AN ALGORITHM FOR FINDING DISJOINT PATHS IN THE ALTERNATING GROUP GRAPH

JEFFE BOATS, LAZAROS KIKAS, JOHN OLEKSIK

ABSTRACT. For the purpose of large scale computing, we are interested in linking computers into large interconnection networks. In order for these networks to be useful, the underlying graph must possess desirable properties such as a large number of vertices, high connectivity and small diameter. In this paper, we are interested in the alternating group graph, as an interconnection network, and the k-Disjoint Path Problem. In 2005, Cheng, Kikas and Kruk showed that the alternating group graph, AG_n has the $(n-2)$-Disjoint Path Property. However, their proof was an existence proof only. They did not show how to actually construct the $(n-2)$ disjoint paths. In this paper we develop an algebraic algorithm that constructs the $(n-2)$ disjoint paths from scratch. We close with remarks on possible research directions stemming from this work.

Keywords: Interconnection networks, graphs, vertex disjoint paths

1. Introduction

For the purposes of large scale computing we are interested in connecting together a large number of processors/computers. In order for these networks to be useful the underlying graph structure should have properties such as vertex symmetry, high connectivity, low diameter and have a large number of vertices. Please see [1, 2].

Recall that connectivity refers to the number of vertices one can delete from a graph without disconnecting it. Deletion of a vertex can be considered as processor failure in a computer network. We want our network to be able to handle a large number of processor failure and still have a function network. That is, we want our graph to remain connected. Thus, we want our graph to have high connectivity.

Since these networks will be used for very large computations involving large data sets, it is desirable for our graph structure to have a large number of vertices. We also want, relative to the number of vertices, for the graph to have low diameter. This reduces communication delay within the network.

The study of interconnection networks and their underlying graph topologies has been the subject of much research. For a very long time the n-cube was the network topology of choice. In 1988, the star graph was introduced

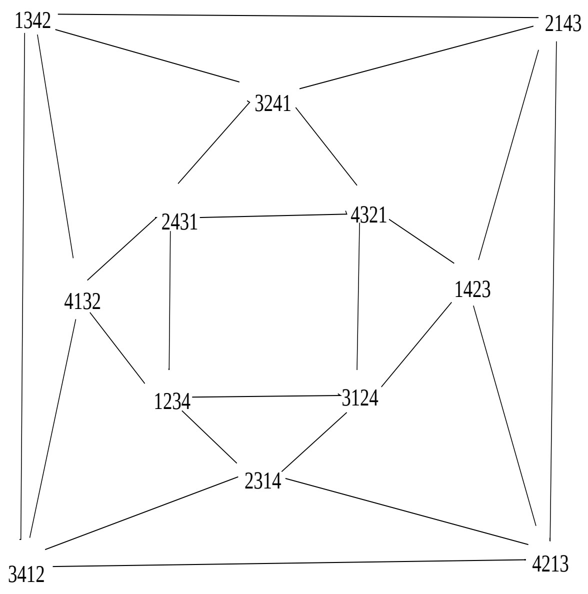

FIGURE 1. AG_4

as a competitive alternative to the n-cube [1, 2]. In the 1990's the split star and its companion graph was introduced as interconnection network topologies [3, 5].

In this paper we study the alternating group graph AG_n. Let A_n be the alternating group the group of even permutations on the symbols $1, 2, 3, ..., n$. The vertex set of AG_n is the set A_n. Two vertices of AG_n are adjacent if and only if one can get from one vertex to the other via a 3-rotation. A 3-rotation is accomplished by rotating elements in the first, second, and kth position where $k \in \{3, 4, 5, ..., n\}$. There are two types of 3-rotations: a left rotation and right rotation. As an example consider the permutation $(12345) \in AG_5$. Then both the permutations (51342) and (25431) are adjacent to (12345). Figure 1 displays the alternating group graph AG_4.

