base set B since B consists of all such sets where x and y are disjoint. It follows that p is in the union of all base sets if and only if p maps at least one element to an element with which it is disjoint.

Lemma 2: $d_n = |X|! - \left| \bigcup_{x,y \in X, x \cap y = \emptyset} E(m_{xy}) \right|$

Proof: d_n is the number of permutations of X for which p(x) is not disjoint from x for any x in X. The total number of permutations is $|X|!$, so by subtracting those permutations for which there is at least one value of x for which p(x) is disjoint from x, it follows from Lemma 1 that:

$$d_n = |X|! - \left| \bigcup_{b \in B} b \right| = |X|! - \left| \bigcup_{x,y \in X, x \cap y = \emptyset} E(m_{xy}) \right|$$

Lemma 3: Let m1 and m2 be mappings whose domains and images are subsets of X.

$$E(m1)\cap E(m2)=\begin{cases}E(m1\cup m2), & \text{if } m1\cup m2 \text{ is defined}\\ \varnothing, & \text{otherwise}\end{cases}$$

Proof: if p is in the intersection of E(m1) and E(m2), then p(x) = m1(x) for all x in the domain of m1 and p(x)=m2(x) for all x in the domain of m2. If m1(x) is unequal to m2(x) for any x in the intersection of their domains, then the intersection is null since no permutation can map the value x to two different values. Alternately, if m1(x) is equal to m2(x) for all x in the intersection of their domains, then the union is defined, and $p(x)=(m1\cup m2)(x)$ for all x in the domain of the union, so p(x) is an extension of the union.

Lemma 4: if m is a mapping of size k whose domain and image are subsets of X, then

$$|E(m)|=\begin{cases}(|X|-k)!, & \text{if m is a partial permutation}\\ 0, & \text{otherwise}\end{cases}$$

Proof: If m is not a partial permutation, then m(x) must equal m(y) for some distinct x and y, and this will be true for any extension of m as well. Since E(m) consists of permutations, it is not possible that a member of E(m) can map two distinct elements to the same value, so $|E(m)|=0$ if m is not a partial permutation. If m is a partial permutation of size k, then there are $|X|-k$ elements of X which are not in the domain of m, and $|X|-k$ elements of X which are not in the image of m, and these subsets of the domain and image may be paired in $(|X|-k)!$ different ways to produce mappings which when combined with m produce a distinct, full permutation which is a member of E(m).

Theorem: Let D_k be the set of all partial permutations p(x) whose domain and image are both subsets of X of size k, and for which p(x) is disjoint from x for all x in the domain of p. Then

$$d_n=\sum_{k=0}^{|X|}(-1)^k C_k(|X|-k)!, \text{ where } |X|=\binom{n}{2} \text{ and } C_k=\begin{cases}1, & \text{if k=0}\\ |D_k|, & \text{otherwise}\end{cases}$$

Proof: By lemma 2:

$$d_n = |X|!-\left|\bigcup_{x,y\in X, x\cap y=\varnothing} E(m_{xy})\right|$$

By inclusion-exclusion on the size of the union,

$$d_n=|X|!-\sum_{k=1}^{|X|}(-1)^{(k+1)}\sum_{I\in \text{k-wise intersections of } E(m_{xy})}|I|$$

Let I'_k be the set of k-wise intersections of m_{xy} for which all pairwise unions of such m_{xy} are defined. By lemma 3, the k-wise intersections which are not in I'_k are null and do not contribute to the sum, so the sum can be taken over I'_k :

$$d_n=|X|!-\sum_{k=1}^{|X|}(-1)^{(k+1)}\sum_{I\in I'_k}|I|$$

By lemma 3, k-wise intersections of extensions of mappings are extensions of k-wise unions of mappings when defined:

$$d_n=|X|!-\sum_{k=1}^{|X|}(-1)^{(k+1)}\sum_{U\in \text{k-wise unions of } m_{xy}}|E(U)|$$

By lemma 4, the size of U is zero when the m_{xy} do not combine to form a partial permutation, which must be a partial permutation of size k, so the second summation can be taken over D_k.

$$d_n = |X|! - \sum_{k=1}^{|X|} (-1)^{(k+1)} \sum_{p \in D_k} |E(p)|$$

Also by lemma 4, since p is a partial permutation of size k, the size of E(p) will be $(|X|-k)!$ which is the same for all $p \in D_k$, the summation becomes a product:

$$d_n = |X|! - \sum_{k=1}^{|X|} (-1)^{(k+1)} |D_k| (|X|-k)!$$

Which, after some algebra and making the size of X explicit, proves the theorem.

TABLE OF BASE SET MAPPINGS FOR n=5

Table 1: Arrangement of mappings m_{xy} for n=5. There is one row for each element *x*, consisting of mappings of *x* to those elements for which *x* is disjoint. If one entry is selected from each of any k rows in the table, the union of those entries form a mapping of size k. This mapping is in general many-to-one, but may be one-to-one in which case the k entries describe a partial permutation. The coefficients for C_k for k>0 are the number of partial permutations that can be formed by combining one entry from each of k rows in the table:

$\begin{pmatrix} AB \\ CD \end{pmatrix}$	$\begin{pmatrix} AB \\ CE \end{pmatrix}$	$\begin{pmatrix} AB \\ DE \end{pmatrix}$
$\begin{pmatrix} AC \\ BD \end{pmatrix}$	$\begin{pmatrix} AC \\ BE \end{pmatrix}$	$\begin{pmatrix} AC \\ DE \end{pmatrix}$
$\begin{pmatrix} AD \\ BC \end{pmatrix}$	$\begin{pmatrix} AD \\ BE \end{pmatrix}$	$\begin{pmatrix} AD \\ CE \end{pmatrix}$
$\begin{pmatrix} AE \\ BC \end{pmatrix}$	$\begin{pmatrix} AE \\ BD \end{pmatrix}$	$\begin{pmatrix} AE \\ CD \end{pmatrix}$
$\begin{pmatrix} BC \\ AD \end{pmatrix}$	$\begin{pmatrix} BC \\ AE \end{pmatrix}$	$\begin{pmatrix} BC \\ DE \end{pmatrix}$
$\begin{pmatrix} BD \\ AC \end{pmatrix}$	$\begin{pmatrix} BD \\ AE \end{pmatrix}$	$\begin{pmatrix} BD \\ CE \end{pmatrix}$
$\begin{pmatrix} BE \\ AC \end{pmatrix}$	$\begin{pmatrix} BE \\ AD \end{pmatrix}$	$\begin{pmatrix} BE \\ CD \end{pmatrix}$
$\begin{pmatrix} CD \\ AB \end{pmatrix}$	$\begin{pmatrix} CD \\ AE \end{pmatrix}$	$\begin{pmatrix} CD \\ BE \end{pmatrix}$
$\begin{pmatrix} CE \\ AB \end{pmatrix}$	$\begin{pmatrix} CE \\ AD \end{pmatrix}$	$\begin{pmatrix} CE \\ BD \end{pmatrix}$
$\begin{pmatrix} DE \\ AB \end{pmatrix}$	$\begin{pmatrix} DE \\ AC \end{pmatrix}$	$\begin{pmatrix} DE \\ BC \end{pmatrix}$

Higher n have a similar table with $\binom{n}{2}$ rows and $\binom{n-2}{2}$ columns.

INCLUSION-EXCLUSION APPLIED TO THE COEFFICIENTS C_k

The C_k as described above require the number of partial permutations p of size k for which p(x) is

disjoint from x. Let us use inclusion-exclusion again to exclude from the set of all such mappings of size k, those which are not partial permutations.

Let M_k be the set of all mappings of size k, for which m(x) is disjoint from x for all x in the domain of m. Note that the size of M_k is easy to compute, since any choice of k items from X can each be freely paired with any of the items in X which are disjoint with it, therefore:

$$|M_k| = \left(\binom{n}{2}\right) \times \binom{n-2}{2}^k .$$

Consequently, since mappings of size k=1 are all partial permutations, $C_1 = |M_1| = \left(\binom{n}{2}\right) \times \binom{n-2}{2}$.

For k larger than 1, not all mappings in M_k are partial permutations, so let us formulate a union of sets which are not partial permutations, and subtract the size of this union from $|M_k|$.

A member of M_k is a partial permutation if and only if the *multiplicity* $|m^{-1}(y)|$, of each element in the image is 1. Let us define a base set B of mappings whose domain has size two and whose image has size one, i.e. the image has a single element with multiplicity two. Therefore,

$$B = \{b \in M_2 | \forall x, y \in \mathrm{dom}(b), b(x) = b(y)\}$$

Remark: As there are $\binom{n}{2}$ elements which could be the image, and $\left(\binom{n-2}{2}\right)$ ways to choose two elements for the domain which are both disjoint with it, the size of the base set B is $\binom{n}{2} \times \left(\binom{n-2}{2}\right)$.

