On Neighbor-Property Cycles

Stanislav Sykora, Extra Byte, www.ebyte.it

First published in May 2014

Cycles of various lengths composed of elements belonging to a set *S* can be used to define monocyclic permutations ("round-table seatings") of subsets of *S*. Those are interesting by themselves, but the whole context becomes still more interesting when the elements possess specific properties or when *pairs of elements* of *S* have a specific property (implying a partition of the product set *SxS*). In this case, for example, one can construct cycles in which every pair of adjacent elements (neighbors) has, or has not, a specific property. For cycle lengths greater than 2, this reduces to finding all distinct Hamiltonian cycles (an NP-complete problem) in a simple graph in which the vertices correspond to the elements of the cycle, and an edge between two vertices is present only if the pair possesses the property.

This essay uses many selected cases to illustrate some of the counting tasks and problems that arise in these situations, using cycles of integer numbers. The pair-properties P(i, j) considered here are: i, j differ by exactly one bit in their binary expansions (Gray property), or by some minimum amount, or by a power of 2 or 3, or by a prime, or they are coprime, or their difference and/or sum are primes, etc.

This study is an extension of the earlier one [1] on canonical Gray cycles (CGC), since the CGCs are a special case of NPC's. In both studies, a rather brute-force algorithm was used to enumerate/list all the NPC's with any given property, while a more efficient algorithm is being developed. The included C++ code is easily adaptable to tackle all problems of this type but, alas, it has severe execution time limits for large cycle lengths.

Keywords: math, sequence, integer, permutation, cycle, pair-property, NPC, Gray cycle, simple graph, Hamiltonian cycle, algorithm, enumeration, counting

Introduction

In a previous study [1,2], of which the present one is a generalization, we have analyzed the canonical Gray cycles (CGC) satisfying the following requirements:

1. An CGC of length n is a permutation $\{c_0, c_1, ..., c_{n-1}\}$ of the integers {0, 1, 2, ..., n-1}.
2. Binary expansions of any two neighbors of a CGC differ by exactly one bit (since the CGC is a cycle, the rule must apply also to the pair composed of the last and the first element).
3. Each CGC starts with $c_0 = 0$ and continues with an integer smaller than the last one (i.e. the successor c_1 of c_0, is smaller than its cyclic precursor c_{m-1}).

Rule (1) is a global condition that can be modified in various ways. For example, one can consider only integers from some initial offset value ω through ω+n-1, or restrict the cycle to only even (or only odd) integers, or to finite sets of primes, or odd primes, etc. In this essay, we will consider only n-tuples of consecutive integers starting at ω, where the most common values of the offset ω will be 0 and 1.

Rule (2) is a particular instance of what in this essay will become a more generic **pair-property** P, to be satisfied by every pair (c_k, c_{k+1}) for k = 0, 1, ..., n-1, with $c_n \equiv c_0$. There are many pair-properties that can replace the specific one that characterizes the CGC's.

Finally, rule (3) simply ensures that cycles that differ from each other just by a rotation are not counted as distinct cases. We will maintain this one without any modification.

In [1] we had found the counts of all CGC cycles of various lengths n and ascertained that, for example,

- there does not exist any CGC of length 1, because the pair (0,0) does not satisfy rule (2),
- there are no CGC's of odd length,
- the first length n for which there is more than one CGC is n = 8, etc.

Here we will analyze analogues of the CGC's with different choices of pair-properties P, and see what more we can learn. We will call these constructs **Neighbor Property Cycles**, or **NPC**'s. One can also think about them as round table seating's in which every person is assigned a distinct label s_k and the pair-property is a condition to be satisfied by the labels of persons sitting next to each other.

An NPC of length n on a subset $S \equiv \{s_0, s_1, ..., s_{n-1}\}$ of distinct elements of a set S is a mono-cyclic permutation $C \equiv \{c_0, c_1, ..., c_{n-1}\}$ of s, such that every pair of neighbors in the cycle satisfies a given property P. To achieve uniqueness, only "canonical" cycles will be admitted, which are those with $c_0 \equiv s_0$ and $c_1 < c_{n-1}$.

The above specifies what we will refer to as the **NPC category** NPC(S; P) and the number of NPC's of length n will be denoted[1] as NPC(n; S; P).

The goals

The next Section contains a few generic notes on NPC's. Beyond that, as anticipated, this essay focuses on the enumeration of NPC's based on sequences of integers of the type

$$S_{\omega,n} \equiv \{\omega, \omega+1, \omega+2, ..., \omega+n-1\},$$

with ω being an offset and n a length.

One of the goals of this study is to provide support for the development of algorithms for finding and enumerating Hamiltonian cycles [3, 4, 5] in simple graphs [6], which is a well-known and important NP-complete algorithmic problem [7, 8]. In the cases studied here, the NPC's were enumerated using a robust, foolproof, but rather inefficient brute-force algorithm based on a slightly optimized backtracking recursion. The presented examples can be useful in testing more sophisticated approaches to Hamiltonian cycles enumeration. For sure, using the NPC's is a very general, and very prolific, approach to generating infinite families of equivalent non-trivial simple graphs.

General features of NPC's

There are two somewhat special cases of NPC lengths:

For length $n = 1$, we have the degenerate cycle $\{s_0, s_0\}$, where s_0 is a single element of S. Whether such a primitive cycle is legitimate depends upon the definition of the currently scrutinized property for the pair (s_0, s_0). In any case, the value of NPC(1) is always either 0 or 1.

For length $n = 2$ we have a pair of elements s_0 and s_1 and we need to investigate the scrutinized property for the pairs (s_0, s_1) and (s_1, s_0). We will focus only on **symmetric pair-properties** P, such that if (s, s') satisfies P then so does (s', s). Under this assumption, the value of NPC(2) is also always either 0 or 1.

For lengths $n > 2$, every NPC can be identified with a Hamiltonian cycle in a simple graph, hence called the **pair-property graph** for the property P and the set $S \equiv \{s_0, s_1, ..., s_{n-1}\}$, defined as follows:

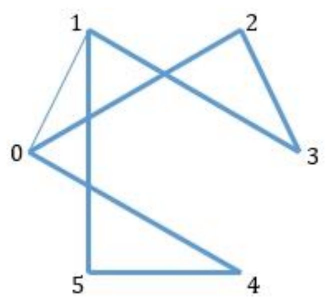
a. Every element of the set $S \equiv \{s_0, s_1, ..., s_{n-1}\}$ is identified with a vertex, and
b. Two vertices s_k and $s_{k'}$ are connected by an edge iff the pair $(s_k, s_{k'})$ satisfies the property P.

The diagram on the next page shows a 6-vertex Gray-property graph drawn for the set $S_{0,5}$, whose only Hamiltonian cycle corresponds to the unique CGC of length 6, namely $C = \{0,2,3,1,5,4\}$.

The Hamiltonian cycle is shown in bold. Notice, however, the edge (0, 1) which, even though it satisfies the Gray property, is superfluous (redundant) for the CGC "solution".

[1] When implicit in the context, the arguments S and P will be dropped. Likewise, since the cycle length n is implicitly determined by the cardinality of S, either n or the definition of S will be generally simplified by dropping redundant parts.