A summary of some of the basic properties of AG_n can be found in [5]. A few worth noting are that the alternating group graph AG_n is a regular graph with degree $2(n-2)$ and it has $\frac{n!(n-2)}{2}$ edges. It is also vertex symmetric and also has a hierarchal structure. Consider the subgraph of

AG_n induced by the vertices where 1 is fixed in the nth position. It is clear our subgraph will be isomorphic to AG_{n-1}. Since, we can do this for all n symbols it is clear that AG_n is made up of n copies of AG_{n-1}. These properties make the alternating group graph AG_n a novel topology in the design of parallel networks.

Suppose that we have four computers in an interconnection network and denote them by A, B, C and D. Suppose we want computer A to communicate with B and C with D simultaneously. Communication between two processors is accomplished by sending the message across a path in the network. Suppose the path of communication between A and B share a computer with the path between C and D,then during the simultaneous communication a resource has to be shared. This sharing we call a **signal collision** and can cause communication delay.

If many signal collisions occur the performance of the network is affected. Hence, signal collision is a major factor in the performance of parallel networks. The question is given an interconnection network, how many simultaneous signals can be routed through a particular network topology while avoiding signal collisions?

In graph theoretic terms we want to study the following problem: Given k pairs of distinct nodes $(s_1, t_1), (s_2, t_2), ..., (s_k, t_k)$ does there exist k-disjoint paths, one connection each pair? This problem is called the k-Disjoint Path Problem and has generated much research. If for a graph G we can do this for any selection of k pairs of distinct nodes then G is said to have the k-Disjoint Path Property.

Much has been studied about this problem. It has been shown for $k \geq 3$ the problem of finding k-disjoint paths is NP-hard. Watkins in [7] showed that if a graph G has the k-Disjoint Path Property then it must be $(2k-1)$ connected. Past work done by Cheng and Lipman shows that the split star graph, S_n^2 has the $(n-1)$-Disjoint Path Property [3]. In 2005, Cheng, Kikas, and Kruk showed that the alternating group graph AG_n has the $(n-2)$-Disjoint Path Property [4, 6]. Both these proofs for the split star and the alternating group are existence proofs; they do not provide an algorithm for the construction of these paths. In this paper, we provide geometric arguments which take advantage of the hierarchal structure of AG_5 to demonstrate the construction of the 3-disjoint paths with the elements of AG_5.

2. Preliminary Results needed for the Algorithm

Let H_i be the substar of AG_5 induced by the elements of A_5 with i fixed in the 5th place. If we consider the elements of A_5 then the elements of H_5 forms a subgroup of A_5 with cosets formed by H_1, H_2, H_3 and H_4. Let H_i^j be the substar of AG_5 induced by the elements of of A_5 with i fixed in the jth position. These substars are said to have a **j-orientation**.

Let $(s_1, t_1), (s_2, t_2), (s_3, t_3)$ be our pairs of 3 distinct nodes in AG_5. Let $S = \{s_1, s_2, s_3\}$ and $T = \{t_1, t_2, t_3\}$. In the paper [4] we have $\psi_i = |V(H_i) \cap (S \cup T)|$. Consider the tuple $(\psi_1, \psi_2, \psi_3, \psi_4, \psi_5)$. We see that we get 10 cases.

- $(6, 0, 0, 0, 0)$
- $(5, 1, 0, 0, 0)$
- $(4, 2, 0, 0, 0)$
- $(4, 1, 1, 0, 0)$
- $(3, 3, 0, 0, 0)$
- $(3, 2, 1, 0, 0)$
- $(3, 1, 1, 1, 0)$
- $(2, 2, 2, 0, 0)$
- $(2, 2, 1, 1, 0)$
- $(2, 1, 1, 1, 1)$

These ten cases are the ψ**-counts**. Define $\psi_i^j = |V(H_i^j) \cap (S \cup T)|$. We then say a ψ-count is N**-psi** if it contains N nonzero values of ψ.

