International Journal of Mathematics and
Computer Science, **12**(2017), no. 2, 117–126

Unexpected Failure of a Greedy Choice Algorithm Proposed by Hoffman

Lauren Heller[1], Andrew Sack[2]

[1]Department of Mathematics
Wellesley College
Wellesley, MA 02481,USA

[2]University of Florida
Gainesville, FL 32611, USA

email: lheller@wellesley.edu, andrewsack@ufl.edu

(Received May 30, 2017, Accepted June 19, 2017)

Abstract

The algorithm referred to was intended to accomplish the following: Given integers n and t satisfying $1 \leq t \leq \lceil \frac{n+1}{2} \rceil$, partition $\{1, \ldots, n\}$ into t sets so that the sums over the t different sets are as nearly equal as possible. The algorithm was proposed in [2], where the partitions provided were used to color the edges of K_{n+1} so that rainbow cycles would be forbidden, and colors would appear as nearly equally often as possible.

1 Background

A *rainbow subgraph* in an edge-colored graph is a subgraph in which no color appears more than once. It is well known that it is possible to color the edges of the complete graph K_{n+1} with t colors appearing so that no cycle subgraph is rainbow if and only if $t \in \{1, \ldots, n\}$. Reference [2] is devoted to raising and answering certain questions about these rainbow-cycle-forbidding edge-colorings of K_{n+1}. Among these questions: For which t is it possible to

Key words and phrases: edge colorings, rainbow subgraphs.
AMS (MOS) Subject Classifications: 05A18, 05C15, 68W05.
This work was supported by NSF grant no. 1560257.
ISSN 1814-0432, 2017, http://ijmcs.future-in-tech.net

color the edges of K_{n+1} with t colors appearing so that there are no rainbow cycles and the different colors appear as nearly equally often as possible? The answer: $t \in \left\{1, \ldots, \left\lceil \frac{n+1}{2} \right\rceil \right\}$.

The proof in [2] that there is such a coloring for each $t \in \left\{1, \ldots, \left\lceil \frac{n+1}{2} \right\rceil \right\}$ starts with a certain rainbow-cycle-forbidding edge coloring of K_{n+1} with n colors appearing, partitions the set of n colors into t non-empty sets $C_1, \ldots, C_t$, and then combines the colors in each C_j into a single color. The resulting coloring of the edges of K_{n+1} with t colors will forbid rainbow cycles regardless of the original coloring and how $C_1, \ldots, C_t$ are chosen, so the trick lies in arranging for the numbers of occurrences of the t colors to be as nearly equal as possible.

The coloring with n colors chosen in [2] is a classic lexicographic coloring: Let the vertices of K_{n+1} be $v_0, \ldots, v_n$ and for each $i < j$ color the edge $v_i v_j$ with the j^{th} color. Then rainbow cycles are forbidden and each color $j \in \{1, \ldots, n\}$ appears on the j edges $v_0 v_j, \ldots, v_{j-1} v_j$. Therefore, all that remains to achieve a rainbow-cycle-forbidding edge-coloring of K_{n+1} with t colors, all appearing about the same number of times, is to partition $[n] = \{1, \ldots, n\}$ into sets $C_1, \ldots, C_t$ such that, if $\sigma(C_k) = \sum_{j \in C_k} j$, then $m = \sigma(C_1) \leq \cdots \leq \sigma(C_t) \leq m + 1$, for some m.

2 Greedy Algorithm

D. G. Hoffman, the co-author of [2], to whom major credit is owed for the particular result under discussion, proposed the following algorithm for obtaining $C_1, \ldots, C_t$.

1. Determine $m = \left\lfloor \binom{n+1}{2} / t \right\rfloor$ and how many values of j will have $\sigma(C_j) = m + 1$. If $\binom{n+1}{2} = mt + r$ with $0 \leq r < t$, then a coloring should have

$$\sigma(C_j) = \begin{cases} m & 1 \leq j \leq t - r \\ m + 1 & t - r + 1 \leq j \leq t. \end{cases}$$

2. Begin with $C_1, \ldots, C_t = \varnothing$. Having determined C_i for $i < j$, form C_j by iterating the following: choose the largest element k of $[n]$ such that $k \notin C_i$ for $i \leq j$ and $k + \sigma(C_j)$ is less than or equal to the desired value of $\sigma(C_j)$, either m or $m + 1$. Add this element to C_j. Repeat until there is no such element and then proceed to C_{j+1}, unless $j = t$.

