On the Extendibility of Venn Diagrams

CSC482 Project Report

Nicholas Kobald

January 22, 2017

Abstract

The purpose of this research is to determine whether all simple Venn diagrams are extendible, as well as explore various properties of them.

1 Introduction

A *simple closed curve* is a curve that does not cross itself, and ends at the same place it begins. We define an *independent family* $C = \{C_1, C_2, ...C_n\}$, to be a set of simple closed curves where each of the 2^n *regions* formed by the intersection of $X_1, X_2, ..X_n$ is not empty where each X_j is inside or outside the region bounded by C_i. Additionally, if each region enclosed by C is connected and there are a finite number of intersections between curves of C, then C is said to be a *Venn diagram.* A Venn diagram with n curves is called a n-Venn diagram. A Venn diagram is *simple* if at most two curves intersect at any point. Finally, an n-Venn diagram is *extendible* if another curve can be added to create an $(n+1)$-Venn Diagram.

A *graph* G is defined as an ordered pair $G = (V, E)$ where V is the set of *vertices* and E is the set of *edges* corresponding to unordered pairs of vertices. In this article, we will refer to the number of vertices as n and number of edges as m. We can define G to be *weighted* by assigning weights to each edge in E. A graph G is said to be *planar* if G can be drawn on a plane with no crossing edges. Additionally, a graph is *4-regular* if every vertex v has exactly four edges incident to it. We can construct a graph embedded in the plane for any Venn diagram by creating a vertex for every intersection between two curves, and adding an edge between two vertices if the intersections they represent are adjacent along a curve in the corresponding Venn diagram. Note that for simple Venn diagrams, the resulting graph will be both planar and 4-regular.

A *path* in a graph G is a sequence of distinct vertices $v_0, v_1..v_n$ such that for every v_i, v_{i+1} there is an edge from v_i to v_{i+1}. A *cycle* in G is a path with the requirement that v_0 and v_n are the same vertex. A *Hamiltonian cycle* in a graph G is a cycle that contains every vertex in V exactly once. A graph that contains such a cycle is *Hamiltonian.* In a planar embedding of a graph G, the edges divide the plane into regions, each of which is a *face.* The *dual* of a planar graph is the graph constructed by creating a vertex for every face in the original graph and placing edges between pairs of vertices for each common edge in the original graph. Finding Hamiltonian cycles is interesting since the dual of a graph constructed from a Venn diagram has a Hamiltonian Cycle if and only if the original Venn diagram is extendible [2, 10] .

2 Previous Work

In 2015 Pruesse and Ruskey[8] showed that every simple Venn Diagram is Hamiltonian. There has also been focus on finding Venn Diagrams with symmetry with both Ruskey and Hamburger finding symmetric Venn Diagrams for Venn Diagrams with five, seven, and eleven curves [7, 9]. In 2004, Griggs proved that there exists a symmetric Venn diagram on n curves when n is prime.

Less work has been done on the extendibility of simple Venn diagrams. While it has been proven that every n-Venn diagram is extendible in the non-simple case[3], Winkler's conjecture states that every simple n-Venn diagram is extendible[11]. This has conjecture has been proven for up to 5-Venn diagrams by Bultena in 2013 [1, 6]. The problem remains open in the simple case for n greater than or equal to six.

There are 3430404 6-Venns and finding Hamiltonian Cycles, both in general and for planar graphs is a known to be a NP-Complete problem[4]. This presents a challenge for our 64-Vertex graphs. The most simple backtracking algorithm which enumerates every viable path through the graph, and runs in $O(n!)$ time on arbitrary graphs, and a $O(3^n)$ time on Venn diagram duals, which have an average degree of four, may not be feasible[5].

While there are many classes of graphs for which finding a Hamiltonian Cycle can be done in polynomial or even linear time, we were unable to determine if the duals fall into any of them. Some properties of the dual graphs, however, are immediately apparent: since the underlying graphs are planar, the duals must be planar. Further, every dual must have exactly 64 vertices, since the original graph, by definition of being a Venn diagram, has $2^6 = 64$ faces. The dual graph will also have exactly 124 edges, the same number as the original graph (since the original graph is 4-regular with 62 vertice, and by the handshake lemma $\frac{4*62}{2} = 124$).