It is clear that the ψ count for most configurations of the s_i's and t_i's will often be different in the different orientation and so we choose whichever available orientation gives us the easiest case. Our algorithm can be written for the $j = 5$ orientation and then, for example, should $j = 3$ be the preferred orientation the program would simply permute the third and fifth places in all the vertices, run the program and switch the indices back.

We now ask the following question. By considering the three different orientations in AG_5 is it possible to eliminate the need to consider certain ψ counts? A simple inspection makes it clear that $(6, 0, 0, 0, 0)$ in any orientation must be at least 3-psi in some other orientation [4]. Hence we can avoid 1-psi case.

Proposition 2.1. *If the elements of $S \cup T$ are 2-psi in every orientation, then they are $(3, 3, 0, 0, 0)$ in at least one orientation.*

Proof. Suppose the elements of $S \cup T$ are 2-psi in every orientation. Then elements of $S \cup T$ can have but two possibilities for the third position, say α and β two possibilities for the fourth position, say γ and δ, and two possibilities for the fifth position say ϵ and ζ. This gives us eight possible vertex forms.

- $(*, *, \alpha, \gamma, \epsilon)$
- $(*, *, \alpha, \gamma, \zeta)$
- $(*, *, \alpha, \delta, \epsilon)$
- $(*, *, \alpha, \delta, \zeta)$
- $(*, *, \beta, \gamma, \zeta)$
- $(*, *, \beta, \gamma, \epsilon)$
- $(*, *, \beta, \delta, \epsilon)$
- $(*, *, \beta, \delta, \zeta)$

For each of these eight forms,there are two possible choices for the first two positions and only one of them will be even. Now $\alpha \neq \beta$, $\gamma \neq \delta$ and $\epsilon \neq \zeta$. Since each element of $\{\alpha, \beta, \gamma, \delta, \epsilon, \zeta\}$ is an element of $\{1, 2, 3, 4, 5\}$ then by the pigeonhole principle two of the elements must be the same,thus eliminating two of the forms. Without loss of generality, let us assume that $\alpha = \gamma$, eliminating the first two forms. The remaining six forms are $(4, 2, 0, 0, 0)$ in two orientations and $(3, 3, 0, 0, 0)$ in the other. $\square$

The cases $(6, 0, 0, 0, 0)$,$(5, 1, 0, 0, 0)$, and $(4, 2, 0, 0, 0)$ can be avoided by changing orientations. Therefore, and algorithm to construct the 3-disjoint paths need only consider seven cases. We order the cases by use of the following function. We define $\theta(\psi)$ as follows: $\theta(\psi) = \sum_{1=1}^{5}(\psi_i)^2 + N$ where N is the number of nonzero ψ's. This function has the property that the ψ-counts have decreasing values of θ in the order listed above. Thus, our algorithm can thus avoid the $(6, 0, 0, 0, 0)$,$(5, 1, 0, 0, 0)$, and $(4, 2, 0, 0, 0)$ cases by computing the orientation with minimum θ value.

Now with the number of cases reduced from ten cases to seven, it must be mentioned that all remaining cases have subcases. Each different subcase requires us to develop a separate routing strategy.

In this paper the routing algorithms will be named $PATH * * * * * @$ where the $* * * * *$ denote one of the remaining seven cases and @ gives the number of pairs of (s_i, t_i)'s share a common substar.

The $PATH * * * * * @$ algorithms depend on other routing algorithms which perform simple yet crucial tasks. We will need to be able to connect two vertices within the same substar using the $CONNECT2$ algorithm and at times this algorithm will need to avoid as many as three blocks (i.e previously used vertices). We will also need to route vertices to other substars in an efficient manner by means of what we call "gating" algorithms. Both of these tasks will make use of the algebra of the alternating group.

3. Algebra of the Alternating Group

The algebra of H_i is defined by the generators of $a = (123)$ and $b = (124)$. Notice that $a^3 = b^3 = e$ where $e = (1)$. Also $aba = b^2$ and $bab = a^2$. From this we get the group elements $\{e, a, b, a^2, b^2, ab, ba, a^2b, b^2a, ab^2, b^2a, ab^2a\}$. Note a, b, a^2 and b^2 are vertices adjacent to e as they represent the 3-cycles $(123), (124), (132)$ and (142) respectively.