In general, when an edge in a simple graph is not traversed by *any* of its Hamiltonian cycles, we will say that it is **HC redundant**, or just redundant. More generally, when, in a graph, all Hamiltonian cycles are enumerated and found to be H, an edge e can be used in only in $h(e) \leq H$ of them. The number h(e) is a property of the particular edge, and can be called its Hamiltonian cycles participation, or **HC participation**. In a given graph with n vertices, there are edges with minimum HC participation (possibly 0), and those with a maximum HC participation[2].



The number of edges in any NPC cycle of length n equals n, while the total number of edges compatible with the property P can be much higher. On the other hand, it is evident that it may never be smaller than n. Hence:

- **Lemma 1:** Given n elements $S \equiv \{s_0, s_1, ..., s_{n-1}\}$ of $\mathcal{S}$, if the number of pairs $(s_k, s_{k'})$ which can be formed from the elements of S and which satisfy the pair-property P is smaller than n, then there is no corresponding NPC on S.

The representation of NPC's as Hamiltonian cycles in simple graphs shows that

- **Lemma 2:** For $n > 2$, any pair-property P, and any subset $S \equiv \{s_0, s_1, ..., s_{n-1}\}$, $NPC(n;s;P) \leq H(n)$, where H(n) is the number of Hamiltonian cycles in a complete simple graph with n vertices.

Using present terminology, H(n) corresponds to **void pair-property** P_0, i.e., one which is "satisfied" for any pair of elements. Intended in this sense, H(n) extends to $H(1) = H(2) = 1$ and the numbers H(n) form an integer sequence [9] whose starting elements are listed in Table I. For $n \geq 3$, we have[3]

(1) $H(n) = (n-1)!/2$, and, for any edge e, $h(e) \equiv h(n) = (n-2)! = H(n)/(2n-2)$.

H(n) defines an upper bound on the numbers NPC(n; S; P), regardless of the choice of the pair-property P and/or of the subset S.

A still another insight gained from the analogy with Hamiltonian cycles is the following:

- **Lemma 3:** Let P be a pair-property and $S \equiv \{s_0, s_1, ..., s_{n-1}\}$ a sequence with $n > 2$ elements. If, for some index K, P holds for fewer than two pairs (s_K, s_i), $i \neq K$, then $NPC(n;S;P) = 0$.

The reason is obvious: a simple graph containing a vertex with degree smaller than 2 does not admit any Hamiltonian cycle.

Equally obvious is the fact that, in any simple graph, when one removes one or more edges, the number of its Hamiltonian cycles cannot increase. In our context this means that if the property P is replaced by a more restrictive property P' (symbolically $P' \Rightarrow P$, but not vice versa) then NPC(n;S;P) cannot increase:

- **Lemma 4:** Let P and P' be two properties of pairs of elements of S such that the validity of P' implies that of P, or $P' \Rightarrow P$. Then, for any n and any $S \equiv \{s_0, s_1, ..., s_{n-1}\}$, $NPC(n;S;P') \leq NPC(n;S;P)$.

There are many NPC features, other than NPC(n; S; P), which can be of interest once one defines the property P and selects the elements of S. Here are a few suggestions, out of many possible:

[2] Possibly equal to that in a complete graph with the same number of vertices.
[3] The formula for H(m) is easily understood. Consider a complete graph G with m vertices and H(m) Hamiltonian circuits. Add another vertex, with an edge to every of the vertices of G, to form a complete graph G' with m+1 vertices. A Hamiltonian circuit in G' can be formed from any of the H(m) Hamiltonian circuits in G by "opening" one of its m edges are inserting therein the newly added vertex. Hence $H(m+1) = m.H(m)$ which, together with the obvious $H(3)=1$, gives the first equation in (1).
The formula for h(e) is easily obtained as follows: Each of the H(m) Hamiltonian circuits contains m edges. This amounts to m.H(m) total edge traversals in all the circuits. Since a complete graph is fully symmetric, h(e) must be the same for all of its m(m-1)/2 edges. Hence, each edge is traversed $h(e) = m.H(m)/[m(m-1)/2]$ times which evaluates to (m-2)!

- The smallest n for which NPC(n; S; P) > 0 (existence of at least one NPC).
- The smallest n for which NPC(n; S; P) > 1 (existence of multiple NPC's).
- Convergence of the ratios r(n) = NPC(n; S; P)/H(n), or R(n) = log(NPC(n; S; P))/log(H(n)).
- Existence of an n_0 such that NPC(n; S; P) = 0 for any $n \geq n_0$ or, vice versa, any $n < n_0$.
- Existence of edges with h(e) = h(n).
- Existence of redundant edges.

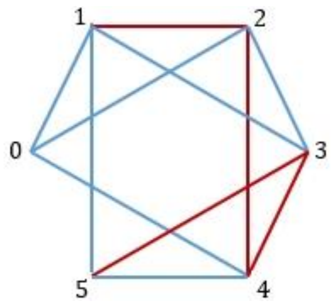
To specify both the NPC of a certain type and, for n > 2, the corresponding graph, we will use the same script, NPC(n; S; P), adopted for their counts. In specifying the property P, we will often use simplified descriptions like "Gray" or "coprimes" and, in some cases, employ generic symbols like Δ and Σ to denote the difference and the sum, respectively, of any pair of neighboring elements.

Properties P depending only on the difference Δ between neighbors

Example 1: Neighbors differing by a power of 2: NPC(n; S_ω; $\Delta = 2^k$, k≥0)

For the CGC's every pair of neighbors had to differ by exactly one bit in their binary expansions (the Gray property P'). There is a weaker property which includes the Gray's, namely that the difference between any two neighbors should be a power of 2 (property P). In fact, when the Gray condition is met (for example, the pair 1 and 3), the difference is always a power of 2, but not the other way round (consider, for example, the pair 3 and 5).

On the basis of Lemma 4 we therefore expect that, for any given cycle length n, the number of NPC's on the set $S_{0,n}$ for which the difference Δ between neighbors is a power of 2 exceeds the number of NPC' in which the neighbors satisfy the Gray condition. This is confirmed by the results of explicit enumerations listed in columns 3 and 4 of Table I. It holds for any cycle with even length and there exist NPC's with property P also for odd cycle lengths, for which there are no CGC's.



The illustration on the right shows the simple graph corresponding to the current property P on a set of six vertices {0, 1, 2, 3, 4, 5}. The blue edges are those matching only the Gray property P', while all the edges, blue and red, connect vertices differing by 1, 2, or 4. The graph admits eight Hamiltonian cycles corresponding to eight possible NPC's of length 6:

C_1 to C_4: {0, 1, 2, 3, 5, 4}, {0, 1, 3, 5, 4, 2}, {0, 1, 5, 3, 2, 4}, {0, 1, 5, 3, 4, 2},
C_5 to C_8: {0, 1, 5, 4, 3, 2}, {0, 2, 1, 3, 5, 4}, {0, 2, 1, 5, 3, 4}, {0, 2, 3, 1, 5, 4}.

Unlike in the CGC case discussed above, there is now no redundant edge, and the edge HC-participations h(e) range from 3 (edges 1-2, 1-3, 2-3, 2-4, 3-4) to 6 (edges 0-2, 3-5).