3 Approach and Results

3.1 Framework

All approaches in this article discuss finding Hamiltonian Cycles in the duals of Venn diagrams using a backtracking approach with various heuristics and pruning strategies. Each approach uses the same algorithm to generate the duals. In addition, each approach outputs the same number of Hamiltonian cycles when run on any of the 20 5-Venns, the numbers of which are outlined in Table 4. In cases where it was possible, the output of the algorithm was also verified using a separate program. All experiments were conducted on the same machine, and were compiled by gcc with the -O3 flag.

3.2 Initial Approach

It became clear that the most straightforward back tracking approach, which enumerates every viable path through the graph, and is outlined in Algorithm 2, was not practical for finding Hamiltonian cycles in the 6-Venns, and showed poor timing results when run on 5-Venns. As a result, we refined our implementation with a number of pruning strategies.

One trivial improvement that was made was insuring that at no point is the start vertex ever completely cut off from the rest of the graph. This can be done in constant time in each recursive call by insuring that, prior to searching for the final vertex in the cycle, not every vertex adjacent to the start vertex gets visited. While this type of optimization did eventually lead to an implementation that was sufficiently quick, this change itself has little effect on timings in the 5 and 6-Venn case, it did in general have a positive effect on run time.

3.3 Preventing Cutting Off any Vertex

Consider figure 1. Suppose that the current cycle being considered is $\{..B, C, A, E..\}$, and that all of the edges incident on D are represented in the image while all other vertices may have other edges. This path clearly cannot result in a Hamiltonian cycle, since there is no way to reach the vertex D any more. To remedy this, we implement a data structure that keeps track of the used degree, and maximum degree of each vertex. This allows us to, in time proportional to the degree of the vertex, increment the used degree of each of its neighbours. If the used degree of any neighbour becomes equal to that neighbours maximum degree, we can safely force our program to take the edge to that neighbour without potentially missing a Hamilton cycle. With these changes, a path at vertex A would always pick the edge to D.

The Path $\{...F, C, A, F\}$ in Figure 2, where all the edges incident on D and B are represented, outlines another case where the current path could never be part of a Hamiltonian cycle. The same data structure as before can be used to to insure we always back track from this point immediately, before wasting any further time.

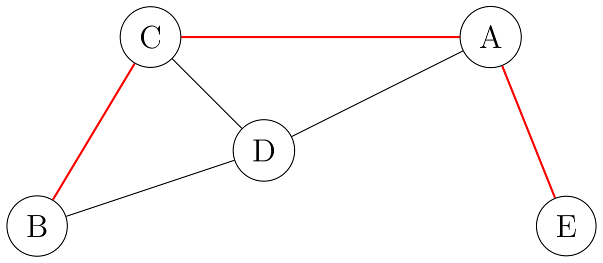


Figure 1: A Path that cannot result in a Hamiltonian Cycle

Algorithm 1 shows how this data structure is used, and Tables 1,2 compare the performance of the two implementations, comparing both time taken and recursive calls.

Table 1: Summary of results for finding all Hamiltonian in the 5-Venn Diagrams

Algorithm	Total Time (seconds)	Recursive Calls Total
Basic Backtracking	58.38	$2.55 * 10^9$
Algorithm 1	15.80	$5.58 * 10^8$
Algorithm 3	23.66	$1.67 * 10^8$
Algorithm 3 and 1	11.67	$6.57 * 10^7$

3.4 Insuring there is a way back to the start vertex

Despite the improvements outlined in the previous section, our algorithm is still not practical when run on 6-Venn duals, taking upwards of an hour on a single graph. This motivated the design of an algorithm that never recurses in a direction if there is no longer a path back to the start vertex. One obvious choice to answer such a question is breadth first search (BFS), which we implement in as a check in our recursive function. If the BFS fails to find the start vertex, we can back track from