Any vertex in H_i can be selected as the "origin", O. Every other vertex in the substar corresponds to one of the other twelve group elements, depending on which permutations are necessary to reach it from the origin. This leads us to the following key observation. The problem of unblocked routing in a network configured in AG_5 can be shown to be equivalent to factoring group elements of A_5 over a set of generators.

From any vertex, one can move in a 4-cycle by moving a then b then a then b. This is true since $baba = (bab)a = a^2a = a^3 = e$. Similarly, $abab = e$. Also note that using a^3 and b^3 we move in a 3-cycle.

To create the $CONNECT2$ algorithm we make use of the algebraic simplification rules mentioned above. Suppose we need to route v_1 to v_2. Let $v_1 = O$ the origin. Our $CONNECT2$ algorithm must be able to avoid as many as three blocks.

Suppose v_2 is adjacent to v_1. Then creating the path from v_1 to v_2 is trivial since it cannot be blocked. We have $v_2 = gv_1$ where g is a generator or the square of a generator.

Suppose v_2 is one of the other seven vertices in H_i. A path can still be found avoiding any three "blocks" because there can always be found four disjoint paths from v_1 to v_2. The following list of equations has each beginning with one of the seven remaining vertices; each is followed by three equivalent algebraic expressions representing mutually disjoint movements from v_1 to that vertex.

- $ab = b^2a^2 = ba^2b^2a = a^2bab^2$;
- $ba = a^2b^2 = ab^2a^2b = b^2aba^2$;
- $ab^2 = a^2ba = b^2a^2b = ba^2ba^2$;
- $ba^2 = a^2b^2a = b^2ab = ab^2ab^2$;
- $a^2b = ab^2a^2 = bab^2 = b^2ab^2a$;
- $b^2a = aba^2 = ba^2b^2 = a^2ba^2b$;
- $ab^2a = ba^2b = a^2ba^2 = b^2ab^2$.

Thus, $CONNECT2$ can be constructed to connect any two vertices in the same substar, circumventing as many as three blocks and doing so with a path of at most four edges.

Definition 3.1. *A* **triad** *is a connected subgraph of H_i consisting of three vertices and two edges, with the condition that travel from one end to the other requires both an a and a b movement.*

So from the origin O, a triad takes either the form $O \Rightarrow aO \Rightarrow baO$ or $O \Rightarrow bO \Rightarrow abO$. We make the following important observation. The twelve vertices of H_i can be easily generated by constructing four disjoint triads. This can be done by initiating them at the vertices of any 4-cycles.

Definition 3.2. *A***triad partition** *is a partition of H_i into four disjoint triads.*

We do this in the following way. From the origin, O, we move in a 4-cycle. For example we have $O \Rightarrow bO \Rightarrow abO \Rightarrow babO \Rightarrow ababO = O$. From each vertex in the 4-cycle, construct an ab or ba triad, whichever is disjoint from the 4-cycle. We get the following triads, $O \Rightarrow aO \Rightarrow baO$, $bO \Rightarrow b^2O \Rightarrow ab^2O$, $abO \Rightarrow a^2bO \Rightarrow ab^2aO$, and $a^2O \Rightarrow ba^2O \Rightarrow b^2aO$.

This partitions H_i into the sets $\{O, a, ba\}$,$\{b, b^2, ab^2\}$,$\{ab, a^2b, ab^2a\}$, and $\{a^2, ba^2, b^2a\}$.

There are six distinct triad partitions of H_i formed in the above manner, since there are six distinct 4-cycles. After selecting an origin O, we can categorize them by where O appears in the triad and by whether the initial vertices move in an *abab* or *baba* 4-cycle beginning with the triad containing O.

The other five partitions are enumerated as follows.