Lemma 5: The subset of M_k consisting of all mappings which are an extension of a mapping in B is precisely the subset of M_k which are not partial permutations.

Proof: Let p be a mapping in M_k such that p is a partial permutation. If p is also an extension of a mapping b in B, then b(x) = b(y) for some distinct x and y, and therefore p(x) = p(y) since p is an extension of b, which contradicts the fact that partial permutations such as p cannot map two distinct elements to the same value, therefore p is not an extension of any b in B. Alternately, if m is a mapping of size k>1 which is not a partial permutation, then there are two distinct values x and y for which p(x) = p(y) = z, and since B contains all such two-element mappings such that b(x)=b(y)=z, it follows that m is an extension of a mapping in b. Therefore a mapping in M_k is not a partial permutation if and only if it is an extension of a mapping in the base set B.

Lemma 6: If b is a mapping in B, and $E_k(b)$ is defined to be the subset of M_k which are extensions of b, then for k > 0:

$$C_k = |M_k| - \left| \bigcup_{b \in B} E_k(b) \right|$$

Proof: For k>0, C_k is the number of partial permutations of size k which are elements of M_k. By lemma 5, the elements of M_k which are not partial permutations is precisely the subset of M_k which are extensions of elements in b, which is given by the union over B of extensions of all its elements.

Inclusion-exclusion can be applied to the union. This time, instead of summing over i-wise intersections of the base set, let us use an alternate form which sums over non-empty subsets of the base set. First let us define S as the set of all non-empty subsets of B. If s is an element of S, let

$I(s) = \bigcap_{b \in s} E_k(b)$, in other words the intersection of extensions of all the base sets included in s. Then by inclusion-exclusion,

$$\left| \bigcup_{b \in B} E_k(b) \right| = \sum_{s \in S} (-1)^{|s|+1} |I(s)|.$$

Lemma 3 applies to these intersections, with one new detail: it is possible that the size of the union $b_1 \cup b_2$ exceeds k, in which case the intersection is empty as an extension to a smaller size is impossible:

$$E_k(b_1) \cap E_k(b_2) = \begin{cases} E_k(b_1 \cup b_2), \text{ if } b_1 \cup b_2 \text{ is defined and } |b_1 \cup b_2| \leqslant k \\ \varnothing \text{ otherwise} \end{cases}$$

This can be extended to larger intersections, which are empty unless all pairwise unions are defined:

$$\bigcap_{i=1}^{m} E_k(b_i) = \begin{cases} E_k\left(\bigcup_{i=1}^{m} b_i\right), \text{ if } b_i \cup b_j \text{ is defined for all } i < j \leqslant m, \text{ and } \left|\mathrm{dom}\left(\bigcup_{i=1}^{m} b_i\right)\right| \leqslant k \\ \varnothing, \text{ otherwise} \end{cases}$$

It follows that $I(s) = \begin{cases} E_k\left(\bigcup_{b \in s} b\right), \text{ if all pairwise unions in s are defined and } \left|\mathrm{dom}\left(\bigcup_{b \in s} b\right)\right| \leqslant k \\ \varnothing, \text{ otherwise} \end{cases}$

Let us define S' to be the subset of S for which s in S' has all pairwise unions of its members defined and the size of the domain of their union is not greater than k. Since I(s) is empty for these excluded sets, they do not contribute to the summation, and they can be excluded from the summation. If S' is substituted for S, all of the subsets summed over will have unions whose size does not exceed k, so I(s) can be replaced:

$$\left| \bigcup_{b \in B} E_k(b) \right| = \sum_{s \in S'} (-1)^{|s|+1} \left| E_k\left(\bigcup_{b \in s} b\right) \right|$$

For any subset of the base mappings b in B with all pairwise unions defined, the union of these mappings will be a *fully-multiple mapping*, i.e. it will have mult(y)>1 for each y in its image. There is a finite set of fully-multiple mappings whose size does not exceed k, and these mappings partition the elements of S' over which the summation is taken. Because of this partitioning, it is possible to regroup the summations so that the sum is taken over the set of all possible fully-multiple mappings. Let T_j be the set of all fully-multiple mappings in M_j. By definition, these mappings are all of size j.

Let $t \in T_j, 2 \leqslant j \leqslant k$ and define $U(t) = \{ s \in S' \mid \bigcup_{b \in s} b = t \}$, i.e. U(t) consists of all subsets of the base set B whose union of elements is the fully-multiple mapping t. By regrouping the elements being summed according to the partition:

$$\left| \bigcup_{b \in B} E_k(b) \right| = \sum_{j=2}^{k} \sum_{t \in T_j} \sum_{s \in U(t)} (-1)^{|s|+1} \left| E_k\left(\bigcup_{b \in s} b\right) \right|$$

The first two summations are over all fully-multiple mappings t of size less than or equal to k. The unions $\bigcup_{b \in s} b$ all evaluate to t, and are therefore all the same size, namely the size of t, and since t is in T_j its size is j, so the number of extensions of t, $|E_k(t)|$ is just the way to select (k-j) domain elements from the remaining $\binom{n}{2} - j$ domain elements, and then select one of $\binom{n-2}{2}$ independent choices of disjoint image elements for each domain element, therefore $|E_k(t)| = \binom{\binom{n}{2} - j}{k-j} \times \binom{n-2}{2}^{(k-j)}$ and for brevity, let's call this $E_{j.k.n}$. This may be brought outside of the last summation:

$$\left|\underset{b\in B}{\cup} E_k(b)\right| = \sum_{j=2}^{k} E_{j.k.n} \sum_{t\in T_j} \sum_{s\in U(t)} (-1)^{(|s|+1)}$$

Let us once more partition the summation over U(t), which are unions of base set elements that equal t, into unions of a specific number (i) of base set elements that equal t. Define
$U(t,i) = \{s\in S' \mid \underset{b\in s}{\cup} b = t \wedge |s| = i\}$, then

$$\left|\underset{b\in B}{\cup} E_k(b)\right| = \sum_{j=2}^{k} E_{j.k.n} \sum_{t\in T_j} \sum_{i=1}^{|B|} \sum_{s\in U(t,i)} (-1)^{(|s|+1)},$$

which simplifies to:

$$\left|\underset{b\in B}{\cup} E_k(b)\right| = \sum_{j=2}^{k} E_{j.k.n} \sum_{t\in T_j} \sum_{i=1}^{|B|} (-1)^{(i+1)} |U(t,i)|$$

The final summation involving U(t,i) requires enumeration of the various ways in which a fully-multiple mapping can be decomposed into different unions of mappings in the base set. Further simplification is possible as shown in the next section.

DECOMPOSING FULLY-MULTIPLE MAPPINGS INTO UNIONS OF BASE SETS

Let us first consider some examples to illustrate the required decompositions. A simple example would be the case where t has a single element in its image, say AB. The multiplicity of (AB) must be at least two since t is a fully-multiple mapping. Let us manually decompose such mappings into unions of members of the base set, and then group them according to size and compute the alternating sum.

If AB has multiplicity 2, in other words the domain of t is composed of two distinct members of X such that AB is disjoint from both of them, then t must be an element of the base set B, and there is only one subset of B of size one whose union is t, namely the singleton subset t. So there is only one decomposition of t containing 1 set, so $|U(t,i)| = \begin{cases} 1, & if\ i=1 \\ 0, & otherwise \end{cases}$, and therefore
$\sum_{i=1}^{|B|} (-1)^{(i+1)} |U(t,i)| = (-1)^2 \times 1 = 1.$

If AB has multiplicity 3, for example t(CD) = t(CE) = t(DE) = AB, then there are $\binom{3}{2} = 3$ base sets, $b_{CD,CE\to AB}$, $b_{CD,DE\to AB}$, $b_{CE,DE\to AB}$ for which any pairwise or three-wise union will equal t. For this case:

$|U(t,i)| = \begin{cases} 3, & if\ i=2 \\ 1, & if\ i=3 \\ 0, & otherwise \end{cases}$ and therefore $\sum_{i=1}^{|B|} (-1)^{(i+1)} |U(t,i)| = (-1)^3 \times 3 + (-1)^4 \times 1 = -2.$

If AB has multiplicity 4, for example t(CD) = t(CE) = t(CF) = t(DE) =AB, then there are $\binom{4}{2} = 6$ base sets which can possibly be extended to t, namely $b_{CD,CE\to AB}$, $b_{CD,CF\to AB}$, $b_{CD,DE\to AB}$,$b_{CE,CF\to AB}$,$b_{CE,DE\to AB}$, $b_{CF,DE\to AB}$. Which unions of these base mappings equal t?

