Scientific Research Publishing

Open Journal of Discrete Mathematics, 2026, 16(4), 49-64
https://www.scirp.org/journal/ojdm
ISSN Online: 2161-7643
ISSN Print: 2161-7635

On Packing Edge-Disjoint Cycles in Digraphs and Its Applications in Connectivity Problems

Vardges Melkonian

Department of Mathematics, Ohio University, Athens, Ohio, USA
Email: melkonia@ohio.edu

How to cite this paper: Melkonian, V. (2026) On Packing Edge-Disjoint Cycles in Digraphs and Its Applications in Connectivity Problems. *Open Journal of Discrete Mathematics*, **16**, 49-64.
https://doi.org/10.4236/ojdm.2026.164005

Received: August 18, 2026
Accepted: September 20, 2026
Published: September 23, 2026

Abstract

We consider the problem of packing edge-disjoint cycles in digraphs. It is shown that for $k=2,3$, any digraph with minimum degree k contains at least $k+1$ edge-disjoint directed cycles of size $\geq k$. In each of these two cases, examples are given to show that $k+1$ edge-disjoint directed cycles is the best we can achieve. These results are applied to obtain approximation algorithms for k-node-connectivity problems.

Keywords

Directed Graphs, Cycle Packing, Node Connectivity, Approximation Algorithms

1. Introduction

We consider the following fundamental problem in algorithmic graph theory. Given a digraph G, how many edge-disjoint *directed* cycles can be packed into G? Throughout the paper, a cycle in a digraph always means a directed cycle. Problems concerning packing edge-disjoint or vertex-disjoint directed cycles in directed graphs have been studied extensively [1]-[4]. It is well-known that computing the maximum size of a set of arc-disjoint directed cycles is an NP-hard problem. More broadly, understanding how degree constraints and degree sequences dictate structural properties in directed graph models—from random geometric digraphs [5] to general network topologies—remains a central theme in digraph theory.

In this paper, we consider the special cases when indegree and outdegree of every node of digraph G is (i) ≥2, (ii) ≥3. For the first case, we show that there are at least three edge-disjoint cycles in G. For the second case, we are interested not just in any cycles but in cycles of size ≥3. We show that in this case there are

at least four edge-disjoint cycles of size ≥3 in G. We also give bad case examples where three and four edge-disjoint cycles, for cases (i) and (ii) correspondingly, are the best we can achieve.

In a related work, Alon *et al.* [6] showed that a k-regular directed graph contains a collection of at least $5k/2-2$ edge-disjoint cycles. For $k=2$, like our result, this gives three directed cycles. However, the digraphs considered in our case are more general and do not have the nice property of being regular. For $k=3$, the number of cycles in [6] is 5.5 versus 4 n our result. But the cycles found in our case are of size ≥3 which is motivated by the application discussed later; and again, we consider digraphs which are more general than 3-regular digraphs.

We also apply our results to node connectivity problems. We consider the problem of finding a minimum-weight 2-node-connected (3-node-connected) subgraph of a digraph where the indegrees and outdegrees of the nodes are ≥2 (≥3). The node connectivity problems are NP-hard, and we show that our results of finding edge-disjoint cycles lead to approximation algorithms for those problems.

The paper is organized as follows. In Sections 2 and 3, we give the algorithms of finding edge-disjoint directed cycles and the worst case examples. In Section 4, these results are used to find approximation algorithms for the node connectivity problems.

2. Packing Cycles in Digraphs with Minimum Degree 2

In this section we show that in any digraph $G=(V,E)$ such that the outdegree and the indegree of any node is at least two (particularly, in any 2-node-connected digraph) there are at least three edge-disjoint directed cycles.

Definition 1 *A* **trip** *is a traversal along the arcs of a directed path such that no arc in the path belongs to any previously constructed directed cycle. A trip is called* **forward** *if we traverse each arc* $a\to b$ *from* a *to* b*; it is called* **backward** *if each* $a\to b$ *is traversed from* b *to* a.

Unless specified differently, by a trip we will mean a forward trip.

Definition 2 *A digraph is called* **balanced** *if* $\text{outdegree}(v)=\text{indegree}(v)$ *for any node* v.

Because every node has an indegree and outdegree of at least two, taking two trips in G yields two edge-disjoint directed cycles. Let C_1 and C_2 be those two cycles. Let p be the number of common nodes of C_1 and C_2. We show the existence of the third cycle by *induction on* p.

- *Basis step*. When $p=0$, each node maintains an unused indegree and outdegree of at least one; thus, taking an additional trip yields a third edge-disjoint directed cycle.

  Suppose $p=1$, and u is the only common node. Three cases are possible:
  - if u is balanced, *i.e.*, $\text{outdegree}(u)=\text{indegree}(u)$, then we can get a third cycle by starting a trip from any node $v\neq u$.
  - if $\text{outdegree}(u)>\text{indegree}(u)$ then taking a forward trip from u will give a third cycle.

- if $\text{outdegree}(u) < \text{indegree}(u)$ then taking a backward trip from u will give a third cycle.

- *Inductive step*. Suppose a third cycle exists whenever C_1 and C_2 have less than p common nodes ($p > 1$). Let's show it for the case of p common nodes. Let $u_1, \cdots, u_p$ be the common nodes indexed in the order we meet them while traversing C_1 starting from u_1.

Two cases are possible.

1) While traversing C_2 starting from u_1 we meet the common nodes in the same order as in C_1.

2) We meet them in a different order.

Case 1:

For any successive common nodes u_i and u_{i+1} we have paths $u_i \overset{C_1}{\rightsquigarrow} u_{i+1}$ and $u_i \overset{C_2}{\rightsquigarrow} u_{i+1}$. Note that we can use these two paths interchangeably for C_1 and C_2. Let $int(u_i \rightsquigarrow u_{i+1})$ denote the set of internal nodes of $u_i \overset{C_1}{\rightsquigarrow} u_{i+1}$ and $u_i \overset{C_2}{\rightsquigarrow} u_{i+1}$. We have $int(u_i \rightsquigarrow u_{i+1}) \neq \varnothing$ since $u_i \overset{C_1}{\rightsquigarrow} u_{i+1}$ and $u_i \overset{C_2}{\rightsquigarrow} u_{i+1}$ can't be the same. Let $w_i \in int(u_i \rightsquigarrow u_{i+1})$.