- $\{e, b, ab\}$, $\{a, a^2, ba^2\}$, $\{ba, b^2a, ab^2a\}$, $\{b^2, ab^2, a^2b\}$
- $\{a^2, e, b\}$, $\{ba^2, ab, a^2b\}$, $\{b^2a, ab^2a, ab^2\}$, $\{a, ba, b^2\}$
- $\{b^2, e, a\}$, $\{ab^2, ba, b^2a\}$, $\{a^2b, ab^2a, ba^2\}$, $\{b, ab, a^2\}$
- $\{ba, b^2, e\}$, $\{b^2a, a, a^2\}$, $\{ab^2a, ba^2, ab\}$, $\{ab^2, a^2b, b\}$
- $\{ab, a^2, e\}$, $\{a^2b, b, b^2\}$, $\{ab^2a, ab^2, ba\}$, $\{ba^2, b^2a, a\}$

The next observation is each of the twenty four triads in the above six partition is different from the rest. There are twelve vertices in H_i, and from each can be initiated from an *ab* triad or a *ba* triad, thus there are twenty-four different triads in H_i. The pigeonhole principle assures us that every possible triad has been listed exactly once.

Proposition 3.1. *In H_i, any pair of vertices share either* 0 *or* 2 *common triads.*

Proof. Let v_1 and v_2 be two distinct vertices of H_i. Suppose a triad exist containing both of them. Then v_1 and v_2 are either b adjacent or separated by two edges.

If v_1 and v_2 are adjacent, then it is possible to move one to the other by using a or b. Suppose, without loss of generality, that $v_2 = av_1$. Then two triads can be constructed. One triad is $X \Rightarrow v_1 \Rightarrow v_2$, where $v_1 = bX$ and $v_2 = av_1 = abX$. And the other is $v_1 \Rightarrow v_2 \Rightarrow X$ where $v_2 = av_1$ and $X = bv_2 = bav_1$.

If v_1 and v_2 are separated by two edges, then there is no guarantee a triad exists containing them both. Suppose, without loss of generality, that there exists a triad of the form $v_1 \Rightarrow X \Rightarrow v_2$. Suppose that $v_2 = abv_1$. Then $abv_2 = ababv_1 = v_1$, and so there exists an *ab* triad $v_2 \Rightarrow Y \Rightarrow v_1$. Similarly, if $v_2 = bav_1$, then we have $bav_2 = babav_1 = a^2av_1 = a^3v_1 = v_1$, and so there exists an *ab* triad of the form $v_2 \Rightarrow Y \Rightarrow v_1$. Thus, v_1 and v_2 share either 0 or 2 common triads.

□

Proposition 3.2. *Given any three vertices of H_i, three disjoint triads can be found, each containing one of the veritices.*

Proof. Let v_1, v_2, and v_3 be vertices of H_i. By Proposition 3.1 there are at most two triads containing v_1 and v_2. So there are at most two triad

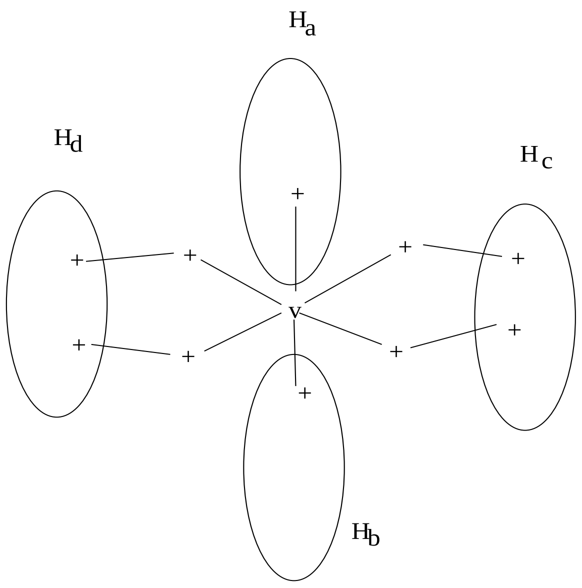

FIGURE 2. Structure of vertex $v = abcde$

partitions which join v_1 and v_2 in a common triad. A similar result is true for the pair v_2 and v_3 and also for v_1 and v_3.