Two examples:

1. Let $n = 18$ and $t = 6$. We have $\binom{n+1}{2} = 171 = 28 \cdot 6 + 3$, so we want $C_1, \ldots, C_6$ partitioning [18] such that $\sigma(C_i) = 28$ for $i = 1, 2, 3$ and $\sigma(C_i) = 29$ for $i = 4, 5, 6$. Following the instructions above, we obtain

   $C_1 = \{18, 10\}$
   $C_2 = \{17, 11\}$
   $C_3 = \{16, 12\}$
   $C_4 = \{15, 14\}$
   $C_5 = \{13, 9, 7\}$
   $C_6 = \{8, 6, 5, 4, 3, 2, 1\}$

2. Again, let $n = 18$, and, this time, $t = 10 = \left\lceil \frac{n+1}{2} \right\rceil$. Then $\binom{n+1}{2} = 171 = 17 \cdot 10 + 1$, so we want $C_1, \ldots, C_{10}$ with $\sigma(C_i) = 17$ for $i = 1, \ldots, 9$ and $\sigma(C_{10}) = 18$. Again, the instructions work: $C_1 = \{17\}, C_2 = \{16, 1\}, \ldots, C_9 = \{9, 8\}$, and $C_{10} = \{18\}$.

The co-authors of [2] were not able to prove that Hoffman's algorithm would work as needed, but this was no impediment to the proof of their edge-coloring result because in [1] it is proven that the desired partition $C_1, \ldots, C_t$ of $[n]$ exists in all cases with $1 \leq t \leq \left\lceil \frac{n+1}{2} \right\rceil$ except when n is even and $t = \left\lceil \frac{n+1}{2} \right\rceil = \frac{n}{2} + 1$, and in that case, as in the second example above, Hoffman's algorithm does work, giving $C_1 = \{n-1\}, C_2 = \{n-2, 1\}, \ldots, C_{\frac{n}{2}} = \left\{\frac{n}{2}, \frac{n}{2} - 1\right\}, C_{\frac{n}{2}+1} = \{n\}$. However, it does not work in general, as illustrated by the counterexamples below.

1. Let $n = 19$ and $t = 6$. Then $\binom{n+1}{2} = 190 = 31 \cdot 6 + 4$ and the instructions give

   $C_1 = \{19, 12\}, \sigma(C_1) = 31$
   $C_2 = \{18, 13\}, \sigma(C_2) = 31$
   $C_3 = \{17, 15\}, \sigma(C_3) = 32$
   $C_4 = \{16, 14, 2\}, \sigma(C_4) = 32$
   $C_5 = \{11, 10, 9, 1\}, \sigma(C_5) = 31$
   $C_6 = \{8, 7, 6, 5, 4\}, \sigma(C_6) = 30$

   which does not achieve the desired values of $\sigma(C_5)$ or $\sigma(C_6)$ and leaves 3 out of the union of the C_j. The algorithm fails after the addition of 1 to C_5 because no unused element of [19] will bring $\sigma(C_5)$ to 32.

2. Let $n = 23$, also with $t = 6$. In this case t evenly divides $\binom{n+1}{2} = 276$, yet the proposed algorithm still fails:

$C_1 = \{23, 22, 1\}, \sigma(C_1) = 46$
$C_2 = \{21, 20, 5\}, \sigma(C_2) = 46$
$C_3 = \{19, 18, 9\}, \sigma(C_3) = 46$
$C_4 = \{17, 16, 13\}, \sigma(C_4) = 46$
$C_5 = \{15, 14, 12, 4\}, \sigma(C_5) = 45$
$C_6 = \{11, 10, 8, 7, 6, 3\}, \sigma(C_6) = 45$

This example demonstrates the failure of an obvious variant of Hoffman's original algorithm, where if $\binom{n+1}{2} = mt + r$ for $0 \leq r < t$ then the $C_1, \ldots C_t$ are sought by greedy choice satisfying $\sigma(C_j) = m + 1$ for $1 \leq j \leq r$ and $\sigma(C_j) = m$ for $r + 1 \leq j \leq t$. This is the same as the original algorithm when $r = 0$.