Start a forward trip T_i from w_i. Based on the definition of a trip, T_i is a directed path that does not include any arcs from C_1 and C_2. Consider cases.

- *Subcase* 1.1. The trip results in a cycle C_3 before hitting any node from $V(C_1) \cup V(C_2) \setminus int(u_i \rightsquigarrow u_{i+1})$. In this case C_3 is the third cycle.
- *Subcase* 1.2. We are not in subcase 1. Suppose the first node of $V(C_1) \cup V(C_2) \setminus int(u_i \rightsquigarrow u_{i+1})$ hit by the trip is $v \neq u_{i+1}$. Let w_i' be the last node from $int(u_i \rightsquigarrow u_{i+1})$ visited before v in trip T_i. Then the path $w_i' \overset{T_i}{\rightsquigarrow} v$ can serve as a shortcut for one of the original cycles, say C_1 (see Figure 1). The new cycle $C_1' = w_i' \overset{T_i}{\rightsquigarrow} v \overset{C_1}{\rightsquigarrow} w_i'$ does not contain u_{i+1}, and thus has less than p common nodes with C_2. Then by induction we have a third cycle.

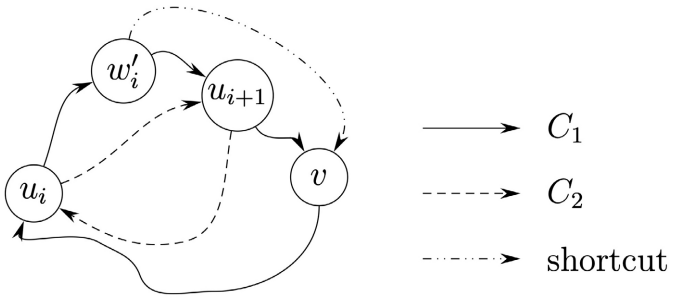

Figure 1. Example of Subcase 1.2 in the proof for the third cycle.

- *Subcase* 1.3. We are not in subcase 1. Suppose the first node of $V(C_1) \cup V(C_2) \setminus int(u_i \rightsquigarrow u_{i+1})$ hit by the trip is u_{i+1}. In this case we have a path $w_i \overset{T_i}{\rightsquigarrow} u_{i+1}$.

Starting a backward trip S_i from w_i will result in similar three subcases. In the first two subcases we will get a third cycle. In the third subcase a path $u_i \overset{S_i}{\rightsquigarrow} w_i$ will be obtained.

After having these forward and backward trips for any $i \in 1, \cdots, p$:

- Either we will get a third cycle as a result of being in subcases 1.1 or 1.2;
- Or the union of all $u_i \overset{S_i}{\rightsquigarrow} w_i$'s and $w_i \overset{T_i}{\rightsquigarrow} u_{i+1}$'s will give a third cycle. Note that even if some of these paths use the same arcs, the union of all the paths forms a strongly connected subgraph from which a directed cycle can be obtained.

Case 2:

Note that the number of common nodes of C_1 and C_2 is at least three in this case. For some common nodes u_i, u_j which are successive on C_2 we have $j > i+1$ (always can achieve this by reindexing). So $u_i \overset{C_2}{\rightsquigarrow} u_j$ creates a shortcut for C_1, *i.e.*, $Q_1 = u_i \overset{C_2}{\rightsquigarrow} u_j \overset{C_1}{\rightsquigarrow} u_i$ is a directed cycle such that $u_{i+1} \notin Q_1$ (see **Figure 2**).

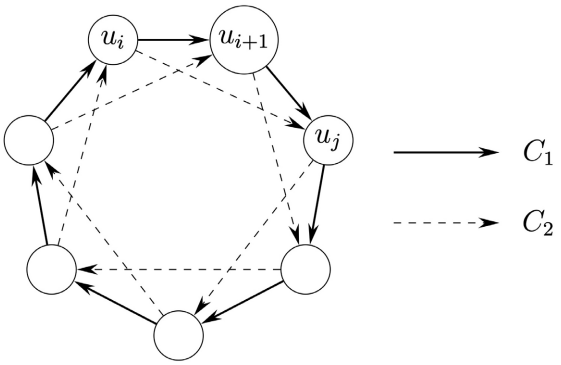

Figure 2. Example of Case 2.

Consider the graph $G' = \left(V, E\left(C_1\right) \cup E\left(C_2\right)\right)$. In this graph, each node on cycle C_1 contributes exactly 1 to both its indegree and outdegree via C_1, and the same is true for C_2. Therefore, indegree equals outdegree for every node in G', so G' is balanced. Now consider the graph $G'' = \left(V, E\left(C_1\right) \cup E\left(C_2\right) \setminus E\left(Q_1\right)\right)$. It is obtained from graph G' by removing the arcs of the directed cycle Q_1. Since removing Q_1 decreases both the indegree and outdegree of its incident nodes by exactly 1, balance is preserved, and G'' is also balanced.

Note that u_{i+1} has indegree and outdegree 2 in G'', as it lies on both C_1 and C_2 but not on Q_1. In G'', we initiate a forward trip T_1 from u_{i+1} along one of its outgoing arcs. Because G'' is balanced, entering any node for the first time guarantees an unused outgoing arc to continue T_1. Since G'' has a finite number of nodes, T_1 must eventually revisit a node, forming a directed cycle Q_2. Removing Q_2 yields a balanced graph G'''. (The argument is the same as the one given for G'' in the previous paragraph.) Because u_{i+1} had outdegree 2 in G'', its outdegree in G''' is at least 1. We can therefore initiate a second trip T_2

from u_{i+1} in G''', which similarly (as in case of T_1) yields a directed cycle Q_3. Thus, G' contains at least three edge-disjoint cycles Q_1, Q_2 and Q_3.