Let $v_1 = O$. Suppose none of the six triad partitions can isolate v_1, v_2, and v_3 in separate triads. It can only be that v_2 and O share a triad in two of the partitions, v_3 and O share a triad in two separate partitions, and v_2 and v_3 share a triad in the remaining two. On inspection of the triad partitions, we find only cases where this could happen.

The first case is $v_1 = O, v_2 = aO, v_3 = a^2O$. From v_1, v_2 and v_3, we can initiate three new triads: $\{O, b, ab\}$, $\{a, ba, b^2\}$ and $\{a\cdot ba^2, b^2a\}$. We thus have three disjoint triads, each containing one of the vertices v_1, v_2, and v_3.

The second case is $v_1 = O, v_2 = bO, v_3 = b^2O$. Similarly, we initiate the triads $\{O, a, ba\}$, $\{b, ab, a^2\}$, and $\{b^2, ab^2, a^2b\}$. Thus, we can always construct three disjoint triads which isolates v_1, v_2, and v_3.

$\square$

Proposition 3.3. *In a substar H_i each triad contains at least one vertex connected to any other given substar.*

Proof. Consider a triad of H_i and let $k \neq i$ be given. We will show that some vertex in the triad connects to H_k. Let v be the middle vertex of the triad. Then v must take one of the three forms: $(k, *, *, *, i)$, $(*, k, *, *, i)$, $(*, *, k, *, i)$, or $(*, *, *, k, i)$.

If $v = (k, *, *, *, i)$ then it is adjacent to the vertex $(*, i, *, *, k)$ in H_k. If v is of the form $(*, k, *, *, i)$, then v connects to a vertex in H_k of the form $(i, *, *, *, k)$.

If $v = (*, *, k, *, i)$, then $av = (k, *, *, *, i)$, while $av = (*, k, *, *, i)$. Thus, if v is an ab-triad the terminal vertex of the triad is av and connects toH_k. If it is a ba-triad then the initial vertex is a^2v and connects to H_k.

If $v = (*, *, *, k, i)$, then bv has the form $(k, *, *, *, i)$ while $b^2v = (*, k, *, *, i)$. Thus if v is an ab-triad, the initial vertex of the triad is b^2v and connects to H_k. If it is a ba-triad, then the terminal vertex is bv and connects to H_k. Hence, any triad in H_i contains a vertex connecting to any other substar. □

Note that the Proposition 3.3 is also easily proved using the structure result found in [4] and Figure 2.

We use these propositions now to prove our Gateway Theorem.

Theorem 3.1 (Gateway Theorem). *For any three vertices in a substar H_i and any three substars to which these vertices are assigned, one can construct disjoint paths routing each vertex to its assigned substar.*

Proof. Let v_x, v_y and v_z be vertices H_i and assigned to substars H_x, H_y and H_z respectively. By Proposition 3.2, three disjoint triads T_x, T_y, and T_z can be generated so that $v_x \in T_x$, $v_y \in T_y$ and $v_z \in T_z$. Then, by Proposition 3.3, at least one vertex in T_x is connected to H_x. Similarly, at least one vertex in T_y is connected to H_y, and at least one vertex in T_z is connected to H_z. It follows that within each triad is a path routing each vertex to its assigned substar,and since the triads are disjoint, the paths must be disjoint. □

This enables us to compose the $GATE3$ algorithm, designed to route any three vertices to any three substars of our choice. This is useful for cases where our ψ-counts have $\psi_i = 3$ for some i. Similar algorithms $GATE1$ and $GATE2$ handle cases where only one or two such routes need to be found. Anytime such a gate is opened for some s_i the path uses at most three vertices, a complete triad, before leaving its substar for the new one. This is important because a gating vertex will therefore not block enough vertices in its originating substar to prevent two other vertices from being connected by the $CONNECT2$ algorithm.