Algorithm 1 Vertex Cutoff Approach

```
function FHC($current, G, path\_len, conflict\_arr$)
    if Start vertex is disconnected then
        Return $False$
    visited[current_vertex] ← True
    Skip ← False
    Forced ← Null
    Conflicts ← 0
    if current_vertex=start_vertex and path_len = num_vertices then
        Return $True$
    for every vertex $V$ adjacent to $current$ do
        $conflict\_arr[V].used \leftarrow conflict\_arr[V].used + 1$
        if $conflict\_arr[V].used = conflict\_arr[V].max$ then
            Forced ← $V$
            Conflicts ← Conflicts +1
            Skip ← True
    if Conflicts = 1 then
        if FHC($Forced, G, 1 + path\_len$) then
            return $True$
    if not $Skip$ then
        for every vertex $V$ adjacent to $current$ do
            if FHC($V, G, 1 + path\_len$) then
                return $True$
    for every vertex $V$ adjacent to $current$ do
        $conflict\_arr[V].used \leftarrow conflict\_arr[V].used - 1$
    visited[current_vertex] ← False
    return $False$
```

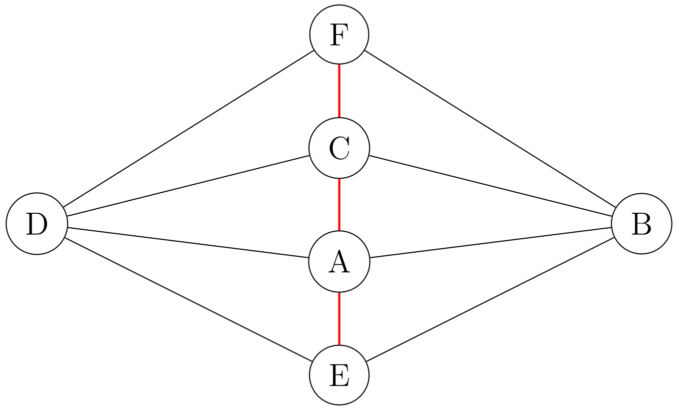


Figure 2: A worse case

the current vertex immediately. This strategy showed good results on the 5-Venn duals, and caused even more of a speed up on the larger 6-Venn duals. This implementation is outlined in Algorithm 3

Algorithm 2 A Naive Approach

```
function FHC($current, G, path\_len$)
    if Start vertex is disconnected then
        Return $False$
    $visited[current\_vertex] \leftarrow$ True
    if current_vertex=start_vertex and path_len = num_vertices then
        Return $True$
    for every vertex $V$ adjacent to $current$ do
        if FHC($V, G, 1 + path\_len$) then
            return $True$
    visited[current_vertex] $\leftarrow$ True
    return $False$
```

Table 2: Basic backtracking vs Algorithm 1
Finding all Hamiltonian Cycles in 5-Venn duals

Graph No.	Time (seconds)	Recursive Calls	Time (seconds)	Recursive Calls
	Algorithm 1	Algorithm 1	Basic Approach	Basic Approach
0	1.15	$4.07 * 10^6$	3.70	$1.63 * 10^7$
1	0.87	$3.06 * 10^6$	3.20	$1.38 * 10^7$
2	0.74	$2.63 * 10^6$	2.63	$1.16 * 10^7$
3	0.66	$2.30 * 10^6$	2.56	$1.13 * 10^7$
4	0.63	$2.19 * 10^6$	2.61	$1.16 * 10^7$
5	0.84	$2.90 * 10^6$	2.87	$1.23 * 10^7$
6	0.84	$3.09 * 10^6$	3.19	$1.38 * 10^7$
7	0.72	$2.49 * 10^6$	2.74	$1.18 * 10^7$
8	0.68	$2.47 * 10^6$	2.79	$1.18 * 10^7$
9	0.63	$2.18 * 10^6$	2.81	$1.18 * 10^7$
10	0.60	$2.02 * 10^6$	2.67	$1.14 * 10^7$
11	1.20	$3.99 * 10^6$	3.78	$1.64 * 10^7$
12	0.94	$3.37 * 10^6$	3.21	$1.42 * 10^7$
13	0.38	$1.34 * 10^6$	2.17	$9.92 * 10^7$
14	0.88	$3.19 * 10^6$	3.17	$1.40 * 10^7$
15	0.79	$2.88 * 10^6$	2.69	$1.20 * 10^7$
16	1.14	$3.97 * 10^6$	4.07	$1.64 * 10^7$
17	0.93	$3.30 * 10^6$	3.06	$1.34 * 10^7$
18	0.48	$1.70 * 10^6$	2.22	$1.00 * 10^7$
19	0.64	$2.30 * 10^6$	2.23	$1.01 * 10^7$