Note that in a worst-case scenario three cycles is the best we can achieve. To illustrate that, consider the following simple example. Let $G=(V,E)$ be a 3-node complete digraph: $V=\{a,b,c\}$, $E=\{a\to b, b\to a, a\to c, c\to a, b\to c, c\to b\}$. In this graph we can find at most three edge-disjoint directed cycles: $C_1=a\to b\to a$, $C_2=a\to c\to a$, $C_3=b\to c\to b$.

3. Packing Cycles in Digraphs with Minimum Degree 3

In this section we show that in any digraph $G=(V,E)$ such that the outdegree and the indegree of any node is at least three (particularly, in any 3-node-connected digraph) there are at least four edge-disjoint directed cycles of size ≥3.

As before, by *having a trip* we will mean traversing the arcs of a directed path which doesn't include arcs from previous trips (since we want to get edge-disjoint cycles). A trip is called *forward* if we traverse each arc $a\to b$ from a to b; it is called *backward* if each $a\to b$ is traversed from b to a. Unless specified differently, by a trip we will mean a forward trip.

A *digon* is a pair of opposite arcs. If a digraph consists only of digons, and ignoring the directions will make it an undirected tree, then the digraph is called *digon-tree*. For an example of a digon-tree see subgraph T_2 of **Figure 3**.

Lemma 1 *Every strongly connected directed graph $G=(V,E)$ is either a digon-tree or contains a directed cycle of length at least* 3.

Proof: We consider two exhaustive cases based on the symmetry of the arcs in G.

Case 1: *Every arc in G belongs to a digon*. That is, for every pair of vertices $u,v\in V$, the arc $(u,v)\in E$ if and only if $(v,u)\in E$. Let G' be the undirected graph obtained by replacing each digon $\{(u,v),(v,u)\}$ in G with a single undirected edge $\{u,v\}$. Since G is strongly connected, G' is connected.

- If G' is an undirected tree, then by definition G is a digon-tree.
- If G' is not a tree, it contains an undirected cycle C' of length at least 3. Because every edge in C' corresponds to a digon in G, C' induces two orientation-opposite directed cycles in G, each of length at least 3.

Case 2: *There exists an arc $(u,v)\in E$ such that $(v,u)\notin E$*. Since G is strongly connected, there exists a directed path P from v to u in G. Because $(v,u)\notin E$, the path P must contain at least two arcs. Concatenating the arc (u,v) with P yields a directed cycle C of length at least 3. □

3.1. Existence of Three Directed Cycles

In this subsection we will show that in any digraph $G=(V,E)$ such that the outdegree and the indegree of any node is at least three, there are at least three edge-disjoint directed cycles of size ≥3.

To obtain the first cycle, start a forward trip $T_1=(v_0,v_1,v_2,\cdots)$ from any node v_0. The strategy for building T_1 is to avoid returning to the node visited imme-

diately before the current node (*i.e.*, $v_{i+1} \neq v_{i-1}$). This is always possible because every node has an outdegree of at least 3. Because the total number of nodes in G is finite, T_1 cannot visit new nodes indefinitely and must eventually revisit a previously visited node. Let $v_k = v_j$ (with $j < k$) be the first repeated node. The subtrip $(v_j, v_{j+1}, \cdots, v_k)$ forms a simple directed cycle C_1. Since immediate backtracking was prohibited, $k - j \geq 3$, ensuring C_1 has size at least 3.

To obtain the second cycle, delete the arcs $E(C_1)$ from G. Because C_1 is a simple cycle, removing its arcs decreases both the indegree and outdegree of each node on C_1 by exactly 1. Thus, every node in $G \setminus E(C_1)$ retains an indegree and outdegree of at least 2. We then start a second trip T_2 in $G \setminus E(C_1)$ using the same non-backtracking rule ($v_{i+1} \neq v_{i-1}$), which remains possible since the outdegree is at least 2. By finiteness, T_2 must close a loop to form a simple directed cycle C_2. Avoiding immediate backtracking again guarantees that C_2 has size at least 3, and by construction, C_2 is arc-disjoint from C_1.

To get a third cycle start a trip from any node using the same strategy as before: *if possible do not return to the node visited right before the current node*. This time it is not always possible since C_1 and C_2 might have used two outgoing arcs of some nodes. Let $D \subseteq V$ be the set of those nodes.

Suppose at some point of our trip we do not yet have the third cycle and we are stuck at some node u, *i.e.*, all the outgoing arcs of u have been used before.

Partition the whole trip T into two parts T_1 and T_2 where T_1 is the subtrip up to the first visit to u and T_2 is the subtrip after that point on. Note that T_2 is a strongly-connected balanced subgraph. Since T_2 does not contain any directed cycles of size ≥3 then it is a digon-tree, based on Lemma 1 (see **Figure 3**).

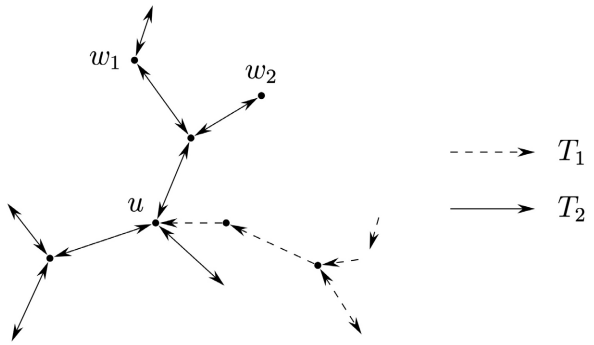

Figure 3. Partitioning the trip to T_1 and T_2.

Note that the only common node of T_1 and T_2 is u; otherwise, T would contain a cycle of size ≥3. Then any leaf v of the digon-tree T_2 is from D; otherwise, based on our strategy we wouldn't return from v to the node visited right before it (if u is a leaf of T_2 then $u \in D$, otherwise we would not be stuck at u). Thus, T_2 contains at least two nodes from D, and so at least two nodes from C_1. Let $w_1, w_2 \in V(C_1) \cap V(T_2)$ such that $int\left(w_1 \overset{T_2}{\rightsquigarrow} w_2\right) \cap V(C_1) = \varnothing$.