3 Inductive Algorithm

In this section we will present an algorithm, in two parts, that achieves the task that Hoffman's algorithm was supposed to perform: given positive integers n, k with $k \leq \frac{n+1}{2}$, find sets $C_1, ..., C_k$ which partition $[n] = \{1, ..., n\}$ such that $\lfloor \binom{n+1}{2}/k \rfloor = m \leq \sigma(C_j) \leq m+1, j = 1, ..., k$. (In the case where n is even and $k = \lceil \frac{n+1}{2} \rceil > \frac{n+1}{2}$, as previously noted, Hoffman's algorithm always works.) Although the new algorithm runs in $O(n)$ time, it is significantly more difficult to understand than Hoffman's algorithm. Simplifications would be welcome.

Our algorithm has been extracted from the proofs of the two results in [1], which will be stated just below. To understand the statements: A partition of $[n]$ into sets $C_1, ..., C_k$ is of type $(m_1, ..., m_k)$ if and only if $\sigma(C_j) = m_j, j = 1, ..., k$. Obviously $\sum_{j=1}^{k} m_j = \binom{n+1}{2}$ is a necessary condition for such a partition to exist.

Proposition 3.1 (Proposition 2.1, [1]) Suppose that n, m, k and ℓ are positive integers satisfying $(k-1)m + \ell + \binom{k-1}{2} = \binom{n+1}{2}$. Then there is a partition of $[n]$ of type $(m, m+1, ..., m+k-2, \ell)$.

Proposition 3.2 (Corollary 2.3, [1]) Suppose that $n, m, k_1,$ and k_2 are non-negative integers satisfying $m \geq n > 0$ and $k_1(m+1) + k_2 m = \binom{n+1}{2}$.

Then there is a partition of $[n]$ of type $(\underbrace{m+1,...,m+1}_{k_1 \text{ terms}}, \underbrace{m,...,m}_{k_2 \text{ terms}})$.

In the first algorithm to follow, for integers $n, m, k,$ and ℓ satisfying the hypothesis of Proposition 3.1 a partition of $[n]$ of type $(m, m+1, ..., m+k-2, \ell)$ is produced. The algorithm is followed by a few remarks on its running time. Then we give a short finishing algorithm based on the proof of Proposition 3.2. (Corollary 2.3 in [1]) to obtain, for positive integers n and $k, k \leq \frac{n+1}{2}$, a partition of $[n]$ of type $(\underbrace{m+1,...,m+1}_{k_1}, \underbrace{m,...,m}_{k_2})$ for some non-negative integers k_1 and k_2 such that $k_1 + k_2 = k$ (so $km + k_1 = k_1(m+1) + k_2 m = \binom{n+1}{2}$ and $m = \lfloor \binom{n+1}{2}/k \rfloor$.)

The second algorithm proceeds by inputting $(n-k, m-n+1, k, m-n+k_1)$ to the earlier Proposition 3.1 algorithm and then modifying the k partition sets obtained.

The following algorithm is mostly a direct translation of the inductive argument in the proof of Proposition 2.1 in [1] with only small modifications made for special cases. Both algorithms have the sets in the partition stored as doubly linked lists to allow for $O(1)$ unions. First the algorithm for Proposition 2.1 from [1] is given by:

1: **function** PROP3.1(n, m, k, ℓ)
2: P := ordered list of k empty sets, that will become the partition
3: **if** $m + k - 2 \leq n$ **then**
4: **for** i from 1 to $k - 1$ **do**
5: **if** $m + i - 1 = 0$ **then**
6: put nothing into the ith set of P
7: **else**
8: Put $m + i - 1$ into the ith set of P
9: **end if**
10: **end for**
11: Add the unused positive integers from $m + k - 1$ to n to the last set of P
12: **else if** $m + k - 2 > n$ and $m \leq n$ **then**
13: $j = n + 1 - m$
14: **for** i from 1 to j **do**
15: Add $m + i - 1$ to the ith set of P
16: **end for**
17: The remaining $k - j$ sets $\leftarrow$ PROP3.1($m - 1$, $m + j$, $k - j$, ℓ)
18: **else if** $m \geq 2n - 2k + 3$ **then**
19: **for** i from 1 to $k - 1$ **do**
20: $B_i := \{n + 1 - i, n - 2k + 2 + i\}$
21: **end for**
22: $A_i := ith$ set in PROP3.1($n - 2k + 2$, $m - 2n + 2k - 3$, k, ℓ)
23: **for** i from 1 to $k - 1$ **do**
24: ith set in P $\leftarrow A_i \cup B_i$
25: **end for**
26: kth set of P $\leftarrow A_k$
27: **else if** $m < 2n - 2k + 3$ and $m + \lfloor \frac{k-1}{2} \rfloor - 1 \geq 2n - 2k + 3$ **then**
28: $j = 2n - 2k + 4 - m$
29: $C' :=$ PROP3.1($n - 2k + 2$, $m - 2n + 2k + 2j - 4$, $k - 2j + 1$, ℓ)
30: **for** i from 1 to $j - 1$ **do**
31: Add $n - k + 1 + 2i$ to the ith set of P
32: Add $n - k - j + 2 - i$ to the ith set of P
33: **end for**
34: **for** i from 1 to j **do**
35: Add $n - k + 2i$ to the $(j - 1 + i)$th set of P
36: Add $n - k + 2 - i$ to the $(j - 1 + i)$th set of P
37: **end for**
38: **for** i from 1 to $k - 2j$ **do**