In order for a union of two of these six base sets to equal t, the domains of b need to be disjoint, and there are three ways to do this, namely:

$$t = b_{CD,CE\to AB} \cup b_{CF,DE\to AB},\ t = b_{CD,CF\to AB} \cup b_{CE,DE\to AB},\ t = b_{CD,DE\to AB} \cup b_{CE,CF\to AB}$$

In order for a union of three of these six base sets to equal t, there are 16 ways, which consist of the

$\binom{6}{3}=20$ possible three-wise unions, minus four whose union has a single element of multiplicity three.

$$t=b_{CD,\,CE\to AB}\cup b_{CD,\,CF\to AB}\cup b_{CD,\,DE\to AB},\ t=b_{CD,\,CE\to AB}\cup b_{CD,\,CF\to AB}\cup b_{CE,\,DE\to AB},$$
$$t=b_{CD,\,CE\to AB}\cup b_{CD,\,CF\to AB}\cup b_{CF,\,DE\to AB},\ t=b_{CD,\,CE\to AB}\cup b_{CD,\,DE\to AB}\cup b_{CE,\,CF\to AB},$$
$$t=b_{CD,\,CE\to AB}\cup b_{CD,\,DE\to AB}\cup b_{CF,\,DE\to AB},\ t=b_{CD,\,CE\to AB}\cup b_{CE,\,CF\to AB}\cup b_{CE,\,DE\to AB},$$
$$t=b_{CD,\,CE\to AB}\cup b_{CE,\,CF\to AB}\cup b_{CF,\,DE\to AB},\ t=b_{CD,\,CE\to AB}\cup b_{CE,\,DE\to AB}\cup b_{CF,\,DE\to AB},$$
$$t=b_{CD,\,CF\to AB}\cup b_{CD,\,DE\to AB}\cup b_{CE,\,CF\to AB},\ t=b_{CD,\,CF\to AB}\cup b_{CD,\,DE\to AB}\cup b_{CE,\,DE\to AB},$$
$$t=b_{CD,\,CF\to AB}\cup b_{CE,\,CF\to AB}\cup b_{CE,\,DE\to AB},\ t=b_{CD,\,CF\to AB}\cup b_{CE,\,CF\to AB}\cup b_{CF,\,DE\to AB},$$
$$t=b_{CD,\,CF\to AB}\cup b_{CE,\,DE\to AB}\cup b_{CF,\,DE\to AB},\ t=b_{CD,\,DE\to AB}\cup b_{CE,\,CF\to AB}\cup b_{CE,\,DE\to AB},$$
$$t=b_{CD,\,DE\to AB}\cup b_{CE,\,CF\to AB}\cup b_{CF,\,DE\to AB},\ t=b_{CD,\,DE\to AB}\cup b_{CE,\,DE\to AB}\cup b_{CF,\,DE\to AB}$$

All four-wise, five-wise and six-wise unions equal t, since there is no way for them to form anything smaller. Therefore:

$$|U(t,i)|=\begin{cases}3, & \text{if } i=2\\ 16, & \text{if } i=3\\ 15, & \text{if } i=4\\ 6, & \text{if } i=5\\ 1, & \text{if } i=6\\ 0, & \text{otherwise}\end{cases}\text{, and therefore}$$

$$\sum_{i=1}^{|B|}(-1)^{(i+1)}|U(t,i)|=(-1)^3\times 3+(-1)^4\times 16+(-1)^5\times 15+(-1)^6\times 6+(-1)^7\times 1=3.$$

This analysis suggests that there is a method of computing these values recursively. Taking the union of base sets whose image consists of the singleton set {AB} in an attempt to construct the mapping *t* is essentially taking the union of the domains of elements in the base set whose image is AB and whose domains are two-element subsets of the domain of *t*. Not all of these unions of two-element subsets will form the domain of t – some unions might be smaller. Let us quantify this.

Let T(k,i) be the number of ways in which (i) different two-element subsets of a set S of size (k) can be chosen such that the union of these two-element subsets is S. Note that the (i) subsets which don't combine to form S form one of the $\binom{k}{k-1}$ subsets of size k-1 in one of T(k-1,i) different ways, or one of the $\binom{k}{k-2}$ subsets of size k-2 in one of T(k-2,i) different ways, and so on down to subsets of size 2. This allows the values of T(k,i) to be computed recursively as shown in Table 2.

Table 2: T(k,i). Let P be a set of size k, and consider the set Q of all two-element subsets of P. How many subsets R of Q of size i are there such that the union of all elements in R is equal to P?

k \ i	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
2	1														
3		3	1												
4		3	16	15	6	1									
5			30	135	222	205	120	45	10	1					
6			15	330	1581	3760	5715	6165	4945	2997	1365	455	105	15	1

To see how the recursion works, notice:

- Row 3 would have been $\binom{\binom{3}{2}}{i}$, but row 2 was multiplied by $\binom{3}{2}$ and subtracted.
- Row 4 would have been $\binom{\binom{4}{2}}{i}$, but row 3 was multiplied by $\binom{4}{3}$ and subtracted, and row 2 was multiplied by $\binom{4}{2}$ and subtracted
- Row 5 would have been $\binom{\binom{5}{2}}{i}$, but row 4 was multiplied by $\binom{5}{4}$ and subtracted, row 3 was multiplied by $\binom{5}{3}$ and subtracted, and row 2 was multiplied by $\binom{5}{2}$ and subtracted.

In other words, $T(k,i)=\binom{\binom{k}{2}}{i}-\sum_{j=1}^{k-2}\binom{k}{j}T(k-j,i)$

For purposes of inclusion-exclusion, the alternating sum of each row is of importance, not the individual values within. Computing the alternating sums for each row in the table, gives 1, -2, 3, -4,5, or in general, $(-1)^k\times(k-1)$. This can be proven by induction as follows. Consider $S_k=\sum_{j=1}^{\binom{k}{2}}(-1)^j T(k,j)$. Evaluation at k=2 gives $S_2 = -1$. Now claim that $S_k = (-1)^{(k-1)}\times(k-1)$, Assume that this is true for all values j < k. By the recursion, $S_k=\sum_{j=1}^{\binom{k}{2}}(-1)^j\binom{\binom{k}{2}}{i}-\sum_{j=1}^{k-2}\binom{k}{j}S_j$.

Now $\sum_{j=1}^{\binom{k}{2}}(-1)^j\binom{\binom{k}{2}}{i}=-1$, since adding 1 to the summation gives the full expansion of $(x+1)^{\binom{k}{2}}$ evaluated at x = -1, and substituting the assumed values of S_j gives

$S_k = -1 - \sum_{j=1}^{k-2} \binom{k}{j} (-1)^{(k-j-1)} \times (k-j-1)$. By rewriting the summation as a double summation in triangular form, $S_k = -1 - \sum_{j=1}^{k-2} \sum_{l=1}^{j} \binom{k}{l} (-1)^{(k-l-1)}$, and by multiplying the sum by -1, $S_k = -1 + \sum_{j=1}^{k-2} \sum_{l=1}^{j} \binom{k}{l} (-1)^{(k-l)}$. The inner summation is the alternating sum of consecutive binomial terms, which simplifies using $\binom{k}{l} = \binom{k-1}{l-1} + \binom{k-1}{l}$ so substituting $\sum_{l=1}^{j} \binom{k}{l} (-1)^{(k-l)} = (-1)^{(k-1)} + \binom{k-1}{j} (-1)^{(k-j)}$ gives $S_k = -1 + \sum_{j=1}^{k-2} (-1)^{(k-1)} + \sum_{j=1}^{k-2} \binom{k-1}{j} (-1)^{(k-j)}$. The first summation is $(-1)^{(k-1)} \times (k-2)$ and the second summation is again the alternating sum of consecutive binomial terms and is equal to $\binom{k-2}{0} (-1)^{(k-1)} + \binom{k-2}{k-2} (-1)^{(k-(k-2))} = (-1)^{(k-1)} + 1$. Substituting these gives $S_k = -1 + (-1)^{(k-1)} \times (k-2) + (-1)^{(k-1)} + 1 = (-1)^{(k-1)} \times (k-1)$, which completes the proof by induction.

Note that $\sum_{i=1}^{|B|} (-1)^{(i+1)} |U(t,i)|$ is the negative of this value, i.e. $\sum_{i=1}^{|B|} (-1)^{(i+1)} |U(t,i)| = -S_{mult(y)}$.

In some sense, the quantity $\sum_{i=1}^{|B|} (-1)^{(i+1)} |U(t,i)|$ can be thought of as the "net value" of the mapping t towards inclusion-exclusion, so let us call it v(t) for *value*. For a fully multiple mapping *t* with a single element y in its image, $v(t) = -(-1)^{mult(y)-1} \times (mult(y)-1)$.