Then $Q_1 = w_1 \overset{C_1}{\rightsquigarrow} w_2 \overset{T_2}{\rightsquigarrow} w_1$ and $Q_2 = w_1 \overset{T_2}{\rightsquigarrow} w_2 \overset{C_1}{\rightsquigarrow} w_1$ are edge-disjoint cycles of size ≥3 (see Figure 4).

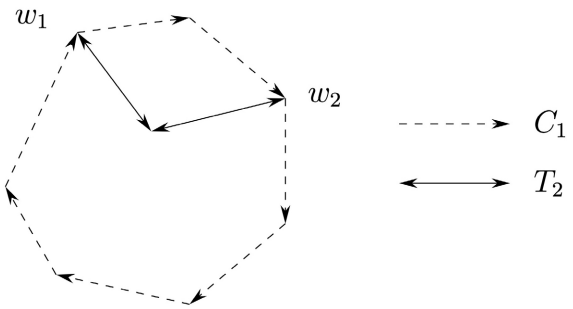

Figure 4. Getting the third cycle.

So we have three edge-disjoint directed cycles of size ≥3: Q_1, Q_2 and C_2.

3.2. Existence of the Fourth Directed Cycle

Let C_1, C_2 and C_3 be the first three edge-disjoint cycles and p be the number of their common nodes. We show the existence of the fourth cycle by *induction on* p.

- $p = 0$. The arguments to get the third cycle in subsection 3.1 are still valid in this case to get a fourth cycle. Here D is defined as the set of the nodes shared by exactly two of the three cycles. Note that any two nodes from D are on the same cycle C_i for some $i \in \{1,2,3\}$.
- Suppose $p = 1$, and x is the only common node. This time D is defined as the set of the nodes shared by two or three cycles. Three cases are possible:
  ◇ if x is balanced, *i.e.*, $\text{outdegree}(x) = \text{indegree}(x)$, then we can get a fourth cycle by starting a trip from any node $v \neq x$ and applying the arguments of subsection 3.1.
  ◇ if $\text{outdegree}(x) > \text{indegree}(x)$ then taking a forward trip from u will give a fourth cycle.
  ◇ if $\text{outdegree}(x) < \text{indegree}(x)$ then taking a backward trip from x and applying arguments dual to those in subsection 3.1 will give a fourth cycle.
- $p > 1$. Suppose a fourth cycle exists whenever C_1, C_2 and C_3 have less than p common nodes. Let's show it for the case of p common nodes. Let $U = \{u_1, \cdots, u_p\}$ be the set of the common nodes.

  Two cases are possible.

  1) While traversing C_2 and C_3 starting from u_1 we meet the common nodes in the same order as in C_1.

  2) We meet them in a different order.

  Case 1:

  Subcase 1.1: $int\left(u_i \overset{C_k}{\rightsquigarrow} u_{i+1}\right)$ does not have any intersection with $int\left(u_j \overset{C_l}{\rightsquigarrow} u_{j+1}\right)$

for $i \neq j$, $k \neq l$. Then let $int(u_i \rightsquigarrow u_{i+1})$ denote the set of internal nodes of $u_i \overset{C_k}{\rightsquigarrow} u_{i+1}, k = 1,2,3$.

Start a forward trip T_i from a node $w \in int(u_i \rightsquigarrow u_{i+1})$.

- Suppose we get stuck in $int(u_i \rightsquigarrow u_{i+1})$ before hitting any node outside it. If we get a cycle of size ≥3 inside $int(u_i \rightsquigarrow u_{i+1})$, then we have a fourth cycle. Suppose we do not have a cycle and get stuck in node $v \in int(u_i \rightsquigarrow u_{i+1})$, by getting a digon at the end of the trip. Then v is visited more than once, and the subtrip of T_i after visiting v the first time is a digon-tree D (could be just a digon like in Figure 5). Note that each leaf of the digon-tree belongs to two of the three cycles, based on our strategy of avoiding digons in the trips. Take two nodes w_1 and w_2 on D such that (i) each of them belongs to two of the three cycles, (ii) no other node on the D-path connecting them belongs to two cycles. Then one of the three cycles, say C_1, contains both w_1 and w_2. If there are other nodes on the D-path connecting w_1 and w_2 that belong to C_1, then we can replace w_1 and w_2 with other nodes on D that belong to C_1 and the D-path connecting them does not have nodes from C_1. Suppose we encounter w_1 before w_2 while traversing cycle $\underset{D}{\overset{C_1}{}} C_1$ from u_i to u_{i+1}. Then we can replace C_1 with cycle $Q_1 = w_1 \rightsquigarrow w_2 \rightsquigarrow w_1$ which has size ≥3 (see Figure 5).

Next, we show that Q_1, C_2, C_3 share no common nodes. Any node on the path $w_1 \overset{C_1}{\rightsquigarrow} w_2$ belongs to C_1 and to at most one of C_2 and C_3; thus, replacing C_1 with Q_1 cannot create a node common to all three cycles Q_1, C_2, C_3. Furthermore, by the construction of D (part (ii) above), no interior node of D belongs to more than one of C_1, C_2, C_3. More specifically, each interior node of D (if any exist) belongs to exactly one of C_2 and C_3. Consequently, after replacing C_1 with Q_1, each interior node of D will belong to Q_1 and to exactly one of C_2 and C_3. In summary, Q_1, C_2, C_3 have no common nodes, allowing us to apply the induction hypothesis.

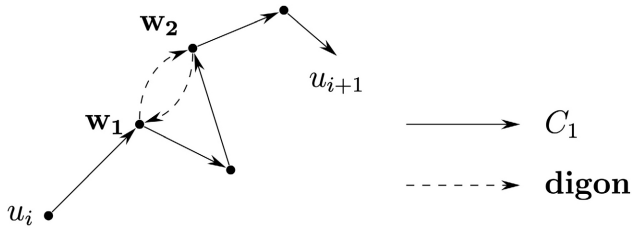

Figure 5. Example of Subcase 1.1.