Two notes are appropriate at this point:

a) The ratio r(n) appears to converge rapidly to zero for this pair-property, but R(n) appear to be increasing and, being bounded from above by 1.0, it might converge to a constant.
b) When, like in this case, the property P depends only on differences between neighboring elements of the cycle, shifting the element values by a constant offset does not affect the problem. Consequently the values of all NPC(n; S_ω; P) are independent of the offset ω.

Example 2: Neighbors differing by a power of 3: NPC(n; S_ω; $\Delta = 3^k$, k≥0)

In comparison with the preceding case, requesting that all neighboring elements in an NPC differ exclusively by one of the numbers {1, 3, 9, 27, ...} we considerably restrict the average number of edges per vertex. Hence, even though there is no implication relationship between the two properties, we intuitively expect a drop in the number of compatible NPC's. This heuristic expectation is indeed confirmed by explicit enumeration (see column 5 of Table I).

Table I. Numbers of NPC's on $S_{\omega,n}$ for selected pair-properties P.

Notes: First row contains references to the *Online Encyclopedia of Integer Sequences*, with links. Second row: Cycle length is indicated by n. H(n) are the counts for the void pair-property P_0, given by Eq.1. CGC(n) indicates that neighbor pairs have the Gray property and the set is $S_{0,n}$. For columns with Δ in their header, the Δ refers to the difference between neighboring pairs, and the set $S_{0,n}$ can be replaced by the more generic $S_{\omega,n}$, with any offset ω.

OEIS:	A001710(n-1)	A236602	A242519	A242520	A242521
n	H(n): no condition	Gray	$\Delta = 2^k, k\geq 0$	$\Delta = 3^k, k\geq 0$	Δ = proper power
1	1	0	0	0	0
2	1	1	1	1	0
3	1	0	1	0	0
4	3	1	1	1	0
5	12	0	4	0	0
6	60	1	8	2	0
7	360	0	14	0	0
8	2520	6	32	3	0
9	20160	0	142	0	0
10	181440	4	426	27	0
11	1814400	0	1204	0	0
12	19958400	22	3747	165	0
13	239500800	0	9374	0	2
14	3113510400	96	26306	676	4
15	43589145600	0	77700	0	6
16	653837184000	1344	219877	3584	9
17	10461394944000	0	1169656	0	42
18	177843714048000	672	4736264	19108	231
19	3201186852864000	0	17360564	0	1052
20	60822550204416000	3448	69631372	80754	3818

It also turns out that there are no odd-length NPC's with the property $\Delta = 3^k$, a fact which is easy to understand: since all powers of 3 are odd numbers, the neighboring elements of the cycle may not be both even, nor both odd. Consequently, there must be a strict alternation of odd and even elements and therefore the total length of the cycle must be even.

These are some instances of NPC's satisfying the current pair-property:

1 cycle of length 4: $C_1 = \{0, 1, 2, 3\}$,
2 cycles of length 6: $C_1 = \{0, 1, 2, 5, 4, 3\}$, $C_2 = \{0, 1, 4, 5, 2, 3\}$,
3 cycles of length 8: $C_1 = \{0, 1, 2, 5, 4, 7, 6, 3\}$, $C_2 = \{0, 1, 2, 5, 6, 7, 4, 3\}$, $C_3 = \{0, 1, 4, 7, 6, 5, 2, 3\}$
etc.

The simplest cycle of length 10 is $C_1 = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$ which, incidentally, indicates a generic solution applicable to any cycle length of the form n = 3^k+1. However, there are 26 other solutions[4] for n=10, such as $C_{27} = \{0, 3, 6, 7, 4, 1, 2, 5, 8, 9\}$. The counts of even-length cycles increase quite rapidly, even though not as fast as in the case of $\Delta = 2^k$.

One could proceed and investigate the sequences of NPC's in which Δ = b^k, k ≥ 0, for b values of 4, 5, 6, etc. All such NPC's form non-trivial sequences. For odd b, they must have even lengths but otherwise, apart from that, it is not easy to derive additional rules.

[4] You can use the provided utility to list them all.

For curiosity, these are the only two NPC's of length 23 for base b = 10:

NPC(23; S_0; $\Delta = 10^k$, k≥0): {0, 1, 2, 3, 13, 14, 4, 5, 15, 16, 6, 7, 17, 18, 8, 9, 19, 20, 21, 22, 12, 11, 10},
{0, 1, 11, 21, 22, 12, 2, 3, 13, 14, 4, 5, 15, 16, 6, 7, 17, 18, 8, 9, 19, 20, 10}.

Example 3: Neighbors differing by a proper power: NPC(n; $S_{\omega,n}$; $\Delta = b^k$, b > 1, k > 1)

The reader might object that, since a property of the type Δ = b^k, with a k ≥ 0, and a given base b, includes the "small" numbers 1 and b, it is relatively easy to construct NPC's which satisfy it. This is no doubt true. In fact, when the property is changed to Δ = b^k, k > 1, i.e., to proper powers of a given base b, no NPC solutions seem to exist[5] for any cycle length and any b > 1.

An interesting case, however, is represented by the condition Δ = b^k, with *some* b > 1 and *some* k > 1. In other words, we will admit values of Δ matching any element of the set {4, 8, 9, 16, 25, 27, 32, ...}, coincident with OEIS A001597 (perfect powers), but excluding 1. The NPC counts for this property are shown in the last column of Table I. It turns out that, somewhat surprisingly, there are growing numbers of such solutions for all cycle lengths starting from 13 up.

For n = 13, the smallest compatible cycle length, there are two distinct solutions, namely
C_1 = {0, 4, 8, 12, 3, 7, 11, 2, 6, 10, 1, 5, 9} and
C_2 = {0, 8, 4, 12, 3, 7, 11, 2, 6, 10, 1, 5, 9},
with many redundant edges (for example 1-9, 2-10, 3-11, ...).

What emerges from these examples is that one can, in principle, define the difference-based property P simply by requiring Δ to belong to any set (or sequence) A of natural numbers. In this way, the sequence A de-facto defines the counts NPC(n; S_ω; Δ∈A). As a mapping between sequences, this is somewhat complicated, but perfectly legitimate.

Example 4: Minimum difference Δ : NPC(n; S_ω; Δ ≥ m)

Let us now investigate the NPC's in which every two neighbors differ by at least 2, thus excluding pairs such as (1, 2) or (9, 8) where the difference is only 1. Still more generally, we can fix the property by choosing a minimum difference m and requiring Δ ≥ m. Evidently, the choice m = 1 amounts to imposing no restrictive condition at all, and the resulting counts would be those of H(n). As m increases, the property becomes more and more restrictive, and the NPC counts are rapidly dropping.

This general behavior is illustrated by the actual values listed in columns 2, 3, and 4 of Table II for the choices m = 2, 3, and 4, respectively.

The smallest cycle solution for an NPC of this type is always a cycle of length n = 2m+1.
It is unique and has the form

C = {0, m, 2m, m-1, 2m-1, m-2, 2m-2, ..., 1, m+1}

For larger cycle lengths, the solutions become multiple. Already for n = 2m+2 we have, for m = 1, 2, 3,...,, the following NPC(2m+2; S_ω; Δ ≥ m) values: 3, 5, 11, 24, 52, 112, ...