Add $n-k+2j+i$ to the $(2j-1+i)$th set of P
Add $n-k-2j+3-i$ to the $(2j-1+i)$th set of P
Union the ith set in C' to the $(2j-1+i)$th set of P
end for
Union the last set of P with the last set in C'
else if $m+\lfloor\frac{k-1}{2}\rfloor < 2n-2k+3$ and k is even **then**
for i from 1 to $\frac{k}{2}-1$ **do**
Add $n-k+1+2i$ to the ith set of P
Add $m-n+k-2-i$ to the ith set of P
end for
for i from 1 to $\frac{k}{2}$ **do**
Add $n-k+2i$ to the $(\frac{k}{2}-1+i)$th set of P
Add $m-n+\frac{3}{2}k-2-i$ to the $(\frac{k}{2}-1+i)$th set of P
end for
Add the unused integers from 1 to n to the last set of P
else if $m+\lfloor\frac{k-1}{2}\rfloor < 2n-2k+3$ and k is odd[1] **then**
for i from 1 to $\frac{k-1}{2}$ **do**
Add $n-k+1+2i$ to the ith set of P
Add $m-n+k-2-i$ to the ith set of P
end for
for i from 1 to $\frac{k-1}{2}$ **do**
Add $n-k+2i$ to the $(\frac{k}{2}+i)$th set of P
Add $m-n+\frac{3}{2}(k-1)-i$ to the $(\frac{k}{2}+i)$th set of P
end for
Add the unused integers from 1 to n to the last set of P
end if
return P
end function

The proof of correctness of the algorithm for PROP3.1 is in [1] although induction on the algorithm will give it as well. The algorithm works by reducing the values of $n, m, k,$ and ℓ to smaller, different values. Most notably, in each recursive step the value of n is reduced. PROP3.1 can be shown to run in $O(n)$ time by induction as follows:

Let $\mathrm{T}(n)$ be the run-time of the algorithm. We will prove that $\mathrm{T}(n) \leq cn$ where c is some arbitrary large constant. We will use x as a large constant less than c.

[1] These cases are exhaustive.

- If the parameters fall under the case in line 3, then the for loop on line 4 has run-time $O(k)$ and line 11 has run-time $O(n)$ using Counting Sort, so the algorithm in this case has run-time $O(n+k) = O(n)$ because $n > k$.

- If the parameters fall under the case in line 12, then the for loop on line 14 runs in $x(n+1-m)$ steps and line 17 has runs in $\mathrm{T}(m-1) \leq c(m-1)$ steps by the induction hypothesis, so the algorithm in this case runs in $x(n+1-m)+(cm-c) \leq cn+c-cm+cm-c = cn$ steps, so the run-time $= O(n)$.

- If the parameters fall under the case in line 18, then the for loop in line 19 runs in $x(k-1)$ steps, line 22 has runs in $\mathrm{T}(n-2k+2) \leq c(n-2k+2)$ steps by the induction hypothesis, the for loop in line 23 has run-time of $x(k-1)$ steps as the union of the disjoint sets is $O(1)$, and line 26 runs in $O(1)$ time, so this case runs in $x(k-1)+c(n-2k+2)+x(k-1+1) \leq c(n-2k+2)+x(2k-1) \leq c(n-2k+2)+c(2k-2) = cn$, so the runtime of the algorithm in this case is $O(n)$.