- Suppose we hit a node outside $int(u_i \rightsquigarrow u_{i+1})$, and the first node of $V(C_1) \cup V(C_2) \cup V(C_3) \setminus int(u_i \rightsquigarrow u_{i+1})$ hit by the trip is $v \neq u_{i+1}$. This is a situation similar to subcase 1.2 of Section 2: trip T_i will give a shortcut for one of the three cycles. Let w_i' be the last node from $int(u_i \rightsquigarrow u_{i+1})$ visited before v in trip T_i. Then the path $w_i' \overset{T_i}{\rightsquigarrow} v$ can serve as a shortcut for one of the original

cycles, say C_1. The new cycle $C_1' = w_i' \overset{T_i}{\rightsquigarrow} v \overset{C_1}{\rightsquigarrow} w_i'$ does not contain u_{i+1}.

By construction, the path $v \overset{C_1}{\rightsquigarrow} w_i'$ contains the exact same set of common nodes as before, with the exception of u_{i+1}, which is excluded because $u_{i+1} \notin V(C_1')$. Furthermore, the interior of the path $w_i' \overset{T_i}{\rightsquigarrow} v$ contains no nodes from $V(C_1) \cup V(C_2) \cup V(C_3)$. Consequently, the number of nodes common to C_1', C_2, and C_3 is strictly less than p (specifically, $p-1$), allowing us to obtain a fourth cycle by induction.

- Suppose we hit a node outside $int(u_i \rightsquigarrow u_{i+1})$, and the first node of $V(C_1) \cup V(C_2) \cup V(C_3) \setminus int(u_i \rightsquigarrow u_{i+1})$ hit by the trip is u_{i+1}. This case is handled like subcase 1.3 of Section 2. If the forward trip T_i hits u_{i+1} and the backward trip S_i hits u_i for all $i \in 1, \cdots, p$, then the union of all forward and backward trips will give a fourth cycle.

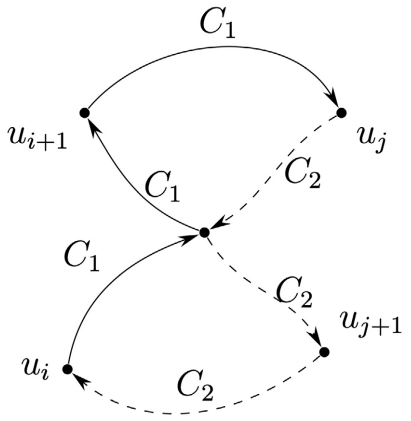

Figure 6. Example of Subcase 1.2 in the proof for the fourth cycle.

Subcase 1.2: $int\left(u_i \overset{C_k}{\rightsquigarrow} u_{i+1}\right)$ has intersection with $int\left(u_j \overset{C_l}{\rightsquigarrow} u_{j+1}\right)$ for $i \neq j$, $k \neq l$. Let $k=1$, $l=2$, and w is an intersection node. Then define two new cycles: $Q_1 = w \overset{C_1}{\rightsquigarrow} u_{i+1} \overset{C_1}{\rightsquigarrow} u_j \overset{C_2}{\rightsquigarrow} w$, $Q_2 = w \overset{C_2}{\rightsquigarrow} u_{j+1} \overset{C_2}{\rightsquigarrow} u_i \overset{C_1}{\rightsquigarrow} w$ (see **Figure 6**).

Number of common nodes of cycles Q_1, Q_2, C_3 is less than the number of common nodes of cycles C_1, C_2, C_3. Particularly, $u_i, u_{i+1}, u_j, u_{j+1}$ are not common nodes now.

On the other hand, no new common nodes are created. For instance, if the path $u_i \overset{C_1}{\rightsquigarrow} w$ in Q_2 intersects the path $u_j \overset{C_2}{\rightsquigarrow} w$ in Q_1 as well as C_3, then any such intersection would also be a common node of C_1, C_2, and C_3. Furthermore, since C_1 is a simple directed cycle, the path $u_i \overset{C_1}{\rightsquigarrow} w$ in Q_2 cannot share any vertices with the paths $w \overset{C_1}{\rightsquigarrow} u_{i+1}$ (except for w) or with $u_{i+1} \overset{C_1}{\rightsquigarrow} u_j$ in Q_1. The arguments for the remaining pairs of paths, one from Q_1 and the other from Q_2, are identical.

Even in the case when u_j coincides with u_{i+1} and u_i coincides with u_{j+1}, the number of common nodes will go down by two.

Thus, we can apply induction in this case.

Case 2: u_i's are encountered in different order while traversing the three cycles. Note that in this case the number of common nodes is at least three.

Subcase 2.1: The order of u_i's is σ or σ^{-1} (the opposite order of σ) for the cycles. Let C_1 and C_2 have the same order σ while C_3 has order σ^{-1}.

- If $int\left(u_i \overset{C_1}{\rightsquigarrow} u_{i+1}\right)$ has intersection with $int\left(u_j \overset{C_2}{\rightsquigarrow} u_{j+1}\right)$ for $i \neq j$, then we can apply the idea of Subcase 1.2.
- Otherwise we can interchange $u_i \overset{C_1}{\rightsquigarrow} u_{i+1}$ and $u_i \overset{C_2}{\rightsquigarrow} u_{i+1}$ for any i. Let $B_{i1} = u_i \overset{C_1}{\rightsquigarrow} u_{i+1} \overset{C_3}{\rightsquigarrow} u_i$ and $B_{i2} = u_i \overset{C_2}{\rightsquigarrow} u_{i+1} \overset{C_3}{\rightsquigarrow} u_i$. B_{i1} and B_{i2} are strongly-connected balanced digraphs. For given i, B_{i1} and B_{i2} cannot be both digon-trees since $u_i \overset{C_1}{\rightsquigarrow} u_{i+1}$ and $u_i \overset{C_2}{\rightsquigarrow} u_{i+1}$ are different. For every i, interchange $u_i \overset{C_1}{\rightsquigarrow} u_{i+1}$ and $u_i \overset{C_2}{\rightsquigarrow} u_{i+1}$ so that B_{i1} contains a directed cycle Q_i of size ≥3. We have at least three edge-disjoint Q_i's obtained this way because the number of common nodes of C_1, C_2, C_3 is at least three in this case. Since we also have C_2, then the number of directed cycles of size ≥3 is at least four.

