A Pansophy Algorithm

Jeffery J. Boats and Lazaros D. Kikas
University of Detroit Mercy
4001 W. McNichols Road, Detroit, MI 48221-3038
boatsjj@udmercy.edu , kikasld@udmercy.edu

John K. Slowik
Northeastern University
slowikjk@udmercy.edu

Abstract

Given a graph G, we are interested in finding disjoint paths for a given set of distinct pairs of vertices. In 2017, we formally defined a new parameter, the pansophy of G, in the context of the disjoint path problem. In this paper, we develop an algorithm for computing the pansophy of graphs and illustrate the algorithm on graphs where the pansophy is already known. We close with future research directions.

Keywords: Interconnection networks, graphs, algorithms, vertex disjoint paths, pansophy

1 Introduction

In 2017, Boats and Kikas introduced a new parameter, the pansophy of G, as a new measure of performance of graphical structures serving as communication networks. [3] However, pansophy is extremely difficult to compute in all but the simplest graphical structures. In this paper we introduce an algorithm for computing the pansophy of G, and we test it on graphs with known pansophy.

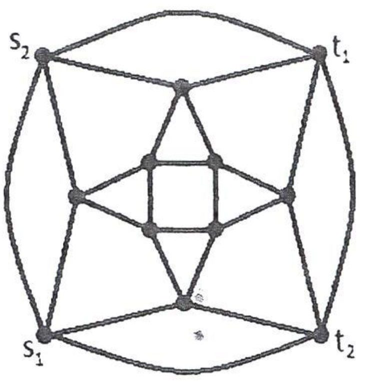

Figure 1: AG_4

2 Motivation

Consider the following problem: Given k pairs of distinct nodes $(s$ (s_2, t_2),..., (s_k, t_k), do there exist k disjoint paths, one connection for pair? This is called the **k-Disjoint Path Problem**, and has gene much research. If for a graph G we can do this for any selection of k of distinct nodes, then G is said to have the **k-Disjoint Path Prop**

In 1992, Jwo et al introduced the alternating group graph [4]. The nating group graphs AG_n have as its vertex set the set of even permuta of n symbols taken from the set $\{1, 2, 3 \ldots, n\}$. Two even permuatior adjacent if and only if one gets from one permutation to the other via tation of symbols in the first, second, and kth position where $k \in \{3, ..$ It has been proven by Kikas, Cheng, and Kruk that AG_n has the $(n$ disjoint property for $n \geq 5$ [2, 5]. However for the $n = 4$ case, i observed that AG_4 does not have the two disjoint path property, be the specific selection of (s_1, t_1) and (s_2, t_2) depicted in Figure 1 mak routing of disjoint paths impossible. [2, 5]

But if we randomly select two pairs of vertices in AG_4, the prob is high that we will be able to route the paths disjointly. Instead of t to guarantee a certain number of disjoint paths, we ask: given a ra choice of (s_1, t_1) and (s_2, t_2), what is the probability of being able to two disjoint paths? How about if we randomly select three pairs o tices? Four pairs? Finally, what is the expected value of the numl pairs we can route disjointly? This expectation represents a new pa ter, $\Psi(G)$, called the **pansophy** of G. Given a graph G and an algo for adaptively finding disjoint paths for a given random selection o tices, the pansophy of G represents optimum performance, and any rc algorithm's performance can be measured in relationship to $\Psi(G)$.

3 Definitions

Given a graph G, a maximum of $\Omega = \lfloor\frac{|V(G)|}{2}\rfloor$ vertex pairs can be selected. Given an ordered list of pairs (s_1,t_1), (s_2,t_2),...,(s_Ω,t_Ω), we wish to determine how far down the list we can go before simultaneous disjoint routing becomes impossible. In other words, we're looking for N such that the pairs $(s_1,t_1), (s_2,t_2),\ldots,(s_N,t_N)$ can be routed disjointly, but (s_{N+1},t_{N+1}) cannot because the Nth path necessarily disconnects the graph with s_{N+1} and t_{N+1} separated. We call this N the *maximal routing volume* of the (s_i,t_i)'s; the value of N varies with each selection of pairs.