For m = 2, the HC-ratios r(n) apparently form a non-decreasing sequence (0, 0, 0, 0, 0.08$\underline{3}$, 0.08$\underline{3}$, 0.091$\underline{6}$, 0.097$\underline{2}$, 0.1019, 0.1055, 0.1084, 0.1108, 0.1129, 0.1146, ...) converging[6] to a constant of the order of ~0.12. A similar behavior might be present also for higher values of m, with the constant of the order of ~0.012 for m = 3, and ~0.001 for m =4. However, these are presently just unproved conjectures and current computational limitations make it impossible to say anything more precise.

[5] This statement is to a mere conjecture, supported only by a number of numeric tests.
[6] Since the ratios r(n) and R(n) have an upper bound of 1, confirming that the progressions are non-decreasing would suffice to prove the convergence.

Table II. Numbers of NPC's on $S_{\omega,n}$ for selected pair-properties P.

Notes: Δ refers to the absolute difference between neighboring NPC elements. Since only differences are involved, the set $S_{\omega,n}$, may have any integer offset ω. In columns 2, 3, and 4, the Δ values have a lower limit, while in columns 5 and 6 they have an upper limit. The last column refers to the case when the differences are required to be prime. A question mark in a cell indicates that the value could not be yet computed because of excessively long execution time (usually days on a PC).

OEIS:	A242522	A242523	A242524	A242525	A242526	A228626
n	$\Delta \geq 2$	$\Delta \geq 3$	$\Delta \geq 4$	$\Delta \leq 3$	$\Delta \leq 4$	Prime Δ
1	0	0	0	1	1	0
2	0	0	0	1	1	0
3	0	0	0	1	1	0
4	0	0	0	3	3	0
5	1	0	0	6	12	1
6	5	0	0	10	36	2
7	33	1	0	17	90	4
8	245	11	0	31	214	16
9	2053	125	1	57	521	60
10	19137	1351	24	104	1335	186
11	196705	15330	504	188	3473	433
12	2212037	184846	8320	340	9016	2215
13	27029085	2382084	131384	616	23220	11788
14	356723177	32795170	2070087	1117	59428	76539
15	?	481379278	33465414	2025	152052	414240
16	?	?	561681192	3670	389636	2202215
17	?	?	?	6651	999776	9655287
18	?	?	?	12054	2566517	69748712
19	?	?	?	21847	6586825	444195809

Example 5: Maximum difference Δ: NPC(n; S_ω; $\Delta \leq m$)

There is no reason why the difference between neighbors, rather than being bounded from below as in the preceding Section, should not be bounded from above by choosing a maximum value M and imposing $\Delta \leq M$. In principle, we could as well impose both a lower bound and an upper bound[7], $m \leq \Delta \leq M$.

For $\Delta \leq M$, the most restrictive cases are M = 1, admitting no NPC solution for any cycle length n > 2, and M = 2 with exactly one NPC for every n, namely[8] :

for even n:	{0, 1, 3, 5, ..., n-1, n-2, n-4, n-6, ..., 4, 2}
for odd n:	{0, 1, 3, 5, ..., n-2, n-1, n-3, n-5, ..., 4, 2}

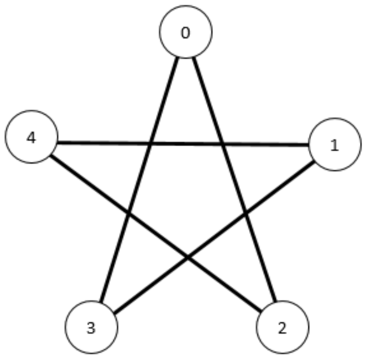
For higher values of M, the counts start increasing, as illustrated on the two examples listed in Table II, columns 5 and 6. For a given cycle length n, when M reaches n-1, the restriction becomes ineffective, and the count becomes equal to H(n). Vice versa, keeping M fixed and increasing n, the restriction becomes progressively heavier and the ratio r(n) drops, apparently converging to zero (though, again, no proof is available).

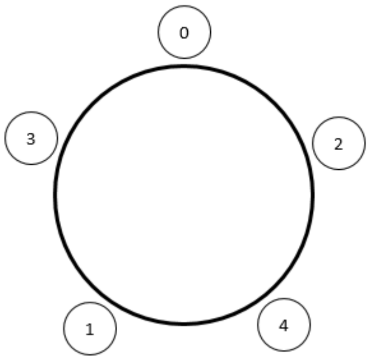
[7] If you want to try, the freeware NPC utility software permits it.
[8] The extra spaces in the cycles script delimit its logical sections.

Example 6: Prime difference Δ: NPC(n; S_ω; prime Δ)

As a last example based exclusively on differences between neighbors, let us require that each difference between neighbors be a prime number [10]. This turns out to be possible for any cycle length exceeding n = 5 with the NPC counts growing fast from 1 for n = 5, to as many as 414240 already for n = 15.

For n = 5, the unique solution can be easily found "visually" by inspecting the corresponding simple graph shown below on the left. In this case, the only possible NPC cycle is, obviously, C = {0, 2, 4, 1, 3}, shown below on the right as a round-table seating.





Selected NPC's with more general pair-properties P

When the pair-property cannot be expressed in terms of a simple difference Δ of the element values, the most important consequence is that all the pertinent results, such as the respective NPC counts, become dependent on the offset ω of the set $S_{\omega,n}$ and care must be taken to specify the value of ω.

Example 7: Prime sum: NPC(n; S_0; prime Σ) and NPC(n; S_1; prime Σ)

In Example 6, we have seen that there exist NPC(n; S_ω; prime Δ) for any $n \geq 5$. It is interesting to see whether one can form also NPC's in which the sum, rather than the difference, of all neighboring pairs is prime. We of course expect the offset ω to have an influence in this case and therefore the counts NPC(n; S_0; prime Σ) and NPC(n; S_1; prime Σ) to be different.

The results are shown in columns 4 and 5 of Table III.

In the case of NPC(n; S_0; prime Σ) there seems to exist at least one NPC solution for every $n \geq 5$, with the counts becoming prominently larger for even n, beyond $n > 17$. However, it is very hard to make any firm prediction for large n values, especially considering the dropping density of primes. The two unique cycles for n = 5 and n = 6, and the first of the 22 solutions for n= 10 are:

NPC(5; S_0; prime Σ):	C = {0, 2, 1, 4, 3},
NPC(6; S_0; prime Σ):	C = {0, 3, 4, 1, 2, 5},
NPC(10; S_0; prime Σ):	C_1 = {0, 3, 2, 1, 4, 9, 8, 5, 6, 7}.

The case of NPC(n; S_1; prime Σ) has already studied by others [11]. It appears to be more complicated than NPC(10; S_0; prime Σ), even though it The odd length cycles of this type admit no solution (apart from the trivial case n = 1). Most of the counts for even cycle length are smaller than those on $S_{0,n}$, but there again exist exceptions, for n = 10, and n = 16. It is far from clear how to tackle these problems theoretically. As examples, these are the unique cycle for n = 6, and the first one of the 48 distinct solutions for n = 10:

NPC(6; S_1; prime Σ):	C = {1, 4, 3, 2, 5, 6},
NPC(10; S_1; prime Σ):	C_1 = {1, 2, 3, 4, 7, 6, 5, 8, 9, 10}.