3.5 Results on 6-Venns

Tests were done on the first 1000 6-Venn duals, timing both the backtracking algorithm using only the BFS heuristic, and backtracking using both the BFS heuristic and the strategy outlined in Algorithm 1 with timings summarized in Figures 3 and 4. We will consider taking more than 3.0 seconds on any graph a poor result. An unexpected result of the BFS-Only algorithm is that while 8% of graphs take over 3 seconds to complete, only 3% of graphs take between 1.0 and 3.0 seconds, with all of the remaining graphs finding solutions in under 1 second. This is interesting

Table 3: Timings for BFS-augmented search on 5-Venns for finding all Hamiltonian Cycles

Graph No.	Time (seconds)	Recursive Calls	Time (seconds)	Recursive Calls
	With only BFS search		BFS Search and Pruning	
0	1.54	$1.06 * 10^7$	0.79	$4.56 * 10^6$
1	1.774	$1.33 * 10^7$	0.82	$4.47 * 10^6$
2	1.063	$7.80 * 10^6$	0.53	$3.04 * 10^6$
3	0.871	$5.992 * 10^6$	0.48	$2.54 * 10^6$
4	1.00	$6.95 * 10^6$	0.47	$2.51 * 10^6$
5	1.011	$6.71 * 10^6$	0.55	$2.99 * 10^6$
6	1.49	$1.05 * 10^7$	0.69	$3.93 * 10^6$
7	0.99	$6.99 * 10^6$	0.49	$2.68 * 10^6$
8	0.967	$6.22 * 10^6$	0.47	$2.50 * 10^6$
9	0.948	$6.29 * 10^6$	0.48	$2.52 * 10^6$
10	1.11	$8.40 * 10^6$	0.46	$2.57 * 10^6$
11	1.72	$1.32 * 10^7$	0.87	$5.11 * 10^6$
12	1.22	$8.90 * 10^6$	0.637	$3.52 * 10^6$
13	0.91	$6.94 * 10^6$	0.36	$1.94 * 10^6$
14	1.08	$7.62 * 10^6$	0.60	$3.43 * 10^6$
15	0.91	$6.00 * 10^6$	0.57	$2.99 * 10^6$
16	2.09	$1.56 * 10^7$	0.95	$5.78 * 10^6$
17	1.18	$8.35 * 10^6$	0.59	$3.43 * 10^6$
18	0.894	$5.38 * 10^6$	0.42	$2.20 * 10^6$
19	0.87	$5.34 * 10^6$	0.47	$2.50 * 10^6$

because it implies that our Algorithm performs very well in general, but on certain graphs performances extremely poorly.

The addition of the pruning from the previous section fixes this to some extent. Causing us to resolve all tested graphs in under 3 seconds, and bringing the worst-case time from 5.90 to 2.88 seconds. Using both strategies, our program finished on the first 1000 graphs in 187.84 seconds, which implies the program would take approximately 177 hours of computation to process all $3.4 * 10^6$ graphs. It is *very*

Algorithm 3 BFS Augmented Approach

```
function FHC($current, G, path\_len, conflict\_arr$)
    if Start vertex is disconnected then
        Return $False$
    Skip ← False
    visited[current_vertex] ← True
    if current_vertex=start_vertex and path_len = num_vertices then
        Return $True$
    if not path_to_origin() then
        Skip ← True
    if not $Skip$ then
        for every vertex $V$ adjacent to $current$ do
            if FHC($V, G, 1 + path\_len$) then
                return $True$
    visited[current_vertex] ← False
    return $False$
```

likely this number is very biased since our tests took place on the first 1000 graphs.