Definition Let a graph G be given, and let there be a random assignment of vertex pairs (s_1,t_1), (s_2,t_2),...,(s_Ω,t_Ω), where $\Omega = \lfloor\frac{|V(G)|}{2}\rfloor$.

The **pansophy** of G, denoted $\Psi(G)$, is the expected value of the maximal routing volume.

It was observed in [3] that if a routing algorithm is given a vertex pair without foreknowledge of future pairs, the algorithm may accidentally block future connections when a non-blocking alternative was available. Since the pansophy of a graph is a measure of optimal disjoint routing, we will assume in our computations of $\Psi(G)$ that this never happens. In reality, a routing algorithm is likely to perform less than optimally. For a given list of vertex pairs on a graph G, we call the algorithm's performance its *routing volume*.

Definition A routing algorithm is **prescient** on a graph G if, given any selection of vertex pairs, its routing volume is equal to the maximal routing volume.

In other words, the pansophy of a graph is the expected value of the routing volume of any prescient algorithm.

4 Computing Pansophy

For a random assignment of Ω vertex pairs, define p_i to be the probability that the first i pairs can be disjointly routed. Define $\phi(i)$ to be the probability that i is the maximal routing value, so that $\phi(i) = p_i - p_{i+1}$. Then

$$\Psi(G) = \sum_{i=1}^{\Omega} i\ \phi(i) = \sum_{i=1}^{\Omega} i(p_i - p_{i+1}) = \ldots = \sum_{i=1}^{\Omega} p_i.$$

Hence, computing the individual p_i's is the key to computing pansophy. But this can be very difficult for large graphs. We will now demonstrate the

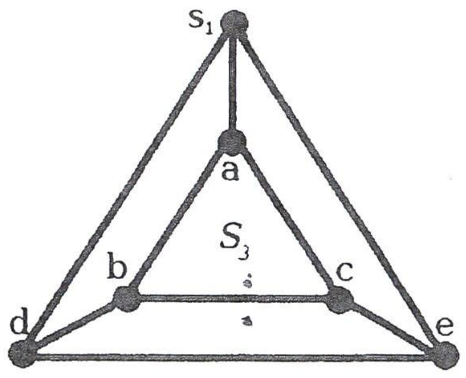

Figure 2: The symmetry group graph S_3

computation of pansophy for a small graph with exploitable symme
before moving on to the algorithmic approach.

Consider the symmetry group graph S_3, depicted in Figure 2. I
Cayley graph with vertex symmtery, which we can exploit by fixing
place, and only considering the random distribution of the other s_1'
t_i's.

1. Since S_3 is connected, we conclude that $p_1 = 1$.
2. Next we compute p_2. We are trying to find the probabilty we can the two paths disjointly given a random selection (s_1, t_1) and $(s$ We go through the cases. With s_1 fixed, there are five possib for t_1, and each occurs with probablility $\frac{1}{5}$.

   If t_1 occurs at a, d, or e, the first path is the one edge betwee adjacent s_1 and t_1, and the remainder graph is connected, so guaranteed that we can route s_2 to t_2.

   Now suppose that t_1 occurs at b. Note there are $C_{4,2}$ ways to c two vertices for s_2 and t_2, and it is only when they are placed at d that we cannot complete the routings. Therefore the probab being able to complete the routing is $\frac{5}{6}$. The same argument when t_1 is at c.

   Hence: $p_2 = \frac{1}{5} + \frac{1}{5} + \frac{1}{5} + \frac{2}{5}(\frac{5}{6}) = \frac{3}{5} + \frac{1}{3} = \frac{14}{15}$.
3. Now for p_3. Again, consider the five equally likely locations fo

   If t_1 is located at node a, the problem reduces to finding the prob of being able to route two disjoint paths in the remainder gra This probability is $\frac{2}{3}$.

If t_1 is located at b or c, the remainder graph has at most three vertices, so the probability of routing two other disjoint paths is 0.