Example 8: Prime sum and difference: NPC(n; S_0; prime ΣΔ) and NPC(n; S_1; prime ΣΔ)

We will now combine the conditions of Examples 6 and 7 into a single one, requiring that the sum as well as the difference of any two neighbors must be prime.

Since the sum is involved, the counts NPC(n; S_0; prime Σ) and NPC(n; S_1; prime Σ) are expected to be different. Indeed, every choice of the offset ω of $S_{\omega,n}$ is bound to generate a different sequence.

From Lemma 4, we also know that, for any n and ω,

$$\text{NPC}(n; S_\omega; \text{prime } \Sigma\Delta) \le \min(\text{NPC}(n; S_\omega; \text{prime } \Delta), \text{NPC}(n; S_\omega; \text{prime } \Sigma))$$

The results, shown in columns 4 and 5 of Table III, are interesting but highly unpredictable.

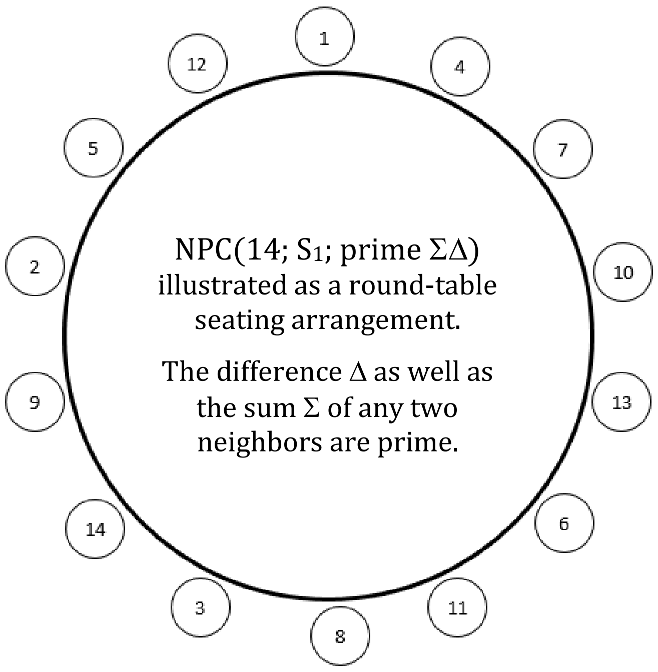
In both cases, NPC(n; S_0; prime ΣΔ) and NPC(n; S_1; prime ΣΔ), the first cycle length admitting a solution is n =12, and in both cases the solutions for n = 12 happen to be two. Explicitly:

NPC(12; S_0; prime ΣΔ):	C_1 = {0, 5, 2, 9, 4, 1, 6, 11, 8, 3, 10, 7},
	C_2 = {0, 7, 10, 3, 8, 5, 2, 9, 4, 1, 6, 11},
NPC(12; S_1; prime ΣΔ):	C_1 = {1, 4, 9, 2, 5, 12, 7, 10, 3, 8, 11, 6},
	C_2 = {1, 6, 11, 8, 3, 10, 7, 4, 9, 2, 5, 12}.

From column 5 of Table III it might appear that NPC(n; S_1; prime ΣΔ) is zero for any odd n, but there is no proof of that, nor can such a statement be trusted on the basis of the limited evidence available so far. The counts could be so far determined only up to n = 29, with non-zero values (beyond those shown in Table III) of 22277 (n = 24), 22365 (n = 26) and 376002 (n = 28).

Rather puzzling is the absence of any solution for NPC(20; S_1; prime ΣΔ), especially considering that there are NPC(18; S_1; prime ΣΔ) = 88 and NPC(22; S_1; prime ΣΔ) = 976.

Another striking case is the unique solution for NPC(14; S_1; prime ΣΔ), shown here as a round-table seating:


1
4
7
10
13
6
11
8
3
14
9
2
5
12
NPC(14; S1; prime ΣΔ) illustrated as a round-table seating arrangement.
The difference Δ as well as the sum Σ of any two neighbors are prime.

Notice the strict alternation of even and odd numbers and the fact that none of the differences is 2 (i.e., all the prime numbers involved are odd).

Table III. Numbers of NPC's on $S_{0,n}$ and $S_{1,n}$ for selected pair-properties P.

Notes:

OEIS:	A242527	A051252	A242528	A227050	A242529	A242530
S	$S_{0,n}$	$S_{1,n}$	$S_{0,n}$	$S_{1,n}$	$S_{1,n}$	$S_{1,n}$
n	**Prime Σ**	**Prime Σ**	**Prime ΣΔ**	**Prime ΣΔ**	**Coprimes**	**Gray**
1	0	1	0	0	0	0
2	0	1	0	0	1	0
3	0	0	0	0	1	0
4	0	1	0	0	1	0
5	1	0	0	0	6	0
6	1	1	0	0	2	1
7	2	0	0	0	36	0
8	6	2	0	0	36	0
9	6	0	0	0	360	0
10	22	48	0	0	288	2
11	80	0	0	0	11016	0
12	504	512	2	2	3888	8
13	840	0	4	0	238464	0
14	6048	1440	18	1	200448	0
15	3888	0	13	0	3176496	0
16	37524	40512	62	4	4257792	0
17	72976	0	8	0	402573312	0
18	961776	385072	133	88	139511808	224
19	661016	0	225	0	?	0
20	11533030	3154650	209	0	?	754
21	7544366	0	32	0	?	0
22	133552142	106906168	2644	976	?	0

Example 9: Coprime neighbors: NPC(n; S_1; coprime)

An interesting property of any pair of natural numbers is their being coprime or not. Accepting the usual convention that 1 is coprime to any non-negative integer, while 0 is coprime only to 1, it is evident, first of all, that no NPC's with coprime neighbors can exist on $S_{0,n}$. On $S_{1,n}$, however, an explicit evaluation gives the counts in column 3 of Table III. Hence, there exist NPC's with coprime neighbors on $S_{1,n}$ for any cycle length $n > 0$ and they become multiple for lengths $n \geq 5$. There is a marked difference between the behavior of the NPC(n; S_1; coprime) counts for even and odd cycle lengths.

It turns out that there are no NPC cycles with the property of all neighbors being *non-coprime*, i.e., having a common divisor greater than 1. This is easy to understand, considering that the largest prime in $S_{1,n}$ would need to have therein a neighbor at least twice as large as itself, which is a contradiction because if so, it would not be the largest one (remember that, for any n>1, there is always at least one prime between n and 2n).

For any $n > 0$, there exists one trivial solution of NPC(n; S_1; coprime), namely $C_1 = \{1, 2, 3, ..., n\}$. The first non-trivial solution is the second one of NPC(6; S_1; coprime), $C_2 = \{1, 4, 3, 2, 5, 6\}$ while, as an example, the last one of the 288 solutions for NPC(10; S_1; coprime) is $C_{288} = \{1, 8, 9, 4, 7, 6, 5, 2, 3, 10\}$.