To generalize now to fully-multiple mappings *t* that have more than one element in the image, first consider mappings *t* with two elements in the image, y_1 with multiplicity $mult(y_1)$ and y_2 with multiplicity $mult(y_2)$. Suppose S is a set of base set mappings whose union is the fully-multiple mapping *t*. Since the image of *t* is $\{y_1, y_2\}$ then S can be partitioned into $S_1 \cup S_2$ where S_1 consists of base set elements whose image is y_1 and S_2 consists of base set elements whose image is y_2. The domain of the union of all base set mappings in S_1 is $t^{-1}(y_1)$, and the domain of the union of all base set mappings in S_2 is $t^{-1}(y_2)$. Let t_1 be the restriction of *t* to $t^{-1}(y_1)$, and let t_2 be the restriction of *t* to $t^{-1}(y_2)$. Then $t_1 \cup t_2 = t$. To decompose *t* into a union of base set mappings, it is necessary and sufficient to decompose t_1 into a union of base set mappings S_1 and t_2 into a union of base set mappings S_2, then $S = S_1 \cup S_2$ is a decomposition of *t* into base set mappings.

Now, if $mult(y_1) = k_1$, then row k_1 of Table 2 gives the number of ways in which *i* base set mappings can combine to form t_1, and similarly, if $mult(y_2) = k_2$, then row k_2 of Table 2 gives the number of ways in which *i* base set mappings can combine to form t_2. Consider the full mapping $t = t_1 \cup t_2$, and suppose we want to know the number of ways in which *i* base set mappings can combine to form *t*.

Suppose S is a set of *i* base set mappings that combine to form *t*. Then these base set mappings can be partitioned into i_1 base set mappings that combine to form t_1 and i_2 base set mappings that combine to form t_2. Now Table 2 gives the $T(k_1, i_1)$ ways to choose i_1 base set mappings that combine to form t_1, and $T(k_2, i_2)$ ways to choose i_2 base set mappings that combine to form t_2. Multiplying these together, gives $T(k_1, i_1)\ T(k_2, i_2)$ ways to form *t* out i_1 base set mappings whose image is y_1, and i_2 base set mappings whose image is i_2. However, (i_1, i_2) is not necessarily the only way to partition the number of base set

mappings. To find the full number of ways to form t out of base set mappings, we must sum over all pairs of (i_1, i_2) whose sum is n, i.e. $\sum_{(i_1+i_2)=n} T(k_1,i_1)T(k_2,i_2)$. This is exactly the way polynomial multiplication works.

Now form polynomials $p_1(x)$ and $p_2(x)$ from the values in the respective rows of Table 2, i.e. $p_1(x)=\sum T(k_1,i)x^i$ and $p_2(x)=\sum T(k_2,i)x^i$ where the sums are taken over the nonzero values of the table row, and consider the product of these polynomials. For example, if mult(y_1) = 3 and mult(y_2) = 4, then $p_1(x)=3x^2+x^3$, $p_2(x)=3x^2+16x^3+15x^4+6x^5+x^6$, and the product polynomial is $p_1p_2(x)=9x^4+51x^5+61x^6+33x^7+9x^8+x^9$. Because polynomial multiplication follows the same rules as computing the number of ways to form t, the polynomial p_1p_2 shows that there are 9 ways to form t out of the union of 4 base set elements, 51 ways to form t out of the union of 5 base set elements, and so on.

It follows that v(t) is simply -1 times the polynomial $p_1p_2(x)$ evaluated at x= -1. For fully-multiple mappings where y is the only element in the image, $v(t) = -(-1)^{mult(y)-1}\times(mult(y)-1)$, so it follows that $p_1(-1) = (-1)^{mult(y_1)-1}\times(mult(y_1)-1)$ and $p_2(-1) = (-1)^{mult(y_2)-1}\times(mult(y_2)-1)$, therefore

$$v(t) = -p_1(-1)p_2(-1) = -((-1)^{mult(y_1)-1}\times(mult(y_1)-1))\times((-1)^{mult(y_1)-1}\times(mult(y_1)-1))$$

and this extends easily to fully-multiple mappings with any number of elements in the image:

$$v(t) = -\prod_{y\in image(t)}(-1)^{mult(y)-1}\times(mult(y)-1)$$

This result is somewhat remarkable because v(t) transforms a sum over a large number of inclusion-exclusion intersections into the product of a few small integers. It can be substituted in the summation above to give

$$\left|\bigcup_{b\in B}E_k(b)\right|=\sum_{j=2}^{k}E_{j.k.n}\sum_{t\in T_j}v(t).$$

Substituting to get equations for the coefficients:

$$\begin{cases} C_0=1 \\ C_1=|M_1| \\ C_k=|M_k|-\sum_{j=2}^{k}E_{j.k.n}\sum_{t\in T_j}v(t) \end{cases}$$

Where $|M_k| = \binom{n}{2}\binom{n-2}{2}^k$, $E_{j.k.n}=\binom{n}{2}-j\binom{n-2}{2}^{(k-j)}$ and

$$v(t) = -\prod_{y\in image(t)}(-1)^{mult(y)-1}(mult(y)-1)$$

The initial sum over inclusion-exclusion intersections has been transformed into a sum over fully-multiple mappings. $|M_k|$, $E_{j.k.n}$, and $v(t)$ are easily computed by their formulas. The challenge now is to count how many fully-multiple mappings are possible for each possible image configuration.

SAMPLE CALCULATIONS

Let us evaluate the first few C_k for n=5 as sample calculations for the summation.

C_0:

$$C_0 = 1 \text{ for all n}$$

C_1:

$$C_1 = |M_1| = \left(\binom{5}{2}\right)\binom{5-2}{2}^1 = 10\times 3 = 30$$

C_2:

$C_2 = |M_2| - E_{2.2.5}\sum_{t\in T_2} v(t)$. Using the formula, $|M_2| = \left(\binom{5}{2}\right)\binom{5-2}{2}^2 = 45\times 9 = 405$. $E_{2.2.5}$ = 1 by its formula. T_2 is the set of fully-multiple mappings of size 2, which is only possible if the mapping has a single image element of multiplicity 2. There are $\binom{5}{2}$=10 choices for the image element, and $\left(\binom{5-2}{2}\right)$ = 3 ways to assign two disjoint domain elements to each, so there are 30 items in this set, and each has $v(t) = -(-1)^{2-1}(2-1)=1$. The result is
$C_2 = 405-30 = 375$

C_3:

$C_3 = |M_3| - \sum_{j=2}^{3} E_{j.3.5}\sum_{t\in T_j} v(t)$. Using the formula, $|M_3| = \left(\binom{5}{2}\right)\binom{5-2}{2}^3 = 120\times 27 = 3240$. For fully-multiple mappings of size 2, $E_{2.3.5} = \left(\binom{5}{2}-2\right)\binom{5-2}{2}^{(3-2)} = 8\times 3 = 24$, and while computing C_2 it was shown that $\sum_{t\in T_2} v(t) = 30$, and therefore $E_{2.3.5}\sum_{t\in T_2} v(t) = 720$. Now for j=3, consider fully-multiple mappings of size 3, which are only possible with a single element in the image of the mapping with multiplicity 3. With n=5, there are only 3 elements disjoint with any range element, so there is just one mapping of size three for each of the 10 image elements. For these fully-multiple mappings of size 3, $v(t) = -(-1)^{3-1}(3-1)=-2$. Finally $E_{3.3.5}$ = 1 as it always is when j=k. Putting it all together yields

$$C_3 = |M_3| - \sum_{j=2}^{3} E_{j.3.5}\sum_{t\in T_j} v(t) = 3240-(720-20) = 2540$$

C_4:

$C_4 = |M_4| - \sum_{j=2}^{4} E_{j.4.5}\sum_{t\in T_j} v(t)$. Using the formula, $|M_4| = \left(\binom{5}{2}\right)\binom{5-2}{2}^4 = 210\times 81 = 17010$.

For fully-multiple mappings of size 2, $E_{2.4.5}=\left(\binom{5}{2}-2\right)\binom{5-2}{2}^{(4-2)}=28\times 9=252$, and while computing C_2 it was shown that $\sum_{t\in T_2} v(t) = 30$, and therefore $E_{2.4.5}\sum_{t\in T_2} v(t) = 7560$. For fully-multiple mappings of size 3, $E_{3.4.5}=\left(\binom{5}{2}-3\right)\binom{5-2}{2}^{(4-3)}=7\times 3=21$, and while computing C_3 it was shown that $\sum_{t\in T_3} v(t) = -20$, and therefore $E_{3.4.5}\sum_{t\in T_3} v(t) = -420$. For fully-multiple mappings of size 4, there are no elements of multiplicity 4 for n=5. However, fully-multiple mappings of size 4 are possible with two elements in the image, each of multiplicity 2. If the elements in the image are disjoint, e.g. {AB,CD}, then the three elements which may map to AB are distinct from the three elements which may map to CD, and choosing two of these three for each yields 3x3 or 9 mappings for each disjoint pair. If, on the other hand, the elements in the image are not disjoint, e.g. {AB, AC}, then there is an element, in this case DE, which can map to either but not both. If the element DE is not used, there is one mapping. If the element DE is used for AB, there are two mappings, and if the element DE is used for AC, another two mappings, for a total of 5. Of the $\binom{10}{2}=45$ possible images for the mapping, $\frac{1}{2}\binom{5}{2}\binom{3}{2}=15$ are disjoint (by the partitioning formula) and $5\times\binom{4}{2}=30$ are not disjoint (five choices for the shared element times two of the remaining four for the distinct element). This yields a total of $15\times 9+30\times 5=285$ mappings with two elements in the image, each of multiplicity 2, which are the only such mappings for j=4. Now
$v(t) = -\prod_{y\in image(t)}(-1)^{mult(y)-1}(mult(y)-1)$ yields
$v(t) = -(-1)^{2-1}(2-1)\times(-1)^{2-1}(2-1)=-1$. Finally $E_{4.4.5} = 1$ as it always is when j=k. Putting it all together yields

$$C_4=|M_4|-\sum_{j=2}^{4} E_{j.4.5}\sum_{t\in T_j} v(t) = 17010-(7560-420-285) = 10155$$

C_5:

$C_5=|M_5|-\sum_{j=2}^{5} E_{j.5.5}\sum_{t\in T_j} v(t)$. Using the formula, $|M_5| = \left(\binom{5}{2}\right)\binom{5-2}{2}^{5} = 252\times 243=61236$.