Suppose that t_1 is located at d or e. Then the remainder graph is K_3 with a leaf, and we can route two more disjoint paths only if the leaf has a mated (s_i, t_i) pair on its ends. The probability of this event is $\frac{1}{3}$. Hence: $p_3 = \frac{1}{5}\left(\frac{2}{3}\right) + \frac{2}{5}\left(\frac{1}{3}\right) = \frac{4}{15}$.

Therefore, the pansophy of S_3 is:

$$\Psi(S_3) = p_1 + p_2 + p_3 = 1 + \tfrac{14}{15} + \tfrac{4}{15} = \tfrac{11}{5}. \quad \square$$

5 Frugality

We begin by specifying which paths are useful for routing.

Definition For vertices s and t within a graph, a connecting path from s to t is **frugal** if no proper subset of its vertices forms a connecing path.

In other words, we cannot trim anything out of the path and still connect s to t. There are no unnecessary detours, so to speak. Observe: this means that as we travel from s to t, at no point will we reach a vertex adjacent to more than one previous vertex in the path. Thus, by programming our algorithm to avoid such paths, it only finds the frugal ones.

Definition The **trail** from s to t, denoted $\text{Trail}(s,t)$, is the set of all frugal paths from s to t.

As a demonstration of how the program efficiently finds a trail, consider the graph Q shown in Figure 3, made by removing a vertex and its adjacent edges from the cube Q_3. We will find all frugal paths from v_1 to v_2, from v_1 to v_3, and then from v_1 to v_7.

Since v_1 and v_2 are adjacent, the only frugal path between them is the edge connecting them, so $\text{Trail}(v_1, v_2)$ consists only of this one path.

To begin routing from v_1 to v_3, we first notice there are three vertices adjacent to v_1, so we separately consider the three branch stems: $v_1 \to v_2$, $v_1 \to v_4$, and $v_1 \to v_5$. The first two are simple, because v_2 and v_4 are adjacent to the destination v_3, so those branches immediately go to v_7 and terminate. For $v_1 \to v_5$, the next step must be to v_6 since backtracking isn't frugal, and then we have a choice between v_4 and v_7. Both are adjacent to v_3, but the path $v_1 \to v_5 \to v_6 \to v_4 \to v_3$ isn't frugal because $v_1 \to v_4 \to v_3$ is a route whose vertices are a proper subset. The program avoids v_4 by noticing it is adjacent to v_1, a previous vertex in the path. It correctly stores $\text{Trail}(v_1, v_3)$ as having three paths: $v_1 \to v_5 \to v_6 \to v_7 \to v_3$ is frugal, along with the other two paths $v_1 \to v_2 \to v_3$ and $v_1 \to v_4 \to v_3$.

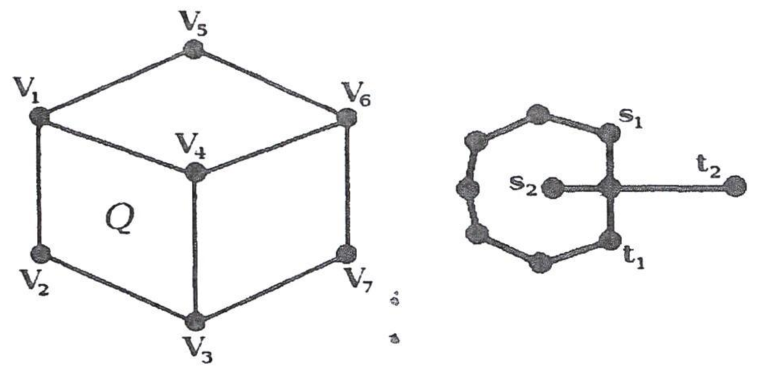

Figure 3: Finding all frugal paths between two vertices

Notice how it's not the length of the path that matters, only that ve
are trimmed off if the path could have been constructed without t
It's not hard to construct and example where taking the shortest pa
incorrect. Observe the guitar-shaped graph in Figure 3. If s_1 and t
connected by the shortest route, the graph is disconnected, while rc
the first path counterclockwise around the perimeter permits a second
from s_2 to t_2.