Example 10: Gray property on $S_{1,n}$: NPC(n; S_1; Gray)

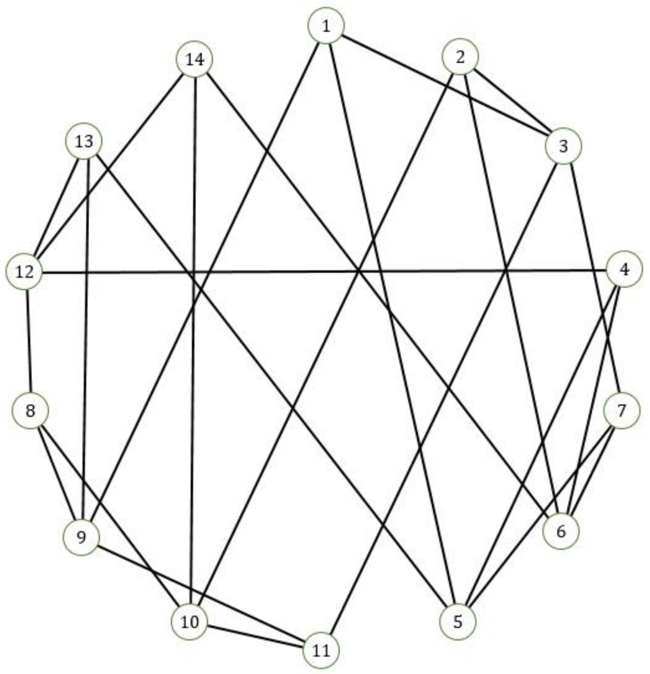
The Gray condition, i.e., imposing that the binary expansions of the two elements must differ in exactly one bit, is a good example of a pair-property which is not a simple function of Δ. The CGC (canonical Gray cycles) counts (OEIS A236602) are normally referred to the set $S_{0,n} = \{0, 1, 2, ..., n\text{-}1\}$ and the results for this case are listed in column 3 of Table 1. When, instead, the same property is applied to the set $S_{1,n} = \{1, 2, 3, ..., n\}$, the resulting counts are those of Table III, column 2. A completely different – and much more irregular – set of values! There are many cycle lengths which allow many distinct cycles on $S_{0,n}$, but which do not allow a single cycle on $S_{1,n}$. It is amazing how, on $S_{1,n}$, there do not exist any cycles of lengths 14 and 16 but, suddenly, well 226 of them for length 18. The cycle lengths, up to 30, with non-zero NPC(n; S_1; Gray) are only the following ones:

NPC(6; S_1; Gray)	= 1,
NPC(10; S_1; Gray)	= 2,
NPC(12; S_1; Gray)	= 8,
NPC(18; S_1; Gray)	= 224,
NPC(20; S_1; Gray)	= 754,
NPC(24; S_1; Gray)	= 26256,
NPC(30; S_1; Gray)	= 22472304.

The fact, mentioned already in [1], that Gray condition admits only cycles with even lengths is universal and applicable to NPC(n; S_ω; Gray) with any offset ω. The reason is that the single-bit "flips" between neighbors must in the end lead to the starting element of the cycle and therefore an even number of single-flip steps is necessary. Hence

NPC(2k+1; S_1; Gray) = 0 for any k.

Graph of NPC(14; $S_{1,n}$; Gray) for which there exists no Hamiltonian cycle.



The reasons why NPC(4; $S_{1,n}$; Gray) and NPC(8; $S_{1,n}$; Gray) are both 0 are evident when one draws the corresponding neighbor property graphs. For n = 4 there are just 2 edges and Lemma 1 is applicable.

while for n = 8, the vertex "8" is isolated (degree 0) and Lemma 3 applies. It is easy to see that the latter argument applies to any cycle length of the type 2^k, so that

$$\text{NPC}(2^k; S_1; \text{Gray}) = 0 \text{ for any k.}$$

With these two rules, the truly surprising case becomes n = 14. Its property graph shows nothing that might immediately pinpoint it as having no Hamiltonian cycle, and yet it does not have one. All its vertices have a degree of at least 3. Excluding disconnected graphs and edge- or vertex-separable graphs, I have not yet encountered a smaller simple graph with such properties. I therefore think that it might be of some interest by itself and include here its graphical representation and its adjacency matrix. The next such puzzling graphs might be those for n = 22, 26 and 28.

The adjacency matrix of NPC(14; S_1; Gray)

Note: for better readability, zeroes were omitted

n\n	1	2	3	4	5	6	7	8	9	10	11	12	13	14
1			1		1				1					
2			1			1				1				
3	1	1					1				1			
4					1	1						1		
5	1			1			1						1	
6		1		1			1							1
7			1		1	1								
8									1	1		1		
9	1							1			1		1	
10		1						1			1			1
11			1						1	1				
12				1				1					1	1
13					1				1			1		
14						1				1		1		

More examples

There are infinitely many pair properties and therefore an infinity of integer sequences for their counts as functions of the cycle length n. Apart from those discussed above, a few more such sequences were submitted to OEIS because they looked interesting and enumerating them took some effort.
These are:

OEIS A242531: NPC(n; S_1; Δ is divisor of Σ) : 0, 1, 1, 1, 1, 4, 3, 9, 26, 82, 46, 397, 283, 1675, 9938, ...
Example for n = 10: {1, 2, 4, 5, 6, 7, 8, 10, 9, 3}

OEIS A242532: NPC(n; S_2; Δ>1 is divisor of Σ) : 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 20, 39, 0, 0, 0, 0,319, ...
Example for n = 14: {2, 4, 8, 10, 5, 7, 14, 12, 15, 13, 11, 9, 3, 6}

OEIS A242533: NPC(n; S_1; Δ, Σ are coprime) : 0, 1, 0, 1, 0, 2, 0, 36, 0, 288, 0, 3888, 0, 200448, 0, ...
Example for n = 10: {1, 8, 9, 4, 7, 6, 5, 2, 3, 10}

OEIS A242533: NPC(n; S_1; Δ, Σ are not coprime) : 1, 0, 0, 0, 0, 0, 0, 0, 0, 72, 288, 3600, 17856, 174528, ...
Example for n = 10: {1, 3, 5, 10, 2, 4, 8, 6, 9, 7}

Conclusions

In this essay, we have introduced the concept of neighbor-property cycles (NPC), delimited its confines, and attempted a symbolic notation for its infinity of categories.

Each NPC category defines, at least in principle[9], an integer sequence of NPC counts for increasing cycle lengths n. Many examples of such categories were provided and briefly described. It is possible that some of the categories give rise to nontrivial finite sequences, i.e., ones that evaluate to zero for every cycle length exceeding a nontrivial maximum $n_{max} > 3$, though no actual example was encountered yet. Many of the discussed sequences of NPC counts were registered in OEIS and references to them are provided in the three Tables and elsewhere in the text.

At present there do not exist any explicit equations for the NPC counts in any non-trivial case, except that of the void property (coincident with Hamiltonian cycles in complete simple graphs). Problems of this type still represent a considerable mathematical challenge. For this reason, and for the appeal of NPC's as "round-table seating arrangements", some of the discussed examples, such as NPC(n; S_1; Gray) and NPC(n; S_1; prime $\Sigma\Delta$) have a potential for recreational mathematics.