The connection algorithm will thus avoid blockage in nearly all the ψ-counts merely by performing all necessary gating before making intra-substar connections. The only difficulty arises in those rare instances where

$(4, 1, 1, 0, 0)$ is the optimal ψ-count and $PATH41100-1$ or $PATH41100-2$ must be used.

For those cases, we now invent the algorithm $QUICKGATE$ which is used only on a substar containing four vertices from $S \cup T$. Such a substar must contain at least one mated pair. $QUICKGATE$ selects the other two vertices and immediately sends each of them to an adjacent vertex in another substar. Because this form of gating does not use triads, the two departing vertices only leave two blocks in their originating substar so that $CONNECT2$ will have no problem joining the mated pair left behind.

The catch is that, in this form of gating, each departure vertex connects directly to only two of the other four substars. For each of the two departing vertices, $QUICKGATE$ must choose which of the two adjacent substars to use. It does so in the following way. If possible, the vertex is sent to the substar of its mate. If not, then it chooses a substar devoid of the elements of $S \cup T$. If this is also not possible, then the algorithm just chooses randomly. The $(4, 1, 1, 0, 0)$ algorithms are more complicated due to the possible outcomes of using $QUICKGATE$, but each outcome is easily resolved.

4. The Algorithm

In this section we give our algorithm and its routing schemes for the various cases. The main algorithm is as follows.

Algorithm 1 (Main Algorithm)**.**

```
• Input $s_i$ and $t_i$ for $i = 1, 2, 3$.
• Compute $\theta(\psi)$ for $j = 3, 4, 5$ and choose optimal orientation.
• Count the number of mated $s_i, t_i$'s sharing a substar.
• If we are in the $(2, 2, 2, 0, 0)$ case and number of mated $(s_i, t_i)$'s is
  3 then call $CONNECT2$ three times find routes, report the paths
  and terminate.
• If not then call the appropriate $PATH * * * * * @$ and return result
• Use $CONNECT2$ three times
• Report the paths
```

Now we describe the routing algorithms coded as subroutines. In the following routing algorithms the term n-**star** will refer to a substar which, in the chosen orientation and before the algorithm runs, contains n elements of $S \cup T$. A vertex is said to be **paired** if its mate is in the same substar. Also note for unblocked routing one may use an algorithm described in [5] for simple routing in AG_n.

Algorithm 2 ($PATH21111 - 0$)**.**

```
• Use $GATE2$ to send vertices from
  the 2-star to their mates.
• Use $GATE1$ to send the remaining unpaired vertex to its mate.
• return
```

Algorithm 3 ($PATH21111-1$)**.**
- *Use $GATE1$ twice, to send unpaired vertices to their mates*
- *return*

Algorithm 4 ($PATH22110-0$)**.**
- *Use $GATE2$ on one of the 2-stars to send vertices to their mates.*
- *Use $GATE1$ on the vertex in the other 2-star to send it to its mate.*
- *return*

Algorithm 5 ($PATH22110-1$)**.**
- *Use $GATE2$ on the unpaired 2-star to send its vertices to their mates.*
- *return*

Algorithm 6 (@PATH22110-2)**.**
- *Use $GATE1$ to send the unpaired vertex to its mate.*
- *Return*

Algorithm 7 ($PATH22200-0$)**.**
- *Use $GATE2$ on a 2-star to send its vertices to separate 0-stars.*
- *Use $GATE2$ on another 2-star to send its vertices to their mates.*
- *Use $GATE1$ on the third 2-star to send the remaining unpaired vertex to its mate*
- *return*

Algorithm 8 ($PATH22200-1$)**.**
- *Select a vertex in each of the unpaired 2-stars so that they are not mated.*
- *Use $GATE1$ on the selected vertices to send them to their mates.*
- *return*

Algorithm 9 ($PATH31110-0$)**.**
- *Use $GATE3$ on the 3-star to send its vertices to their mates.*
- *return*