Now consider the frugal paths from v_1 to v_7. We start with the
three branch stems as before, but this time it's the first and third th
simple: $v_1 \to v_2 \to v_3 \to v_7$ and $v_1 \to v_5 \to v_6 \to v_7$. If we first
to v_4, we get v_3 and v_6 as next choices, and both are adjacent to v
yield frugal paths; these other two paths are $v_1 \to v_4 \to v_3 \to v$
$v_1 \to v_4 \to v_6 \to v_7$.

6 The Pansophy Algorithm

We've created a program in MATLAB, and another in JavaScript,
computes the pansophy of a given graph. It is available upon request
tact Corresponding Author), and we describe its function here.

The graph $G(V, E)$ to be analyzed is input in the form of an adja
matrix. The vertices are indexed 1 through $|V|$ based on how the
cency matrix is entered. At the time of this writing, we are only stu
undirected simply-connected graphs, so only the upper triangular p
the matrix need be entered, and the program will fill in the lower tr
automatically. It will be very easy to adapt the program to study di
graphs as well, should future research avenues demand it.

The program then begins its "SuperSubPath" routine, the purpose of which is to quickly build a library of all trails within G. There are $C_{|V|,2}$ possible choices for the end vertices of a given path, but every edge in G represents one choice where the program need not think; the only frugal path between adjacent vertices is the edge connecting them. Hence the paths library contains $C_{|V|,2}$ trails, but only $C_{|V|,2} - |E|$ need be computed. Later, each time the algorithm is given a set of (s_i, t_i) vertex pairs and attempts to maximize the number of disjoint paths, it calls on this SuperSubPath library.

The user must decide whether the algorithm should compute the maximum number of paths for every possible choice of (s_i, t_i)'s, or instead take a Monte Carlo approach.

Monte Carlo approach: the user toggles off the "TryAllPaths" switch and inputs the desired number of attempts; the program then generates a random selection of (s_i, t_i) pairs for each attempt. Once all its attempts are completed, the program takes the mean of all the results, and declares it to be $\Psi(G)$, the pansophy of G. It then also computes the sample standard deviation of all its results, which can be used to construct a confidence interval, as a reasonable estimate for the accuracy of the run. The Monte Carlo approach gives an inexact result, but its approximation error due to statistical variance can be made arbitrarily small with a sufficiently large sample.

"Try All Paths" approach: the program uses Heap's Algorithm [1] to generate the complete list of possible permutations for the (s_i, t_i) vertices, thus eliminating the need to store this list, which is impractically large for graphs of more than 12 vertices. Since swapping the s_i and t_i for any $i = 1, \ldots, \Omega$ has no effect on which paths are to be connected, nor the order of their connection, the program further reduces its workload by ignoring all (s_i, t_i) assignments where any s_i is indexed higher than its t_i; this measure alone cuts run time by a factor of 2^{Ω}. The program tries every remaining possibility, and at the end of its run, it takes the mean of all the results, and declares it $\Psi(G)$. Trying all paths will, by definition, give the exact pansophy.

For a given (s_i, t_i) set, the algorithm determines the maximal routing volume via the following steps. First, the trails for each (s_i, t_i) pair are recalled, and all frugal paths are stored using step structures, with each vertex in the graph represented as one step, and with each step leading to other steps to form the paths.

The program then begins exploring each path in Trail 1, recording the vertex it uses at each step. Whenever the end of a trail is reached, this