Conclusions and Future work

More sophisticated implementations of the algorithms we present may find better results. In particular, it may be possible to determine some depth of recursion where it is no longer worth using the BFS heuristic. Further, the algorithm displaying volatility in run time on different graphs suggests that relabeling the graph prior to running the recursive procedure might improve run time. We ran experiments using a breadth first search to relabel the graph, which did not noticeably improve run time, however different - perhaps random - relabelings of the graph might have better results.

Figure 3: Histogram of Timings using only search heuristic

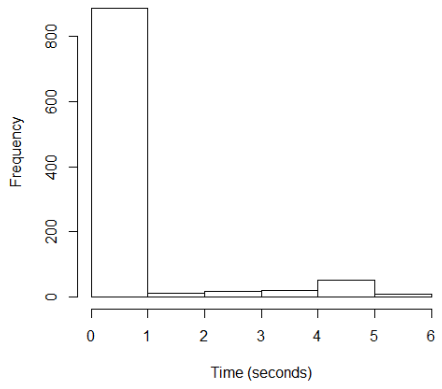
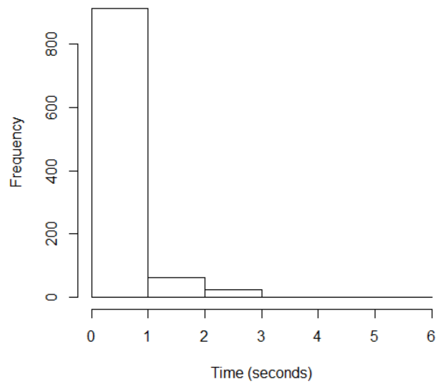


Figure 4: Histogram of Timings using both BFS search heuristic and pruning strategies



While we did not exhaustively find Hamiltonian cycles in every 6-Venn, our findings suggest that there is almost certainly no counter example to Winklers conjecture in the 6-Venns, given that the number of Hamiltonian Cycles in the graphs seem to grow exponentially from the five to six Venns, with our program having computed over $5 * 10^6$ Hamilton cycles in the first 6-Venn before being stopped.

The number of 6-Venns is significantly greater than the total number of Hamilton cycles in all of the 5-Venns. It follows immediately, then, that for most 6-Venn diagrams, there isn't a 5-Venn that can be extended to create it. Given that it is likely that a similar relationship holds between the 6 and 7 Venn diagrams, coming up with some way of calculating the number of 7-Venn diagrams will likely remain an open problem from some time.

References

[1] Bette Bultena. *Face-balanced, Venn and polyVenn diagrams.* PhD thesis, University of Victoria, 2013.

[2] Kiran B. Chilakamarri, Peter Hamburger, and Raymond E. Pippert. Hamilton cycles in planar graphs and Venn diagrams. *Journal of combinatorial theory, Series B*, 67(2):296–303, 1996.

[3] Kiran B Chilakamarri, Peter Hamburger, and Raymond E Pippert. Hamilton cycles in planar graphs and Venn diagrams. *journal of combinatorial theory, Series B*, 67(2):296–303, 1996.

[4] Michael R Garey, David S. Johnson, and R Endre Tarjan. The planar hamiltonian circuit problem is NP-complete. *SIAM Journal on Computing*, 5(4):704–714, 1976.

[5] Gregory Gutin and Abraham P Punnen. *The traveling salesman problem and its variations*, volume 12. Springer Science & Business Media, 2006.

[6] Peter Hamburger and Raymond E. Pippert. Simple, reducible Venn diagrams on five curves and hamiltonian cycles. *Geometriae Dedicata*, 68(3):245–262, 1997.

[7] Peter Hamburger and Attila Sali. Symmetric 11-Venn diagrams with vertex sets $231, 242, ..., 352$. *Studia Scientiarum Mathematicarum Hungarica*, 40(1-2):121–143, 2003.