What interested us most here was the close connection between the NPC's and the Hamiltonian cycles of simple graphs, conveniently defined by means of the same pair-property P and the same subset S of elements used to define each NPC category (the pair-property graphs). When it comes to finding an NPC of a given cycle length (or enumerating all of them) the problem is equivalent to finding (or enumerating) Hamiltonian cycles in a corresponding pair-property graph.

A potentially interesting aspect that has not been discussed yet is whether one might kind of reverse the relationship between the NPC's and the pair-property graphs. If, given a simple graph, one could redefine it in terms of an NPC for some manageable property P, there might be a chance to exploit the isomorphism. We have seen, in fact, that in several cases the property P enabled us to determine, for example, that no odd-length cycles could exist, even though this was not immediately evident from the corresponding graph. Studying the graphs jointly with the NPC's might therefore prove to be fruitful. Already the fact that the NPC's permit to define infinite categories of graphs, many of which non-trivially indexed by the offset ω and other parameters, is interesting.

Studying the case NPC(n; S_1; Gray) we have hit on highly interconnected simple graphs having no Hamiltonian cycle. The heavily "restricted" NPC's, like NPC(n; S_1; Gray) and NPC(n; S_1; prime $\Sigma\Delta$), look like handy "generators" of such graphs, a feature which might be useful to study and test Hamiltonian cycles enumeration algorithms.

The software used to carry out the enumerations is described in the Appendices. It includes also an executable utility, running under Windows XP or higher, which can be downloaded from the author's website [12]. All this, however, while useful for small cycle lengths, is severely limited by execution times for cycle lengths of the order of 20 or more (depending upon the pair-property). Some of the largest data items tabulated in the Tables I, II, and III required days of a PC time.

Enumeration of all Hamiltonian cycles in a simple graph is one of the most demanding NP-complete problems, and so is the enumeration of all NPC's for many pair-properties. That, however, does not mean that, by applying suitable algorithms, it cannot be accelerated beyond the current state-of-the art, something that will be tackled in a next article.

[9] Of course, the sequence can contain all zeroes, or all ones. Which does not make it illegitimate, just rather boring.

Appendix A: The freeware NPC utility

This essay relies on a relatively brute force algorithm to be described in Appendix B. Since the problem of finding/enumerating NPC's is in large part equivalent to finding/enumerating Hamiltonian cycles in corresponding simple graphs, one might question why a graph-theoretical algorithm was not used, considering that many are available [13-18]. The author has several answers to this objection:

a) It is fun to try an independent way, to be used as a check and, at the same time, checking on others.
b) The more independent algorithms to tackle this task there are, the better. In particular, comparing them with each other facilitates debugging and helps to avoid mistakes.
c) Some of the referenced algorithms look only for one Hamiltonian cycle, or for a specified number of them, which is insufficient for the needs of this study.
d) Unlike the software presented here, those algorithms that find multiple, or all, Hamiltonian cycles, such as [13] often do not list them in any specific order, which complicates comparisons.
e) Since the NPC's correspond to just a subset of simple graphs, NPC handling algorithms might be more efficient than those handling similar tasks in completely general simple graphs.
f) The author intends to tackle Hamiltonian cycles enumeration tasks independently in another study, using the present results as references, checks, and benchmarks.

Accompanying this article is a freeware utility [12], NPC.exe, executable on Windows-based PC's with NTFS file system (any OS from XP up). It makes it easy to play with the NPC's for a number of pair-properties, pre-defined in a user menu[10]. For each pair-property choice, the User can specify the offset ω and, in some cases, one or more additional parameters. The User can also select one of three "modes", addressing three somewhat different tasks:

- Mode 1: find the first NPC of a given length.
- Mode 2: enumerate all NPC's of a given length and list them in alphanumeric order.
- Mode 3: count the number of NPC's for all lengths between user-defined L_{min} and L_{max}.

In all cases, the User can also specify an optional output plain-text file with the compulsory extension ".txt" where the results are "dumped", in addition to being displayed in the console application window.

As already mentioned, the limit of this software, at the time of this writing in its build #13, is that of excessive execution time typical of all Hamiltonian cycles enumeration problems. Depending upon the pair-property, modes 2 and 3 start being tediously slow for cycle lengths between 15 and 30, while mode 1, from this point of view, is quite unpredictable: sometimes it finds the first solution even for cycle length 100, while in other cases it can be as slow as the other modes.

In any case, enjoy the utility!

[10] Note that the NPC utility includes NPC categories that were NOT discussed in the article, nor were the resulting sequences registered in OEIS. It can be used also to compute the counts for any value of the offset ω and/or any additional parameters. In principle, therefore, though limited to a fixed list of pair properties, the Utility could be used to generate an infinity of sequences. Among those, one can only select suitable examples, considered interesting and/or representative.

Appendix B: The C++ code

The following is a commented listing of the number-crunching core of the C++ program actually used to compute the discussed data (plus more). For those who wish to play with it, it is part of the downloadable zipped file NpcUtilitySources.zip (the NPC function is at the end of the file ConsoleAppNPC.cpp). The zipped package contains all the source files and libraries needed to compile the Utility.