Subcase 2.2: Two of the three cycles have neither the same nor the opposite directions. Let C_1 and C_2 be those two cycles. Let $u_1, u_2, \cdots, u_p$ be the order we encounter the common nodes on cycle C_1. Then for some i, the node following u_i on cycle C_2 is neither u_{i+1} nor u_{i-1}. Suppose that node is u_j (see Figure 7).

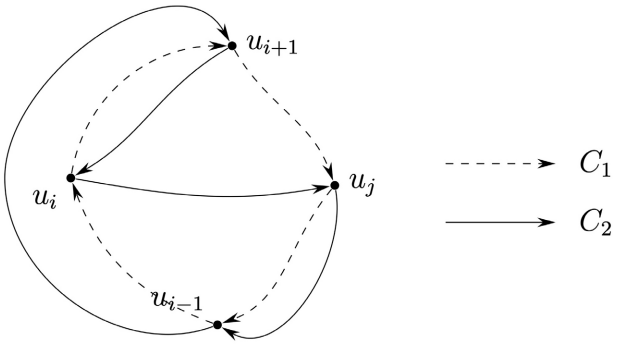

Figure 7. Example of Subcase 2.2.

Let $Q_1 = u_i \overset{C_2}{\rightsquigarrow} u_j \overset{C_1}{\rightsquigarrow} u_i$ and $Q_2 = u_i \overset{C_1}{\rightsquigarrow} u_j \overset{C_2}{\rightsquigarrow} u_i$. Q_1 and Q_2 are strongly-connected balanced graphs. Since $u_{i-1} \in \left\{u_j \overset{C_1}{\rightsquigarrow} u_i\right\} \setminus \left\{u_i \overset{C_2}{\rightsquigarrow} u_j\right\}$, Q_1 cannot be a digon-tree and contains a directed cycle Q_1' of size ≥3. Similarly, since $u_{i-1} \in \left\{u_j \overset{C_2}{\rightsquigarrow} u_i\right\} \setminus \left\{u_i \overset{C_1}{\rightsquigarrow} u_j\right\}$, Q_2 cannot be a digon-tree and contains a directed

cycle Q_2' of size ≥ 3.

Note that the set U' of common nodes of Q_1', Q_2', C_3 is a subset of U. But $u_{i+1} \notin V(Q_1')$. Thus, $u_{i+1} \notin U'$, $|U'| < |U| = p$ and taking Q_1', Q_2' and C_3 as the first three cycles we will get a fourth cycle by induction.

3.3. Worst-Case Example

Note that in a worst-case scenario four cycles is the best we can achieve. To illustrate that, consider the following example. Let $G = (V, E)$ be a complete digraph on a 4-node set $V = \{x, y, z, u\}$. In this graph we can find at most four edge-disjoint directed cycles of size ≥3: $C_1 = x \to y \to u \to x$, $C_2 = y \to x \to z \to y$, $C_3 = u \to z \to x \to u$, $C_4 = z \to u \to y \to z$. This is not surprising since the total number of edges is only 12 in this example.

4. Approximation Algorithms for *k*-Vertex-Connectivity Problems

Connectivity is a fundamental problem to the study of graphs and graph algorithms. In this paper we consider uniform connectivity problems in directed graphs. A strongly connected graph is said to be k-vertex-connected (or simply k-connected) if the deletion of at most $k-1$ vertices still leaves the graph strongly connected. A set of paths from vertice u to vertice v is said to be openly disjoint if they do not share any internal vertices. This definition is extended to the case when u or v is replaced by a set of vertices as follows: a set of k paths from u to a set of vertices R (from R to v) with $|R| = k$ is openly disjoint if the paths are all vertex-disjoint, except for sharing the endpoint u (or v), and each of the k paths ends (starts) at a distinct vertex of R. If $u \in R$ ($v \in R$) then we find $k-1$ openly disjoint paths from u to the vertices in $R-u$ (from the vertices in $R-u$ to v).

The k-vertex-connectivity problem in directed graphs is the following. The input is an integer k, a k-connected digraph $G = (V, E)$ and a weight function w on the edges of G. The goal is to find a minimum-weight k-connected spanning subgraph of G.

The practical motivation to study the problem is the following. Let E be a collection of all possible links for a proposed a telecommunications network. The weight w_e of edge $e \in E$ represents the cost of building and including e in the network. A minimum spanning tree in $G = (V, E)$ is the cheapest network that will allow all the sites in V to communicate. But such a network is highly susceptible to failures, since it cannot survive a failure of a single link or site. For more reliable communication, higher connectivities of the sites are necessary. A k-vertex-connected allows communication between functioning sites even after as many as $k-1$ sites have failed.

The k-vertex-connectivity problem is known to be NP-hard. Approximation algorithms have been developed for many related NP-hard connectivity problems [7]. We will discuss those problems where triangle inequality holds for edge

weights.

For the k-vertex-connectivity problem in undirected graphs, Khuller and Raghavachari [8] gave an approximation algorithm which has an approximation factor of $2+2(k-1)/n$. Kortsarz and Nutov [9] obtained a $2+2k/n$-approximation algorithm for the k-vertex-connectivity problem in directed graphs. In this paper, we combine some ideas from [8] with our results from sections 2 and 3 to obtain approximation algorithms for the k-vertex-connectivity problem in directed graphs when $k=2$ and $k=3$.

4.1. Basic Technique

The main algorithm given in this subsection is similar to the one given in [8] for the undirected case.

First consider the following related problem. The input is a directed graph G with nonnegative weights on the edges, a root vertex r and an integer p. The goal is to find a minimum-weight directed subgraph H of G, such that for each vertex v there are p openly disjoint paths from r to v (alternately, from v to r for each v) in the subgraph H. Frank and Tardos [10] gave an algorithm which solves this problem in polynomial time. We will use that algorithm as a subroutine for solving our problem.

In the algorithm for solving the k-vertex-connectivity problem, using a set of root vertices R plays an important role. Below we will discuss two subroutines that use root vertices. Those are called ROOT-VERTICES-IN (G,p,R) and ROOT-VERTICES-OUT (G,p,R). The input of both subroutines is a k-vertex-connected directed graph $G=(V,E)$, a nonnegative weight function w defined on the edges, and a set R of p vertices. The output of ROOT-VERTICES-IN (G,p,R) (ROOT-VERTICES-OUT (G,p,R)) is an approximate minimum-weight subgraph of G in which there are p openly disjoint paths from any vertex v to R (from R to any vertex v). Note that v could be in R, in which case we find $p-1$ paths from v to the remaining vertices in R (from remaining vertices of R to v).