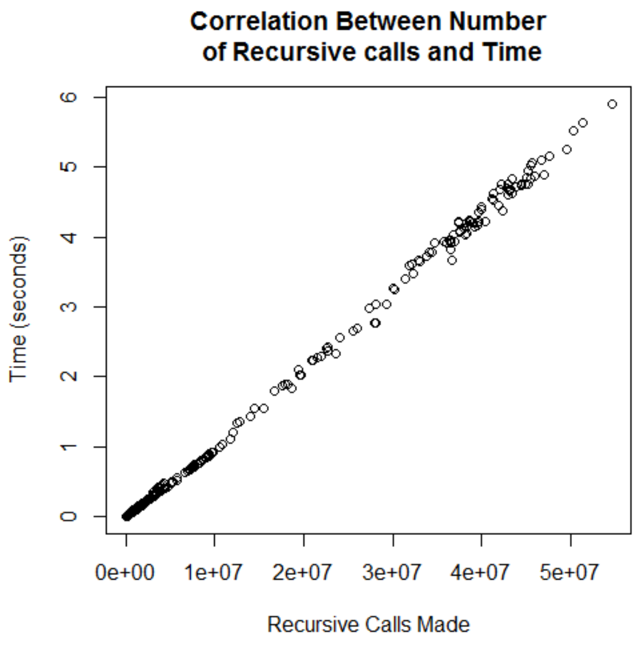
[8] Gara Pruesse and Frank Ruskey. All simple Venn diagrams are hamiltonian. *arXiv preprint arXiv:1504.06651*, 2015.

[9] Frank Ruskey, Carla D Savage, and Stan Wagon. The search for simple symmetric Venn diagrams. *Notices of the AMS*, 53(11):1304–1311, 2006.

[10] Frank Ruskey and Mark Weston. A survey of Venn diagrams. *Electronic Journal of Combinatorics*, 4:3, 1997.

[11] Peter Winkler. Venn diagrams: some observations and an open problem. *Congressus Numerantium*, 45:267–274, 1984.

A Appendix


Correlation Between Number of Recursive calls and Time
Time (seconds)
0
1
2
3
4
5
6
0e+00
1e+07
2e+07
3e+07
4e+07
5e+07
Recursive Calls Made

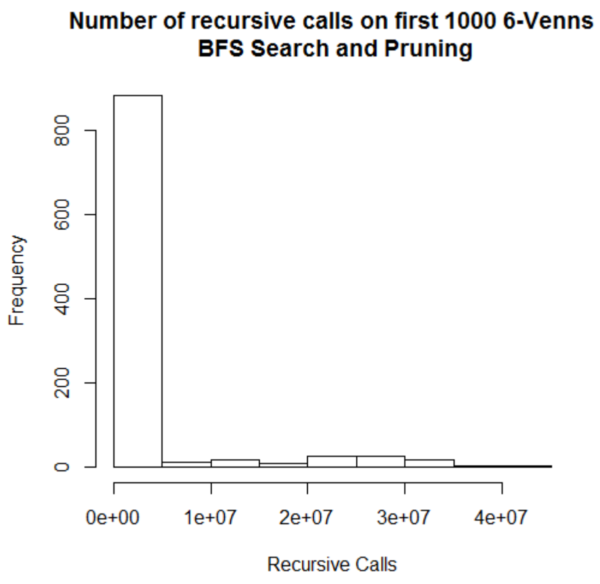

Number of recursive calls on first 1000 6-Venns BFS Search and Pruning
Frequency
0
200
400
600
800
0e+00
1e+07
2e+07
3e+07
4e+07
Recursive Calls

Table 4: Number of Hamiltonian Cycles in 5-Venn Duals

Graph No.	Number of Cycles
0	33070
1	18394
2	15468
3	14338
4	11424
5	16108
6	19316
7	14188
8	14816
9	9994
10	7989
11	29354
12	22817
13	2484
14	21640
15	16912
16	28556
17	24528
18	5884
19	13257