For fully-multiple mappings of size 2, $E_{2.5.5}=\left(\binom{5}{2}-2\right)\binom{5-2}{2}^{(5-2)}=56\times 27=1512$, and while computing C_2 it was shown that $\sum_{t\in T_2} v(t) = 30$, and therefore $E_{2.5.5}\sum_{t\in T_2} v(t) = 45360$. For fully-multiple mappings of size 3, $E_{3.5.5}=\left(\binom{5}{2}-3\right)\binom{5-2}{2}^{(5-3)}=21\times 9=189$, and while computing C_3 it was shown that $\sum_{t\in T_3} v(t) = -20$, and therefore $E_{3.5.5}\sum_{t\in T_3} v(t) = -3780$. For

fully-multiple mappings of size 4, $E_{4.5.5}=\left(\binom{5}{2}-4\right)\binom{5-2}{2}^{(5-4)}=6\times 3=18$, and while computing C_4 it was shown that $\sum_{t\in T_4} v(t) = -285$, and therefore $E_{4.5.5}\sum_{t\in T_4} v(t) = -5130$. For fully-multiple mappings of size 5, there must be two elements in the image, one with multiplicity 2 and the other with multiplicity 3. If the elements in the image are disjoint, e.g. {AB,CD}, then the three elements which may map to AB are distinct from the three elements which may map to CD, and there are two ways to choose which domain element will have multiplicity 2 times three ways to assign two of three domain elements, for a total of 6. If, on the other hand, the elements in the image are not disjoint, e.g. {AB, AC}, then there is an element, in this case DE, which can map to either but not both. That element must have multiplicity three and there are two elements left which must both map to the multiplicity two element, thus two mappings in this case. Since, as has been shown, there are 15 disjoint pairs and 30 non-disjoint pairs, there are thus $15\times 6+30\times 2=150$ fully-multiple mappings of size 5. Now $v(t) = -\prod_{y\in image(t)} (-1)^{mult(y)-1}(mult(y)-1)$ yields $v(t) = -(-1)^{2-1}(2-1)\times(-1)^{3-1}(3-1)=2$. Finally $E_{5.5.5} = 1$ as it always is when j=k. Putting it all together yields

$$C_5=|M_5|-\sum_{j=2}^{5} E_{j.5.5}\sum_{t\in T_j} v(t) = 61236-(45360-3780-5130+300) = 24486$$

C_6:

Because fully-multiple mappings of size 6 can have up to 3 elements in the image, and there are an increasing number of cases which need to be analyzed for such mappings, this is a good place to stop and let the computer take over. For the full list of C_k for n=5, refer to table 7-5a.

USE OF SYMMETRY AND ORBITS TO REDUCE COMPUTATIONS

As seen in the sample calculation, manually finding fully-multiple mappings becomes tedious once k becomes larger than about 5. If there is only one element in the image of the mapping, then finding the number of fully-multiple mappings is simple. There are $\binom{n-2}{2}$ elements disjoint from it that could be mapped to it, so if the multiplicity of that element is k, then there are $\left(\binom{n-2}{2}\right)$ ways to choose domain elements for it, and this also puts an upper limit of $\binom{n-2}{2}$ on the multiplicity of any image element.

When there are multiple elements in the image of the fully-multiple mapping, the situation becomes more complicated. Again consider enumerating all fully-multiple mappings with two elements in the image, each of multiplicity two. Recall that image elements are actually two-element subsets of X, and the results differ based on whether the image elements are disjoint from one another. For n=5, the smallest n for which fully-multiple mappings are possible, it was shown in the sample calculation that there are 9 such fully-multiple mappings if the image elements are disjoint, and 5 such mappings if not.

What if there are more than two image elements? To generalize this beyond two image elements, consider the set of all subsets of X, each of which subset might be the image of a fully-multiple mapping. Let the group of permutations on {1...n} act on the set of all subsets of X. This yields a set of

orbits for which the number of fully-multiple mappings with specified multiplicity values is identical. For example, with two elements in the image, {AB, AC} is a representative of one of the orbits (non-disjoint) and {AB, BC} is a representative of the other orbit (disjoint). For subsets of size 3, there are already 5 orbits {AB,AC,AD}, {AB,AC,BC}, {AB,AC,BD}, {AB,AC,DE}, {AB,CD,EF} which do not fit simply into named categories such as "disjoint." These orbits are easy for a computer to generate, although a human might be hard-pressed to patiently evaluate these and their sizes without error.

Table 3: Let $|X|=n$, and let Y be the set of all two-element subsets of X such that $|Y|=k$. Let the group of permutations on X act on Y. How many orbits are formed? (Computer calculated)

n\k	1	2	3	4	5	6	7	8	9	10	11	12	13	14
5	1	2	4	6	6									
6	1	2	5	9	15	21	24							
7	1	2	5	10	21	41	65	97	131	148				
8	1	2	5	11	24	56	115	221	402	663	980	1312	1557	1646

REMARK: The numbers in this table are also the number of graphs (up to an isomorphism) with n nodes and k edges, as given in the OEIS sequence A008406 [3]. This is to be expected because there is a one-to-one correspondence between these orbits and such graphs simply by letting the n elements of X correspond to the n nodes of the graph in any desired way. Since the elements of Y are distinct two-element subsets of X, each element of Y corresponds to a distinct pair of nodes of the graph, and therefore a distinct edge, so a k-element subset of Y corresponds to k distinct edges on the graph.

To enumerate the complete set of fully-multiple maps to be analyzed, it suffices to take a representative from each orbit. The representative contains a set of values to use as the image of a fully-multiple mapping. Enumerate all ways to assign multiplicity values mult(y) to each each y in the image such that $2 \leqslant mult(y) \leqslant \binom{n-2}{2}$ (because that is the number of x disjoint with y eligible to match with it), and $\sum mult(y) \leqslant \binom{n}{2}$ (because the sum of multiplicities over the image is the size of the domain of the mapping, which cannot exceed the size of $|X|$. Each of these sets of members of X for the image and corresponding multiplicity values for each will be referred to as an *image configuration*. Since the image configurations will be solved by computer, it is convenient to standardize a representation. The representation used in this analysis consists of a two-letter domain element followed by a two-digit multiplicitiy value, repeated as necessary for each domain element, in lexicographic order. For example, the image configurations discussed above for two elements in the image, each of multiplicity two, are:

```
AB02CD02
AB02AC02
```

which represent the disjoint and non-disjoint image configurations respectively.

To determine the number of image configurations needing to be solved (without additional searches for orbits once the multiplicities are assigned to the image), orbit sizes can be multiplied by multiplicity combinations as follows:

Let $y_1, y_2, \ldots, y_k$ be elements in the image of a fully-multiple mapping. Let Y(k) be the number of ways

to assign multiplicity values satisfying the constraints described above, namely that each individual multiplicity ranges from 2 through $\binom{n-2}{2}$ and the sum of all k multiplicities is not greater than $\binom{n}{2}$.

The Y(k), orbit sizes and total image configurations are calculated by computer. The total image configurations using this method for n=5 through 8 respectively are 114, 8721, 696840, and 127118173. These numbers are the sum of products of Y(k) and orbits as detailed in the tables below.

To calculate an even smaller set of orbits, it is possible to let the group of permutations on X act not only upon the set of images, but instead on the set of image configurations, which are pairs of image elements and multiplicities. This avoids, for example, six separate calculations for AB02AC03BC04, AB02AC04BC03, AB03AC02BC04, AB03AC04BC02, AB04AC02BC03, and AB04AC03BC02, which are all clearly equivalent under permutations of A,B and C, but would require six different computations under the original scheme. Orbits via group action on image configurations were more difficult to compute, but they reduced the number of image configurations (and consequently, total computing time) by roughly half once implemented. The last rows in each table 4a through 4d contain these values, and illustrate the improvement from including multiplicities in the group action.