The algorithm ROOT-VERTICES-OUT (G,p,R) works as follows. Augment G by adding a new vertex r and add p new directed edges of weight 0 from r to each vertex in R. Use the algorithm of [10] on this graph with r as the root and find a minimum-weight subgraph H_{out} with p openly disjoint paths from r to any vertex of G. Since these disjoint paths go through the vertices of R, for any vertex v in G, there are p openly disjoint paths from R to v in H_{out}.

The algorithm ROOT-VERTICES-IN (G,p,R) works similarly. Augment G by adding a new vertex r and add p new directed edges of weight 0 from each vertex in R to r. Use the algorithm of [10] on this graph with r as the root and find a minimum-weight subgraph H_{in} with p openly disjoint paths from any vertex of G to r. Since these disjoint paths go through the vertices of R, for any vertex v in G, there are p openly disjoint paths from v to R in

H_{in}.

Let S_{in} and S_{out} be the subgraphs of G obtained from correspondingly H_{in} and H_{out} by deleting r and its adjacent edges.

Lemma 2 *There are p openly disjoint paths from R to v in S_{out}. For any $v \in R$, there are $p-1$ openly disjoint paths from $R-v$ to v in S_{out}. There are p openly disjoint paths from v to R in S_{in}. For any $v \in R$, there are $p-1$ openly disjoint paths from v to $R-v$ in S_{in}.*

Lemma 3 *The weight of S_{in} (S_{out}) is not more than the weight of a minimum-weight p-vertex-connected spanning subgraph of G.*

Proof: We will give the proof for S_{out}; the proof for S_{in} is similar. Let T be a minimum-weight p-vertex-connected spanning subgraph of G. Add the zero-weight edges from r to the vertices of R to obtain a new subgraph T'. T' has p openly disjoint paths from r to each vertex of G. Since the algorithm of [10] returns a minimum-weight solution H_{out} satisfying that property, we have that

$$\text{weight}(S_{out}) = \text{weight}(H_{out}) \leq \text{weight}(T') = \text{weight}(T). \qquad \square$$

4.2. The Main Algorithm

The algorithm for the k-vertex-connectivity problem first finds a subset R of k vertices such that the complete graph induced on R is relatively low-weight. Then it calls ROOT-VERTICES-IN(G, p, R) and ROOT-VERTICES-OUT(G, p, R) to obtain subgraphs S_{in} and S_{out} of G. The algorithm returns $S_{in} \cup S_{out} \cup K_R$ as its output, where K_R is the complete subgraph on R.

First, we show that the output of the algorithm is k-vertex-connected, regardless how we find R.

Lemma 4 *The subgraph $S_{in} \cup S_{out} \cup K_R$ is k-vertex-connected.*

Proof: The proof by contradiction. Assume that the graph contains a vertex cut C with $|C| < k$, *i.e.*, the removal of C breaks G into strongly-connected components $C_1, C_2, \cdots, C_l$, where $l \geq 2$. Since K_R is a complete subgraph of R, all vertices of R remaining after the removal of C belong to C_i for some i. Consider a vertex $v \in C_j$ for some $j \neq i$. By lemma 2 there must be k openly disjoint paths from v to R and k openly disjoint paths from R to v in $S_{in} \cup S_{out} \cup K_R$. Thus, the removal of less than k nodes will leave v strongly connected to the remaining nodes of R. This is a contradiction since $v \notin C_i$. $\square$

The key of getting a good approximation factor is to find a low-weight K_R. Next, we will show how to achieve that for $k = 2, 3$ by using our results in sections 2 and 3.

4.3. Approximation Factor for $k = 2$

For $k = 2$, the subset R is chosen as the nodes of the minimum-weight digon in G. It can be found in $O(n^2)$ time since there are $\binom{n}{2}$ possible digons.

Theorem 1 *The total weight of* $S_{in} \cup S_{out} \cup K_R$ *is at most* $2\frac{1}{3}$ *times the optimum.*

Proof: Let T be a minimum-weight 2-vertex-connected spanning subgraph of G. Based on the result of section 2, T has at least three edge-disjoint directed cycles: C_1, C_2, C_3. Since each digon is itself a cycle,

$$\text{weight}(T) \geq \text{weight}(C_1) + \text{weight}(C_2) + \text{weight}(C_3) \geq 3 \cdot \text{weight}(R).$$

Based on lemma 3,

$$\text{weight}(S_{in}) + \text{weight}(S_{out}) + \text{weight}(K_R) \leq 2\frac{1}{3} \cdot \text{weight}(T). \qquad \square$$

4.4. Approximation Factor for $k = 3$

For $k = 3$, the subset R is chosen as the nodes of the minimum-weight directed cycle C of size 3 in G. It can be found in $O(n^3)$ time since there are $2 \cdot \binom{n}{3}$ possible directed cycles of size 3.

Theorem 2 *The total weight of* $S_{in} \cup S_{out} \cup K_R$ *is at most* $2\frac{3}{4}$ *times the optimum.*

Proof: Let T be a minimum-weight 3-vertex-connected spanning subgraph of G. Based on the result of section 3, T has at least four directed cycles of size ≥3: C_1, C_2, C_3, C_4. For any $i = 1,2,3,4$, let C_i' be a directed cycle of size 3 obtained from C_i by taking shortcuts if necessary. Based on the triangle inequality, $\text{weight}(C_i') \leq \text{weight}(C_i)$, for $i = 1,2,3,4$. Therefore, since C is the minimum-weight directed cycle of size 3,

$$\text{weight}(C) \leq \text{weight}(C_i') \leq \text{weight}(C_i), \text{ for } i = 1,2,3,4.$$

Thus,

$$\begin{aligned} &\text{weight}(T) \geq \text{weight}(C_1) + \text{weight}(C_2) + \text{weight}(C_3) + \text{weight}(C_4) \\ &\geq 4 \cdot \text{weight}(C). \end{aligned} \tag{1}$$