Algorithm 10 ($PATH31110-1$)**.**
- *Use $GATE1$ on the unpaired vertex in the 3-star to send it to its mate.*
- *Use $GATE1$ on a vertex in one of the remaining unpaired 1-stars.*
- *return*

Algorithm 11 ($PATH32100-0$)**.**
- *Use $GATE1$ on the 2-star to send one vertex to a 0-star.*
- *Use $GATE3$ on the 3-star to send its vertices to their mates*
- *return*

Algorithm 12 ($PATH32100-1$)**.**
- *Use $GATE1$ on the unpaired vertex in the 3-star to send to the 2-star.*
- *Use $GATE1$ on the remaining unpaired vertex in the 2-star to send it to its mate.*
- *return*

Algorithm 13 ($PATH32100-2$). • *Use $GATE1$ on the unpaired vertex in the 3-star to send it to its mate.*
- *return*

Algorithm 14 ($PATH33000-0$). • *Use $GATE3$ on a 3-star to send its vertices to separate 0-stars.*
- *Use $GATE3$ on the other 3-star to send its vertices to their mates.*
- *return*

Algorithm 15 ($PATH33000-2$). • *Use $GATE1$ on the unpaired vertex in one of the 3-stars to send it to a 0-star.*
- *Use $GATE1$ on the unpaired vertex in the other 3-star to send it to its mate.*
- *return*

Algorithm 16 ($PATH41100-1$). • *Use $QUICKGATE$ to evict the unpaired vertices from the 4-star.*
- *If both evictees go to their mates then return.*
- *If only one evictee goes to its mate, then the other went to a 0-star. Use $GATE1$ on the other evictee to send to its mate and return.*
- *If neither evictees goes to its mate then both went to 0-stars. If they went to the same 0-star then use $GATE2$ to send them to their mates and return. If the evictees went to separate 0-stars then use $GATE1$ twice to send evictees to their mates and return.*

Algorithm 17 ($PATH41100-2$). • *Use $QUICKGATE$ to evict one of the mated pairs from the 4-star.*
- *If the evictees go to the same substar then use $GATE1$ to send one of the vertices in a 1-star to its mate and return.*
- *If the evictees go to separate substars then use $GATE1$ to send one of the evictees to its mate and use $GATE1$ to send one of the vertices in a 1-star to its mate and return.*

5. Conclusions

The purpose of this paper was to give an algorithmic proof that AG_5 has the 3-Disjoint Path Property. Future research includes extending our algorithm for AG_n and other interconnection networks.

References

[1] S. B. Akers, D. Harel, and B. Kirshnamurthy. The star graph: An attractive alternative to the n-cube. *Proceedings of the International Conference on Parallel Processing*, pages 393–400, 1987.

[2] S. B. Akers and B. Kirshnamurthy. A group theoretic model for symmetric interconnection networks. *IEEE Transactions on Computers*, 38(4):555–566, 1989.

[3] E. Cheng and M. Lipman. Disjoint paths in split-stars. *Congressus Numerantium*, 137:47–63, 1999.

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

[5] J. S. Jwo, S. Lakshimivarahan, and S. K. Dhall. A new class of interconnection network based on the alternating group. *Networks*, 23:315–325, 1993.

[6] Lazaros D. Kikas. *Interconnection Networks and the k-Disjoint Path Problem.* PhD thesis, Oakland University, 2004.

[7] M. E. Watkins. On the existence of certain disjoint arcs in graphs. *Duke Mathematics Journal*, 35:231–246, 1968.

Jeffe Boats, Department of Mathematics and Computer Science, University of Detroit Mercy, Detroit, MI, 48221, USA
E-mail address: `boatsjj@udmercy.edu`

Lazaros Kikas, Corresponding Author, Department of Mathematics and Computer Science, University of Detroit Mercy, Detroit, MI, 48221, USA
E-mail address: `kikasld@udmercy.edu`

John Oleksik, Department of Mathematics and Computer Science, University of Detroit Mercy, Detroit, MI, 48221, USA
E-mail address: `oleksijj@udmercy.edu`