As for the algorithm, the principle of backtracking has been described in many places [13-16] and is quite intuitive. Therefore, its abstract description is omitted here in favor of a very heavily commented, fully operative and fully tested C++ implementation. The only aspect which may need a few words of explanation is the mild effort at optimization which in some cases reduces execution times by a factor of about 2. But let us do that after the code presentation:

```
REAL NPC(DWORD cyclen,DWORD mode,LONG offset,
         BOOL pairproperty(LONG i,LONG j,LONG offset))
//----------------------------------------------------------------------
// This is the arithmetic core of all NPC functions.
// mode can be 1, 2, or 3  and determines the following behavior:
// mode = 1: As soon as a NPC is found, it is dumped and
//           the function exits, returning 1.
// mode = 2: All found NPC's ar dumped, and
//           the function returns their count.
// mode = 3: All NPC's are  computed, but not dumped.
//           The function returns the NPC's count.
// If no NPC is found, nothing is dumped the returned value is 0.
// The argument BOOL pairproperty(LONG i,LONG j,LONG offset) points
// to the pair-property testing function for two neighbors with
// values i+offset and j+offset. It must return "true" when the pair
// satisfies the property.
// For the prototype of the "dumping" function DumpNPC,
// and an example of its implementation, see below.
// If modes 1 or 2 are used, DumpNPC must be supplied by the user,
// while in mode 3 DumpNPC is not used at all.
//----------------------------------------------------------------------
{
  REAL   count;           // DWORD count would be conceptually better,
    // but REAL can accomodate integers up to 52 bits without overflow

  DWORD  k,bktidx;                              // Backtracking variables
  DWORD  i,j,used,npc1max;                    // Element search variables

  count  = 0;                                           // Default result
  if (!cyclen) return count;                         // Zero cycle length

  DWORD* npc = new DWORD[cyclen];  // Allocate a new array for the npc
             // The stored npc values range from 0 through cyclen-1:
             // The "offset" is added only to carry out the tests
  npc[0] = 0;                    // First element in any npc must be 0

  if (cyclen > 2) {                                  // When cyclen > 2:
```

```
                                          // Prepare optimization: start
npc1max = cyclen;   // Find largest value compatible with 0+offset
while (--npc1max) {
  if (npc1max && pairproperty(0,npc1max,offset)) break;
}
if (npc1max) {                             // And now the second largest
  while (--npc1max) {
    if (npc1max && pairproperty(0,npc1max,offset)) break;
  }
}                                           // Prepare optimization: end
                                                     // Initializations:
bktidx = 0;                                // Start backtrack index at 0
k = 0;                                                // Dummy backtrack
j = npc[k];                  // Note: 0 will never be used again for j
while (++k < cyclen) {       // npc[0] through npc[k] were done, go on
      // Now k was increased, and we are looking for a good npc[k]

  while (++j < cyclen) {       // Use j to scan all npc[k] candidates
        // starting just beyond where we left it in last backtrack
                                                  // Optimization: start
    if ((k == 1) && (j > npc1max)) {j = cyclen; break;}
                                                    // Optimization: end
    used = 0;             // Make sure it was not yet used in the npc
    for (i=1;i<k;i++){if (npc[i]==j) {used=1;break;}}
    if (!used) {                                     // If not yet used:
      // Test pair property between j and the last valid npc term:
      if (pairproperty(npc[k-1],j,offset)) {                  // If ok:
        npc[k] = j;          // We have a good npc[k] candidate ...
        if ((k+1)<cyclen) {    // If this k is not the last one ...
          bktidx = k;      // ... just remember it for backtracking
        } else {           // But if this is the last k-term, then:
          if (j >= npc[1]) {    // test if the npc is canonical ...
            //...and do an extra pair-property test for cyclicity:
            if (pairproperty(npc[0],j,offset)) {
              count = count+1;        // Found a good npc! Dump it?
              if (mode<3) DumpNPC(cyclen,npc,count,mode,offset);
            }
          }
          j = cyclen;      // Last term forces j-scan termination!
        }
        break;
      }
    }
  }
  // We are done with this k. There are three possible situations:
  // k< cyclen-1 (an intermediate term) and j<cyclen (success)
  // k< cyclen-1 but no good j exists as indicated by j==cyclen
  // k==cyclen-1 (last term) and j was forcibly set to cyclen.

  if ((mode==1) && count) break;         // mode 1; first npc found

                           // Test whether we are done with this k?
  if (j < cyclen) {        // No, just reset the j-cycle for next k
    j = 0;
```

```
      } else {                                              // Yes:
        if (!bktidx) break;          // Do we have pending backtracks?
        k = bktidx-1;                      // If yes, backtrack k, ...
        j = npc[bktidx];                     // ... initialize j, ...
        bktidx--;           // ... and decrement backtrack depth index
      }
    }
  } else if (cyclen==2) {                // Special case: cycle length 2
    npc[1] = 1;
        // Do the single test of the pair property for (npc[0],npc[1])
    if (pairproperty(0,1,offset)) {  // If satisfied, it counts as npc
      count = count+1;           // and can be dumped, if so specified
      if (mode < 3) DumpNPC(cyclen,npc,count,mode,offset);
    }
  } else if (cyclen==1) {                // Special case: cycle length 1
        // Do the single test of the pair property for (npc[0],npc[0])
    if (pairproperty(0,0,offset)) {  // If satisfied, it counts as npc
      count = count+1;           // and can be dumped, if so specified
      if (mode < 3) DumpNPC(cyclen,npc,count,mode,offset);
    }
  }
  delete [] npc;                      // Disallocate the temporary array
  return count;                                 // and return the result
}
```

Regarding the two optimization sections, they were added only after extensive tests of the non-optimized software and compared and the results of the separate runs were compared for an exact match.

The principle of the optimization is the following:

Remember the canonical condition by which the last element of an NPC cycle, npc[n-1] may not exceed the second one, npc[1]. This is intended to prevent doubling all NPC counts by "reading" the cycles both clockwise and counter-clockwise, and it is tested in the line reading

```
              if (j >= npc[1]) {   // test if the npc is canonical ...
```

at the end (!) of the whole k-cycle. But one can do much better by

a) finding first the second larges value compatible with npc[0], and
b) testing that npc[1] does not exceed it already at the beginning of the k-cycle.

The effect of this simple modification is a speed-up by a factor of about 2 (on the average).

The two optimization sections are enclosed between the comments

1) `// Prepare optimization: start`, and `// Prepare optimization: end`
2) `// Optimization: start`, and `// Optimization: end`.

References

[1] Sykora S., *Canonical Gray Cycles*, Stan's Library V, 2014, DOI 10.3247/SL5Math14.001.
[2] Sykora S., *Number of canonical Gray cycles of length 2n*. OEIS A236602.
[3] Wikipedia, *Hamiltonian path*.
[4] Wikipedia, *Hamiltonian path problem*.
[5] Weisstein E. W., *Hamiltonian Cycle*, on MathWorld.
[6] Narsingh D., *Graph Theory with Applications to Engineering and Computer Science*, Prentice Hall, Englewood Cliffs, 1974, ISBN 978-0133634730.
[7] Karp R. M., *Reducibility Among Combinatorial Problems*, in *Complexity of Computer Computations*, (Editor R. E. Miller and J. W. Thatcher), Plenum Press, pp. 85-103, 1972. ISBN 978-1468420036.
[8] Garey M. R., Johnson D. S., *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1983. ISBN 978-0716710448.
[9] Sloane N.J.A, *Order of alternating group A_n, or number of even permutations of n letters*. OEIS A001710. The numbers H(n) as defined in this article equal A001710(n-1). See also an interesting insight by Berry P., in OEIS A105752.
[10] Zhi-Wei Sun, *Number of Hamilton cycles in the undirected simple graph G_n with vertices 1,...,n which has an edge connecting vertices i and j if and only if |i-j| is prime*. OEIS A228626.
[11] McCranie Jud, *Number of essentially different ways of arranging numbers 1 through 2n around a circle so that sum of each pair of adjacent numbers is prime*. OEIS A051252.
[12] Sykora S., *NPC Utility*, an executable freeware utility. Also available is the zipped file *NPC Utility Sources*, containing also the NPC function of Appendix B.
[13] Mathematica function *FindHamiltonianCycle*, with its options "Backtrack", "Heuristic", "AngluinValiant", "Martello", and "MultiPath".
[14] *Hamiltonian Cycle* on the Stony Brook Algorithm Repository.
[15] Skiena S. S., *The Algorithm Design Manual*, Springer, 2nd Ed. 2008, ISBN 978-1848000698.
[16] Patel Dipak et al, *Hamiltonian cycle and TSP: A backtracking approach*, International Journal on Computer Science and Engineering (IJCSE), 2, p.1413, 2011.
[17] Chalaturnyk A., *A Fast Algorithm For Finding Hamiltonian cycles*, Thesis, University of Manitoba, 2008, available online.
[18] Ashay Dharwadker, *The Hamiltonian Circuit Algorithm*, Amazon CreateSpace 2011, ISBN 978-1466381377, also available online.

History of this document

25 May 2014: Assigned a DOI (10.3247/SL5Math14.001) and uploaded online.

09 Apr 2016: The title registered in the 'properties' of the PDF file did not match the actual title of the document. Since it is 'hidden', showing exclusively in the browser tab header when the browser displays the PDF, it escaped attention. At the occasion, five very minor typos were also corrected (no math).