Let C' be the cycle of the opposite orientation to C, that is, if $C = v_1 \to v_2 \to v_3 \to v_1$ then $C' = v_1 \to v_3 \to v_2 \to v_1$. Based on the triangle inequality,

$$\text{weight}(v_1 \to v_3) \leq \text{weight}(v_1 \to v_2) + \text{weight}(v_2 \to v_3)$$

$$\text{weight}(v_3 \to v_2) \leq \text{weight}(v_3 \to v_1) + \text{weight}(v_1 \to v_2)$$

$$\text{weight}(v_2 \to v_1) \leq \text{weight}(v_2 \to v_3) + \text{weight}(v_3 \to v_1)$$

Adding up the left and right hand sides of the above three inequalities will result in

$$\text{weight}(C') \leq 2 \cdot \text{weight}(C). \tag{2}$$

That is, the weight of the cycle of the opposite orientation to C is no more

than twice the weight of C. Based on (1) and (2),

$$\text{weight}(K_R) = \text{weight}(C) + \text{weight}(C') \leq 3 \cdot \text{weight}(C) \leq 3/4 \cdot \text{weight}(T).$$

Using also the results of lemma 3,

$$\text{weight}(S_{in}) + \text{weight}(S_{out}) + \text{weight}(K_R) \leq 2\frac{3}{4} \cdot \text{weight}(T). \qquad \square$$

4.5. Complexity Analysis

The total running time of the algorithm is dominated by finding the set R and executing the two calls to the Frank–Tardos subroutine, ROOT-VERTICES-IN (G, p, R) and ROOT-VERTICES-OUT (G, p, R). For an input digraph $G = (V, E)$ with $n = |V|$ and $m = |E|$:

- **Root Set Identification:** For $k = 2$, finding the minimum-weight digon requires $O(n^2)$ time by inspecting all $\binom{n}{2}$ vertex pairs. For $k = 3$, finding the minimum-weight directed 3-cycle takes $O(n^3)$ time by inspecting all $2\binom{n}{3}$ candidate 3-cycles.
- **Subroutine Executions:** For constant $p = k \in \{2, 3\}$, the subroutines ROOT-VERTICES-IN or ROOT-VERTICES-OUT have $O(n^2 m)$ running time [10].

Since $m \geq n$ in strongly connected digraphs, the $O(n^2 m)$ subroutine phase bounds the total execution time. Thus, the main algorithm achieves an overall time complexity of $O(n^2 m)$ for both $k = 2$ and $k = 3$.

5. Future Directions

Getting similar results in digraphs of minimum degree k for general k would be interesting. That result could be used to get approximation algorithms for general k -vertex-connectivity problems.

Furthermore, it would be natural to extend these cycle-packing techniques to more refined network resilience metrics. In particular, investigating whether degree-constrained cycle packings can yield structural bounds for super connectivity [11] or approximation guarantees for k -robustness in networks [12] presents a promising direction for future research.

Another interesting question is whether the cycle packing properties could be used to obtain approximation algorithms for other network design problems.

Conflicts of Interest

The author declares no conflicts of interest regarding the publication of this paper.

References

[1] Balister, P. (2003) Packing Digraphs with Directed Closed Trails. *Combinatorics, Probability and Computing*, **12**, 1-15. https://doi.org/10.1017/s0963548302005461

[2] Seymour, P.D. (1995) Packing Directed Circuits Fractionally. *Combinatorica*, **15**, 281-288. https://doi.org/10.1007/bf01200760

[3] Conlon, D., Fox, J. and Sudakov, B. (2014) Cycle Packing. *Random Structures & Algorithms*, **45**, 608-626. https://doi.org/10.1002/rsa.20574

[4] Salavatipour, M.R. and Verstraëte, J. (2005) Disjoint Cycles in Directed Graphs. *Proceedings of the Sixteenth Annual ACM-SIAM Symposium on Discrete Algorithms* (*SODA*), Vancouver, 23-25 January 2005, 1152-1151.

[5] Shang, Y. (2010) On the Degree Sequence of Random Geometric Digraphs. *Applied Mathematical Sciences*, **4**, 2001-2012. https://www.m-hikari.com/ams/ams-2010/ams-41-44-2010/index.html

[6] Alon, N., McDiarmid, K. and Molloy M. (1996) Edge-Disjoint Cycles in Regular Directed Graphs. *Journal of Graph Theory*, **22**, 231-237. https://onlinelibrary.wiley.com/doi/abs/10.1002/(SICI)1097-0118(199607)22:3%3C231::AID-JGT3%3E3.0.CO;2-N?utm_source=gemini

[7] Gupta, A. and Könemann, J. (2011) Approximation Algorithms for Network Design: A Survey. *Surveys in Operations Research and Management Science*, **16**, 3-20. https://doi.org/10.1016/j.sorms.2010.06.001

[8] Khuller, S. and Raghavachari, B. (1996) Improved Approximation Algorithms for Uniform Connectivity Problems. *Journal of Algorithms*, **21**, 434-450. https://doi.org/10.1006/jagm.1996.0052

[9] Kortsarz, G. and Nutov, Z. (2003) Approximating Node Connectivity Problems via Set Covers. *Algorithmica*, **37**, 75-92. https://doi.org/10.1007/s00453-003-1027-4

[10] Frank, A. and Tardos, É. (1989) An Application of Submodular Flows. *Linear Algebra and its Applications*, **114**, 329-348. https://doi.org/10.1016/0024-3795(89)90469-2

[11] Shang, Y. (2019) Super Connectivity of Erdős-Rényi Graphs. *Mathematics*, **7**, Article 267. https://doi.org/10.3390/math7030267

[12] Shang, Y. (2023) On Connectivity and Robustness of Random Graphs with Inhomogeneity. *Journal of Applied Probability*, **60**, 284-294. https://www.cambridge.org/core/journals/journal-of-applied-probability/article/abs/on-connectivity-and-robustness-of-random-graphs-with-inhomogeneity/63E67543B2CC627844B0E62DBD5599BA?utm_source=gemini