- If the parameters fall under the case in line 27, then line 29 runs in $\mathrm{T}(n-2k+2) = c(n-2k+2)$ by the induction hypothesis, the for loops in line 30, 34, and 38 run xk total steps, and line 43 runs in $x(1)$ step, so this case runs in $c(n-2k+2)+x(k+1) \leq cn$ steps, so the runtime of the algorithm in this case is $O(n)$.

- If the parameters fall under the case in line 44, then the for loop in line 45 has runtime $O(k)$, the for loop in line 49 has runtime $O(k)$, and line 53 has runtime $O(n)$ using Counting Sort to determine the unused integers, so the algorithm in this case has runtime $O(n)$.

- If the parameters fall under the case in line 54, then the for loop in line 55 has runtime $O(k)$, the for loop in line 59 has runtime $O(k)$ and line 63 has runtime $O(n)$ using Counting Sort, so the algorithm in this case has runtime $O(n)$.

As all cases have been shown to run in $O(n)$, this algorithm has runtime $O(n)$.

The following algorithm is a translation of Corollary 2.3 from [1] into pseudocode, which provides, with PROP3.1(n, m, k, ℓ), an algorithm that will

achieve the purpose proposed in [2].

The inputs to this algorithm are $n > 1$ and k, satisfying $k \leq \frac{n+1}{2}$. Notice that at line 5, after $m = \lfloor \frac{n(n+1)}{2k} \rfloor, k_1 = \frac{n(n+1)}{2} - mk$, and $k_2 = k - k_1$ are defined, the algorithm PROP3.1$(n-k, m-n+1, k, m-n+k_1)$ is invoked, so the quadruple $(n-k, m-n+1, k, m-n+k_1)$ should satisfy the requirements of that algorithm – see the hypothesis of Proposition 3.1. In fact, it is straightforward to see that $k \leq \frac{n+1}{2}$ and $n > 1$ imply that all is well except in the case $k = \frac{n+1}{2}$, in which case $m - n + k_1 = 0$ which is not a positive integer. However, we find that requiring ℓ to be a positive integer in Proposition 3.1 is superfluous; the proposition holds with $\ell = 0$ in which case the last set in the partition whose existence is claimed is $\emptyset$. Further, we claim that the algorithm PROP3.1(n, m, k, ℓ) works fine when (n, m, k, ℓ) satisfy the requirements of Proposition 3.1, with $\ell = 0$ allowed.

Ironically, $k = \frac{n+1}{2}$ is another case in which Hoffman's algorithm works; n must be odd and Hoffman's algorithm generates the $\frac{n+1}{2}$ sets $\{n\}, \{n-1, 1\}, ..., \{\frac{n+1}{2}, \frac{n-1}{2}\}$

```
 1: function PROP3.2(n, k)
 2:     k_1 := n(n+1)/2  mod k
 3:     k_2 := k − k_1
 4:     m = ⌊n(n+1)/2k⌋
 5:     P := PROP3.1(n − k, m − n + 1, k, m − n + k_1)
 6:     used := 0
 7:     for i from 1 to k_1 do
 8:         Add n − i + 1 to the ith set in P
 9:     end for
10:     for i from k_1 + 1 to k − 1 do
11:         Add n − i to the ith set in P
12:     end for
13:     Add n − k_1 to the last set in P
14:     return P
15: end function
```

As line 5 of PROP3.2 runs in $O(n)$, the for loops in lines 7 and 10 run in total $O(k)$, and every other line runs in $O(1)$, this algorithm runs in $O(n)$. The proof of correctness of this algorithm is taken directly from [1] although one may also observe that assuming the result of PROP3.1 the partition will behave as desired.

Thus we have given an algorithm that works, with proof of correctness, that has runtime $O(n)$, the same runtime as the suggested algorithm in [2].

References

[1] Hung-Lin Fu, Wei-Hsin Hu, A special partition of the set I_n, Bull. Inst. Combin. Appl., **6,** 1992, 57-60.

[2] Adam Gouge, Dean Hoffman, Peter Johnson, Laura Nunley, Luke Paben, Edge-colorings of K_n which forbid rainbow cycles, Util. Math.,**83,** 2010, 219-232.