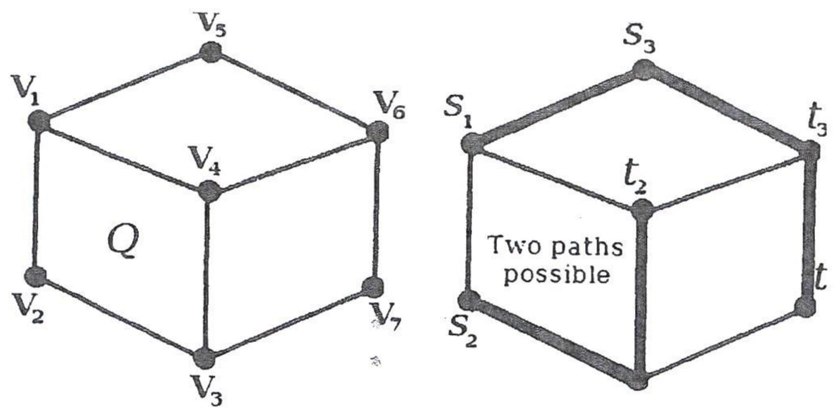

Figure 4: Finding disjoint paths within Q for a given (s_i, t_i) ass

represents a successful routing, so the current number of disjoint incremented and the program moves on to try the next trail.] point the program finds that its next child step does not exist in tł that child is skipped. Whenever the program reaches a dead enc back to the last step where it had an unexplored child step, and new branch of its search. When all valid child steps have been є the program returns those routes which constitute the maximum disjoint paths.

Consider the example in Figure 4, reusing Q, with vertex-paiı ments: $(s_1, t_1) = (v_1, v_7)$, $(s_2, t_2) = (v_2, v_4)$, $(s_3, t_3) = (v_5, v_6)$, a unpaired.

The program creates a library of trails, recording paths as ordeı

$$\begin{aligned}
\text{Trail}(v_1, v_7) &= \{1 \to 2 \to 3 \to 7,\ \ 1 \to 4 \to 6 \to 7,\\
&\qquad\qquad 1 \to 4 \to 3 \to 7,\ \ 1 \to 5 \to 6 \to 7\}\\
\text{Trail}(v_2, v_4) &= \{2 \to 1 \to 4,\ \ 2 \to 3 \to 4\}\\
\text{Trail}(v_5, v_6) &= \{5 \to 6\}
\end{aligned}$$

As the program sorts through the trails, it records step infor: Below we show the step-by-step process and the information the p stores along the way, with the following notations: let d be the re depth, v be the vertex of the current step, b be the best trail depth and in brackets are listed the vertices still available in the graph.

Recursion starts with the first path in Trail 1:

$$d = 0, v = 1, b = 0, \{2, 3, 4, 5, 6, 7\}$$

$d = 1, v = 2, b = 0, \{3, 4, 5, 6, 7\}$

$d = 2, v = 3, b = 0, \{4, 5, 6, 7\}$

$d = 3, v = 7, b = 0, \{4, 5, 6\}$

The end point has been reached, so the best path is now at least one, but this is not reflected until this recursion iteration returns. The first step of Trail 2 is now checked. Since 2 is not in the set of valid vertices, this recursion abandons this avenue and goes back to Trail 1.

$d = 0, v = 1, b = 1, \{2, 3, 4, 5, 6, 7\}$ The recursion returns to depth 0, since it has alternative steps it can take. The best trail depth has been incremented to 1.

$d = 1, v = 4, b = 1, \{2, 3, 5, 6, 7\}$

$d = 2, v = 3, b = 1, \{2, 5, 6, 7\}$

$d = 3, v = 7, b = 1, \{2, 5, 6\}$

$d = 4, v = 2, b = 1, \{5, 6\}$

Since 2 is now a valid vertex, this time we can actually try to connect Trail 2. However, none of its adjacent vertices are available, so the program abandons this avenue. The recursion returns to depth 1, since it still has alternative steps it can take at the previous d=1 step.

$d = 1, v = 4, b = 1, \{2, 3, 5, 6, 7\}$

$d = 2, v = 6, b = 1, \{2, 3, 5, 7\}$

$d = 3, v = 7, b = 1, \{2, 3, 5\}$

$d = 4, v = 2, b = 1, \{3, 6\}$

The program once again begins Trail 2, but again can't get far.

$d = 5, v = 3, b = 1, \{6\} \ldots$ and the program abandons this avenue. The recursion returns to depth 0 to check the final path in Trail 1.