Table 4a: Computation of number of image configurations to be evaluated for n=5. The total number of image configurations with and without including multiplicities are 69 and 114 respectively.

Number of image elements, k	1	2	3	4	5	Row Total
Y(k)	2	4	8	11	1	26
Number of image orbits	1	2	4	6	6	19
Y(k) times number of orbits	2	8	32	66	6	114
Number of image configuration orbits	2	6	20	35	6	69

Table 4b: Computation of number of image configurations to be evaluated for n=6. The total number of image configurations with and without including multiplicities are 3650 and 8721 respectively.

Number of image elements, k	1	2	3	4	5	6	7	Row total
Y(k)	5	25	115	270	247	84	8	754
Number of image orbits	1	2	5	9	15	21	24	77
Y(k) times number of image orbits	5	50	575	2430	3705	1764	192	8721
Number of image configuration orbits	5	30	229	918	1514	834	120	3650

Table 4c: Computation of number of image configurations to be evaluated for n=7. The total number of image configurations with and without including multiplicities are 295,742 and 696,840, respectively.

Number of image elements, k	1	2	3	4	5	6	7	8	9	10	Row total
Y(k)	9	81	564	2100	4263	4999	3432	1287	220	11	16966
Number of image orbits	1	2	5	10	21	41	65	97	131	148	521
Y(k) times number of image orbits	9	162	2820	21000	89523	204959	223080	124839	28820	1628	696840
Number of image configuration orbits	9	90	992	7109	30727	76057	100264	63312	16052	1130	295742

Table 4d: Computation of number of image configurations to be evaluated for n=8. The total number of image configurations with and without including multiplicities in the orbits are 62,411,967 and 127,118,173 respectively.

Number of image elements, k	1	2	3	4	5	6	7	8	9	10	11	12	13	14	Row total
Y(k)	14	193	1805	9786	33019	74445	116273	125970	92378	43758	12376	1820	105	1	511943
Number of image orbits	1	2	5	11	24	56	115	221	402	663	980	1312	1557	1646	6995
Y(k) times number of image orbits	14	386	9025	107646	792456	4168920	13371395	27839370	37135956	29011554	12128480	2387840	163485	1646	127118173
Number of image configuration orbits	14	206	3013	32144	243684	1324616	4980647	12252576	18504464	16052259	7346692	1552365	117641	1646	62411967

COMPUTER ALGORITHMS TO EVALUATE IMAGE CONFIGURATIONS

All computer programs were written in c++ and compiled with gcc version 4.8.4 on Ubuntu Linux 14.04 LTS. The Boost version 1.54 multiprecision package was used to perform arithmetic on integers larger than 64-bits. The hardware was a HP G60-230US laptop computer (circa 2009) with 4GB RAM, and a Pentium Dual-Core CPU T4200 @ 2.00GHz with 1MB Level-2 Cache. None of the algorithms was implemented using any parallel processing or multi-threading, so only one of the processor cores was being used to perform calculations. Repeated runs of the same program often ran with approximately 5 percent variations in timing.

For every y in X, and every possible mult(y) value, there are $\left(\binom{n-2}{2} \atop mult(y) \right)$ ways to assign domain elements. If each of the elements of X is assigned a bit to be used in a bitmap, the computer can accumulate bits in an integer denoting which possible domain elements have been used, and this bitmap can be tested against additional image elements and their required multiplicity values to examine whether the new domain elements are free via bitwise AND operation and if so, then set those bits via arithmetic OR operation to indicate they have been used.

For example, with n=5, let us illustrate the simple image configurations previously discussed, namely disjoint and non-disjoint with multiplicity of two for each. Bitmaps for all possible two-element domain assignments for AB, AC, and CD will be needed, for example:

Table 5: Bitmap showing all possible two-element domain element selections for the given image element, e.g. AB02 can be assigned to {CD,CE}, {CD,DE}, or {CE,DE}

	AB	AC	AD	AE	BC	BD	BE	CD	CE	DE
AB02								1	1	
								1		1
									1	1
AC02						1	1			
						1				1
							1			1
CD02	1			1						
	1						1			
				1			1			

Each row is a bitset with 10 bits. Typical processors have at least 32 bits in an unsigned integer, which can accommodate up to n=8 (requiring 28 bits). The three rows for each image value are put into an array, and a data structure is defined containing the array and two other values for its size and the current bitset under consideration.

Most programmers would start with a straightforward algorithm to test each bitset one-at-a-time against the other bitsets in order to gauge the performance. The straightforward algorithm is as follows:

Figure 1: “Straightforward” C-language implementation of an algorithm to count the number of mappings possible for a given image configuration.

```
struct image_element_t {
   unsigned current_domain_bitmap;
   unsigned total_domain_bitmaps;
   unsigned *domain_bitmaps;
};

/* evaluate_image_configuration
   input:
      first_image_element: beginning entry of array of image elements
      last_image_element: final entry of array of image elements
   output:
      the number of possible mappings where each specified image element
      has the specified preimage size, (i.e. specified number of domain
      elements map to it).

*/
uint64_t evaluate_image_configuration(
         image_element_t *first_image_element,
         image_element_t *last_image_element)
{
   uint64_t num_mappings = 0;
   image_element_t *image_element = first_image_element;
   unsigned bit_accumulator = 0;

   /* loop until all combinations are tested */
   for (;;) {
      /* test the next domain bitmap to see if its bits are completely free */
      if ((image_element->domain_bitmaps[image_element->current_domain_bitmap] & bit_accumulator) == 0) {
         /* bits are free. If this is not the final image value,
            then set the bits and go on to the next image value */
         if (image_element != last_image_element) {
            bit_accumulator |= image_element->domain_bitmaps[image_element->current_domain_bitmap];
            ++image_element;
            image_element->current_domain_bitmap = 0;
            continue;
         }
         /* this was the final image value, so increment the number of mappings found */
         ++num_mappings;
      }
      /* go on to the next bitmap for the current domain element. If this was the final bitmap
         for the current domain element, back up to the previous and increment that instead */
      while (++image_element->current_domain_bitmap == image_element->total_domain_bitmaps) {
         if (image_element-- == first_image_element) {
            /* if the image_element has backed up to to the first image element, it means
               that all domain elements have been tested against each other and the search is complete.
               Return the number of mappings found */
            return num_mappings;
         }
         /* when backing up to a new image element, it is necessary to reset the bits that were set
            for the old image element */
         bit_accumulator &= ~image_element->domain_bitmaps[image_element->current_domain_bitmap];
      }
   }
}
```

The straightforward algorithm worked well for n<7, however with n=7, it began to experience long delays computing various image configurations as shown in Table 6.

Table 6: Performance of straightforward algorithm with a selection of n=7 image configurations.

Image Configuration	Number of mappings computed	Time to compute, seconds
AB02AC02AD02AE02AF02AG02BC02	359501364	7
AB02AC02AD02AE02AF02AG02BC02BD02	2134746351	83
AB02AC02AD02AE02AF02AG02BC02BD02BE02	4743469131	550
AB02AC02AD02AE02AF02AG02BC02BD02BE02BF02	1591247304	1380

With many thousands of image configurations to evaluate, it is clear that this algorithm will not complete in a reasonable time. Further optimization is necessary.

To further optimize, notice that for n=7, there are 21 domain elements, resulting in $2^{21} = 2097152$ possible bitmaps, while some of the image configurations are seen to have mappings in the billions. Moreover, the number of bitmaps of a specific size are limited by the binomial coefficients, which take on a maximum value of $\binom{21}{10} = 352716$ for a 10- (or 11-) bit bitmap. Clearly there are many more pigeons (mappings) than holes (domains of mappings), therefore it should be possible to consolidate those mappings as a bitmap and a value for further processing. The following code implements this idea concisely using a *std::map* data structure from the standard c++ programming library. A *std::map* dynamically manages a set of keys and their values while presenting an array-like interface to the programmer. The keys will be bitmaps representing domain elements which have been used, and the values will be the number of mappings found whose domain elements are represented by that bitmap.