$d = 0, v = 1, b = 1, \{2, 3, 4, 5, 6, 7\}$

$d = 1, v = 5, b = 1, \{2, 3, 4, 6, 7\}$

$d = 2, v = 6, b = 1, \{2, 3, 4, 7\}$

$d = 3, v = 7, b = 1, \{2, 3, 4\}$

$d = 4, v = 2, b = 1, \{3, 4\}$

$d = 5, v = 3, b = 1, \{4\}$

$d = 6, v = 4, b = 1, \{\emptyset\} \ldots$ and since a path from Trail 2 has been completed, b increments to 2, and the program begins looking at Trail 3.

Since $s_3 = v_5$ is not available, the program abandons this avenue.

Since the entire library of Trail 1 has been explored, the program and returns $b = 2$ for the best performance for this (s_i, t_i) set. □

7 Testing the Pansophy Algorithm

Previously, in [3], we demonstrated that the graphs of P_n, C_n, and
are **pansophical**, meaning that the pansophy of every graph in that cl
known, either by explicit formula or through a recursion which can be
in polynomial time. As part of the debugging process, we ran the pro
on "Try All Paths" mode for graphs of known pansophy, to see if its re
matched.

We began with the complete graphs as they represent the most t
case, since we can always connect every (s_i, t_i) pair through the edge
necting them: $\Psi(K_n) = \Omega = \lfloor \frac{n}{2} \rfloor$. The program passed this test, s
moved on to more interesting graphs.

The class $K_{m,n}$ is pansophical by polynomial time recursion, an
program's results matched perfectly up to $m = n = 5$; we have not t
further yet due to long run times. Meanwhile, P_n and C_n are pansop
by the formulas: $\Psi(P_{2n}) = \Psi(P_{2n+1}) = \sum_{i=1}^{n} \frac{1}{(2i-1)!!}$

$$\text{and} \qquad \Psi(C_n) = -1 + 2\,\Psi(P_n) \quad , \ \forall n \in \mathbf{N}.$$

Again, our program had matching results through $n = 10$.

Finally, we explored the "wheel graphs," formed by adding a vert
adjacent to every vertex in a cycle graph, i.e. $W_n = C_{n-1} + V$. We wi
proving the wheel graphs to be pansophical in an upcoming paper;
matters here is that, again, the program exactly matched our combinat
calculations.

We feel quite confident that the program can be applied accurate
any simple graph, and that it will give good approximations of pans
when in Monte Carlo mode. In order to use Try All Paths mode, we
need to improve the efficiency of the program.

8 Future Directions

The research presented here was part of an institutionally-sponsored S
mer undergraduate research project, which we are happy to say has
tinued into the school year and should continue on for some time. Ou

of MATLAB and JavaScript was to take advantage of existing subroutines which made the programs' constructions easier. Now that the students have gotten the program off-the-ground and performing well, our upcoming plans involve translating it into C++ or Python, as this will greatly improve run time. There are a number of other small inefficiencies which we will also eliminate.

Once this is accomplished, the improved program will be used to augment our continuing research into:

1. Proving other graph classes are pansophical, such as $C_n + V$, $P_n + V$, and n-partite graphs;
2. Exploring other communications structures (trees, grid graphs, etc.);
3. Exploring the pansophies of the numerous Cayley graphs which have been recommended in past research as communications structures (Q_n, S_n, AG_n, etc.), and comparing their relative efficiencies;
4. Examining the effects on pansophy of attaching a "super-user" to a graph (i.e. adding a vertex which is connected to all other vertices). Is there a relationship between $\Psi(G)$ and $\Psi(G + V)$?

References

[1] B.R.Heap. Permuations by interchanges. *The Computer Journal*, 6(3):293-294, 1963.

[2] E.Cheng, L.D. Kikas, and S.Kruk. A disjoint path problem in the alternating group graph. *Congressus Numerantium*, 175:117-159, 2005.

[3] J.Boats and L.D.Kikas. The pansophy of a graph. *Congressus Numerantium*, 229:125-134, 2017.

[4] J.S. Jwo, S.Lakshimivaharan, and S.KDhali. A new class of interconnection network based on the alternating group. *Networks*, 23:315-325, 1993.

[5] Lazaros D. Kikas. Interconnection networks and the k-disjoint path property. *Ph.D Thesis, Oakland University*, 2004.