Figure 2: “Pigeonhole optimization” C++ -language implementation of an algorithm to count the number of mappings possible for a given image configuration.

```
/* evaluate_image_configuration
   input:
      first_image_element: beginning entry of array of image elements
      last_image_element: final entry of array of image elements
   output:
      the number of possible mappings where each specified image element
      has the specified preimage size, (i.e. specified number of domain
      elements map to it).

*/
uint64_t evaluate_image_configuration(
         image_element_t *first_image_element,
         image_element_t *last_image_element)
{
   typedef std::map<unsigned, uint64_t> bmap_index_t;
   bmap_index_t b1, b2; /* two maps to alternate as source / destination map */
   bool b_source_dest = true;
   b1[0] = 1; /* initialize the first map to a single bitmap with no bits set */

   image_element_t *image_element = first_image_element; /* to iterate from first to last image element */
   for(;;) {
      /* alternate the source and destination bitmaps once each loop */
      bmap_index_t &source = b_source_dest ? b1 : b2;
      b_source_dest = !b_source_dest;
      bmap_index_t &dest = b_source_dest ? b1 : b2;

      /* clear the destination, and keep a total for this iteration */
      dest.clear();
      uint64_t total = 0;

      /* iterate through the source bitmaps */
      for(std::map<unsigned, uint64_t>::const_iterator i = source.begin(); i != source.end(); ++i) {
         /* iterate through the possible domain bitmaps for the current image element */
         for(unsigned j = 0; j< image_element->total_domain_bitmaps; ++j) {
            unsigned bmap = image_element->domain_bitmaps[j];
            if ((i->first & bmap) == 0) /* if the bitmaps have no bits in common... */
            {
               /* then they can be combined to be tested against the next image element later */
               dest[i->first | bmap] += i->second;
               total += i->second;
            }
         }
      }
      /* return the total when the last image element has been processed */
      if (image_element++ == last_image_element) return total;
   }
}
```

This algorithm was implemented and was producing nearly instant results for n<7. For n=7, the 696840 image configurations were being processed at an average rate of about 8 per second, indicating that the program would finish in about 24 hours. Although this was satisfactory, one further optimization was made once it was observed that many of the image configuration calculations differ only in the final stages. For example, since both

`AB02AC02AD02AE02AF02AG02BC02`

and `AB02AC02AD02AE02AF02AG02BC03`

need to be evaluated, it is inefficient to start each calculation at stage 1, since the initial six of seven stages involve identical calculations. To capitalize upon this, the set of image configurations to be evaluated was arranged in lexicographic order. The result of the std::map was saved in a stack for each intermediate stage. When advancing from one image configuration to the next, the stack is popped one stage at a time (each stage representing the four characters specifying the image element and multiplicity), and ceases when the remaining stages match those of the next image configuration to be evaluated, This optimization took only a few minutes to implement, however it reduced the time to compute n=7 from approximately 24 hours to 1.5 hours.

ADDITIONAL OPTIMIZATIONS TO COMPUTE N=8

Computing techniques described thus far were successful in computing d_7, but were too slow and memory intensive to compute d_8. In order to compute d_8, the following additional optimizations were implemented:

- Arrange all of the image configurations in a directed graph, so that each image configuration is at most one domain element away from a previous image configuration. For instance, AB03 and AB04 are one step away since AB04 can be computed from the *bmap_index_t* results of AB03 by adding one more domain element. It turns out that only about 1% of the image configurations were more than one step away from another. This might have been remedied by searching for a better set of representatives from each orbit, but 1% was deemed an acceptable inefficiency. Once the image configurations were properly arranged in a directed graph, and "stepping stones" added for image configurations which were more than one domain element away from another, the graph could be traversed, computing all of the needed image configurations in the process.
- Eliminate the std::map pointer and search overhead by redefining the *bmap_index_t* to use a flat memory scheme. It was noticed that for image configurations with a large domain (of size *k*, for instance), many of the std::map structures were full, meaning that they were holding all possible *k*-bit bitmaps and associated 64-bit values. When this situation occurs, it is often preferable to store the values consecutively in memory, if memory locations for each index can be computed. Routines were written to compute the rank of a *k*-bit bitmap, as well as the *k*-bit bitmap associated with a given rank, so that the value associated with that bitmap could be stored at a definite, easily computed memory location. Since each 64-bit value takes up 8 bytes, and all possible bitmaps need to be stored, this brought the memory requirement for n=8 down to $2^{\binom{8}{2}} \times 8\, bytes$, which is 2GiB. The computer being used had 4GiB of memory so it was able to perform the calculation without using virtual memory.

Computation and verification of n=8 each required 3-4 weeks of runtime.

Working source code for anyone wishing to examine the techniques or verify the results is available online at https://github.com/RonNiles/Computation-of-A244493

VERIFYING THE RESULTS USING A COMPLEMENTARY PROBLEM

The number and complexity of techniques implemented in order to compute d_n raises the possibility that errors in reasoning or implementation might produce erroneous results. Unfortunately, there is no known test that can be applied directly to the computed value of d_n to confirm its correctness. In situations such as these, one often looks for an alternate method to compute the result, and if both methods produce an identical result, although not a mathematical proof of correctness, the common result can be considered a *consistency check* which lends credence to the correctness of the reasoning and implementation.

The method used herein to verify the results is to modify the computer program slightly so that it solves the problem *complementary* to d_n. This complementary problem is so-called because it is can be performed on the graph complementary to J(n,2). This complementary problem is to find the number of permutations on X such that p(x) is disjoint from x for all X in x, and will be denoted $\overline{d}_n$.

The motivation for solving the complementary problem lies in the observation that since C_k is the number of partial permutations of size k for which m(x) is disjoint from x for all x, that when $k=|X|$ the partial permutations are full permutations satisfying the condition of the complementary problem. Therefore $C_{|X|}=\overline{d}_n$. Moreover, if the program is modified slightly to search for $\overline{d}_n$ using identical reasoning, and the corresponding constants are called $\overline{C}_k$, then $\overline{C}_{|X|}=d_n$. When one considers that the logic behind the calculations to be performed to evaluate both the original problem and its complement is the same, yet the quantity and size of image configurations and their evaluation are vastly different, the fact that they produce results where both $C_{|X|}=\overline{d}_n$ and $\overline{C}_{|X|}=d_n$ as seen in Tables 7-5 through 7-7 serves as a evidence in support of the correctness of the logic and implementation.

Tables 7-5a and 7-5b: Computation of d_5 and its complement $\overline{d}_5$. Note that the consistency check is valid, i.e. $\overline{d}_5=C_{10}$, and $d_5=\overline{C}_{10}$

Results for d_n, n=5

k\	$(-1)^k C_k$	*	(10-k)!	Product
0	1	*	3628800	3628800
1	-30	*	362880	-10886400
2	+375	*	40320	15120000
3	-2540	*	5040	-12801600
4	+10155	*	720	7311600
5	-24486	*	120	-2938320
6	+34945	*	24	838680
7	-27840	*	6	-167040
8	+11040	*	2	22080
9	-1720	*	1	-1720
10	+60	*	1	60
d_5=				126140

Results for $\overline{d}_n$, n=5

k\	$(-1)^k \overline{C}_k$	*	(10-k)!	Product
0	1	*	3628800	3628800
1	-70	*	362880	-25401600
2	+1995	*	40320	80438400
3	-30100	*	5040	-151704000
4	+262015	*	720	188650800
5	-1346574	*	120	-161588880
6	+4021945	*	24	96526680
7	-6616480	*	6	-39698880
8	+5377920	*	2	10755840
9	-1733240	*	1	-1733240
10	+126140	*	1	126140
$\overline{d}_5$=				60

Tables 7-6a and 7-6b: Computation of d_6 and its complement $\overline{d}_6$. Note that the consistency check is valid, i.e. $\overline{d}_6 = C_{15}$, and $d_6 = \overline{C}_{15}$

Results for d_n, n=6

k\	$(-1)^k C_k$	*	(15-k)!	Product
0	1	*	1307674368000	1307674368000
1	-90	*	87178291200	-7846046208000
2	+3555	*	6227020800	22137058944000
3	-81330	*	479001600	-38957200128000
4	+1197945	*	39916800	47818130976000
5	-11949138	*	3628800	-43361031974400
6	+82686195	*	362880	30005166441600
7	-400122810	*	40320	-16132951699200
8	+1348166475	*	5040	6794759034000
9	-3113171470	*	720	-2241483458400
10	+4786438257	*	120	574372590840
11	-4677587190	*	24	-112262092560
12	+2699885595	*	6	16199313570
13	-814646790	*	2	-1629293580
14	+102166425	*	1	102166425
15	-3013854	*	1	-3013854
d_6=				855966441

Results for $\overline{d}_n$, n=6

k\	$(-1)^k \overline{C}_k$	*	(15-k)!	Product
0	1	*	1307674368000	1307674368000
1	-135	*	87178291200	-11769069312000
2	+7965	*	6227020800	49598220672000
3	-271035	*	479001600	-129826198656000
4	+5913225	*	39916800	236037019680000
5	-87002847	*	3628800	-315715931193600
6	+884399445	*	362880	320930870601600
7	-6261094035	*	40320	-252447311491200
8	+30738321675	*	5040	154921141242000
9	-103009220045	*	720	-74166638432400
10	+228929330367	*	120	27471519644040
11	-322129793385	*	24	-7731115041240
12	+266680246155	*	6	1600081476930
13	-114955943085	*	2	-229911886170
14	+20507308335	*	1	20507308335
15	-855966441	*	1	-855966441
$\overline{d}_6$=				3013854

Tables 7-7a and 7-7b: Computation of d_7 and its complement $\bar{d}_7$. Note that the consistency check is valid, i.e. $\bar{d}_7 = C_{21}$, and $d_7 = \bar{C}_{21}$

Results for d_n, n=7

k\	$(-1)^k C_k$	*	(21-k)!	Product
0	1	*	51090942171709440000	51090942171709440000
1	-210	*	2432902008176640000	-510909421717094400000
2	+20055	*	121645100408832000	2439592488699125760000
3	-1155490	*	6402373705728000	-7397878793231646720000
4	+44935065	*	355687428096000	15982837701176586240000
5	-1250308458	*	20922789888000	-26159941161923272704000
6	+25749785775	*	1307674368000	33672334839458515200000
7	-400575453210	*	87178291200	-34921483507513354752000
8	+4762451163015	*	6227020800	29655882451078595712000
9	-43511565506990	*	479001600	-20842109496353021184000
10	+305663455504617	*	39916800	12201107020686695865600
11	-1644582767904510	*	3628800	-596786194817188588800 0
12	+6721007731524275	*	362880	2438919285615528912000
13	-20590820917882110	*	40320	-830221899409006675200
14	+46404302391494325	*	5040	233877684053131398000
15	-74925220247495310	*	720	-53946158578196623200
16	+83541999718579500	*	120	10025039966229540000
17	-61040258771327640	*	24	-1464966210511863360
18	+27027549293371900	*	6	162165295760231400
19	-6393764833511160	*	2	-12787529667022320
20	+640267331029536	*	1	640267331029536
21	-15304715502400	*	1	-15304715502400
d_7=				102526835994056

Results for $\overline{d}_n$, n=7

k\	$(-1)^k \overline{C}_k$	*	(21-k)!	Product
0	1	*	51090942171709440000	51090942171709440000
1	-231	*	2432902008176640000	-562000363888803840000
2	+24255	*	121645100408832000	2950501910416220160000
3	-1535765	*	6402373705728000	-9832541454177361920000
4	+65602215	*	355687428096000	23333883130750832640000
5	-2004101757	*	20922789888000	-41931399975882633216000
6	+45294002607	*	1307674368000	59229806233299077376000
7	-772877706165	*	87178291200	-67378157730040405248000
8	+10074208135005	*	6227020800	62732303600205343104000
9	-100863289737175	*	479001600	-48313677165370404480000
10	+776092531015977	*	39916800	30979130342058550713600
11	-4571515733275827	*	3628800	-16589116292911321017600
12	+20443995167704325	*	362880	7418716966456545456000
13	-68505263121009015	*	40320	-2762132209039083484800
14	+168779355309831285	*	5040	850647950761549676400
15	-297777625642156695	*	720	-214399890462352820400
16	+362630881480828590	*	120	43515705777699430800
17	-289245414695515110	*	24	-6941889952692362640
18	+139747304707701460	*	6	838483828246208760
19	-36056171198898060	*	2	-72112342397796120
20	+3936173233476456	*	1	3936173233476456
21	-102526835994056	*	1	-102526835994056
$\overline{d}_7$=				15304715502400

Tables 7-8a and 7-8b: Computation of d_8 and its complement $\bar{d}_8$. Note that the consistency check is valid, i.e. $\bar{d}_8 = C_{28}$, and $d_8 = \bar{C}_{28}$

Results for d_n, n=8

k\	$(-1)^k C_k$ *	(28-k)!	Product
0	1 *	304888344611713860501504000000	304888344611713860501504000000
1	-420 *	10888869450418352160768000000	-457332516917570790752256000000 0
2	+82110 *	4032914611266056355840000000	331142618731055887378022400000000
3	-9935380 *	155112100433309859840000000	-1541097660403098111525713920000000 0
4	+834601635 *	62044840173323943936000000	51782725051969846993633935360000 0
5	-51750714456 *	25852016738884976640000000	-1337860336365768737924870307840000 0
6	+2458746262140 *	1124000727777607680000000	2763632588065832552391357235200000 0
7	-91719326277240 *	51090942171709440000000	-4686026794858618912322165145600000 0
8	+2730957403569465 *	2432902008176640000000	6644151751389014081752230297600000 0
9	-65636054374906540 *	121645100408832000000	-7984304424875062931155790561280000 0
10	+1282813229398258194 *	6402373705728000000	8213049689259429265068320735232000 0
11	-20480119938680495100 *	355687428096000000	-7284521188086874529998930329600000 0
12	+267650609704894203795 *	20922789888000000	5599997470250595111271837224960000 0
13	-2863426007889409067280 *	1307674368000000	-3744428795181546015948843479040000 0
14	+25030137207374224744200 *	87178291200000	2182084590240424952124113111040000 0
15	-178095238585377254997968 *	6227020800000	-1109002755052106742719250693734400 0
16	+1025521344534266069048475 *	479001600000	4912263648660647018999300025600000 0
17	-4741168328063756755228860 *	39916800000	-1892522679176553656471193588480000 0
18	+1741513594658516499929101 0 *	3628800000	6319604532296824674942721708800 0
19	-5014222482801217787461038 0 *	362880000	-1819561054558905910713861469440 0
20	+1112220481481818197063432 0609 *	40320000	4484472981334697055975806954880
21	-1858600466636680181907254 00 *	5040000	-9367346351848868116812560160 00
22	+2272460962818147391482759 00 *	720000	1636171893229066121867586480 00
23	-1954838702070634666929342 00 *	120000	-2345806442484761600315210400 0
24	+1120424770975228424845650 35 *	24000	2689019450340548219629560840
25	-3950109927466576929336229 2 *	6000	-2370065956479946157601737 52
26	+7540995066568632323914350 *	2000	1508199013313726464782870 0
27	-6168495950462544236074 60 *	1000	-6168495950462544236074 60
28	+1217811244207670069578 5 *	1000	1217811244207670069578 5
d_8=			258081861682902430193

Results for $\overline{d}_n$, n=8

k\	$(-1)^k \overline{C}_k$	*	(28-k)!	Product
0	1	*	304888344611713860501504000000	304888344611713860501504000000
1	-364	*	10888869450418352160768000000	-3963548479952280186519552000000
2	+61698	*	403291461126605635584000000	24882276568589314504261632000000
3	-6475196	*	15511210043330985984000000	-100438125227736627119652864000000
4	+471966635	*	6204484017332394393600000	2928309443571651858440257536000000
5	-25402835304	*	25852016738884976640000	-656714523494146234185807298560000
6	+1048053722436	*	1124000727777607680000	1178013146768094834590821908480000
7	-33962926684872	*	51090942171709440000	-1735197923238802751244427591680000
8	+878829021327369	*	2432902008176640000	2138104890831267223784218460160000
9	-18363203266937060	*	121645100408832000	-2233793705234350475434412113920000
10	+312146065535226734	*	6402373705728000	1998475762329184728684274532352000
11	-4335988557175091796	*	355687428096000	-1542256618155294248071949500416000
12	+49324050030276395787	*	20922789888000	1031996735208673067617329401856000
13	-459500562961008770544	*	1307674368000	-600877108265681352663582216192000
14	+3499039410466433940600	*	87178291200	305040276645919105899190302720000
15	-21697050728260167099952	*	6227020800	-135107986183531208342876783001600
16	+1089267736391991344839 63	*	479001600	521760988560142081364334513408 00
17	-4392352825046376714458 76	*	39916800	-175328669246811210035707431168 00
18	+1407805797230803970109502	*	3628800	5108645676991141446733360857600
19	-3538405116512191886464420	*	362880	-1284016448679944191760208729600
20	+6854393247442700648512089	*	40320	2763691357368896901480074284 80
21	-1000752460170273854754 2056	*	5040	-5043792399258180227961196224 0
22	+1069523335318581704075 9812	*	720	7700568014293788269347064640
23	-8045466948531950619496584	*	120	-9654560338238340743395900 80
24	+4034275711744410085123835	*	24	9682261708186584204297204 0
25	-1244897646707282800681148	*	6	-7469385880243696804086888
26	+2081119393561733153437 62	*	2	4162238787123466306875 24
27	-1491410951939290999828 4	*	1	-1491410951939290999828 4
28	+258081861682902430193	*	1	258081861682902430193
$\overline{d}_8$=				12178112442076700695785

REFERENCES

[1] J. East, R.D. Gray, Diagram monoids and Graham–Houghton graphs: Idempotents and generating sets of ideals, J. Combin Theory Ser. A 146 (2017) 63–128

[2] The Online Encyclopedia of Integer Sequences, http://oeis.org, 2017, Sequence A244493.

[3] http://www.cut-the-knot.org/arithmetic/combinatorics/InclusionExclusion.shtml

[4] Weisstein, Eric W. "Derangement." From MathWorld--A Wolfram Web Resource. http://mathworld.wolfram.com/Derangement.html

[5] The Online Encyclopedia of Integer Sequences, http://oeis.org, 2017, Sequence A008406.