Approximating the Minimum Strongly Connected Subgraph via a Matching Lower Bound.

Adrian Vetta*

Abstract

We present a $\frac{3}{2}$-approximation algorithm for the problem of finding a minimum strongly connected spanning subgraph in a given directed graph. As a corollary we obtain a $\frac{3}{2}$-approximation algorithm for the more general minimum equivalent digraph problem. The performance of our algorithm is measured against a lower bound obtained from a simple matching problem. The performance guarantee is optimal with respect to the lower bound.

1 Introduction

A directed graph $G = (V, E)$ is *strongly connected* if, for every pair of vertices $u, v \in V$, there is a directed path from u to v and vice versa. Given a strongly connected graph, the problem of finding a strongly connected spanning subgraph containing the fewest possible arcs is NP-hard. We denote this problem by SCS. A slightly more general problem is the minimum equivalent digraph problem, denoted MED. Here we are given a directed graph G and we wish to find the smallest subgraph that maintains the reachability relations between all pairs of vertices in G. This is a long studied problem, with Moyles and Thompson [4] showing that MED decomposes into two parts; a solution to the strongly connected subgraph problem on each strongly connected component of G and a solution of MED on an induced acyclic graph. The latter problem can be solved exactly in polynomial time [1]. It follows that an α-approximation algorithm for the strongly connected subgraph problem can easily be converted into an α-approximation algorithm for the minimum equivalent digraph problem. The current best approximation algorithm for SCS was given by Khuller, Raghavachari and Young ([2],[3]). Their algorithm has an approximation factor of just over 1.617. We improve this factor to $\frac{3}{2}$.

2 The Lower Bound

The performance of our algorithm will be measured with respect to a lower bound from matching theory. The motivation for this lower bound was an undirected version applied in [5]. There it proved useful for the analogous problem in undirected graphs, the minimum 2-edge connected subgraph problem (as well as the minimum 2-vertex connected subgraph problem). Consider the problem of finding the minimum sized subgraph in which every node has both in-degree and out-degree at least one. Call this problem D1. A optimal solution to D1 can be computed in polynomial time via a reduction to the undirected maximum matching problem.

LEMMA 2.1. *D1 gives a lower bound on the size of the optimal solutions to SCS.*

Proof. Every node must be incident to both an in-arc and an out-arc in any strongly connected subgraph. □

Our algorithm will take an optimal solution to D1 and alter it in order to obtain a strongly connected subgraph. In the sections that follow we will formalise how to transform our solution to D1 into a solution to SCS. In addition, we will show that this transformation process gives a strongly connected subgraph that is at most $\frac{3}{2}$ times the size of the optimum solution for SCS.

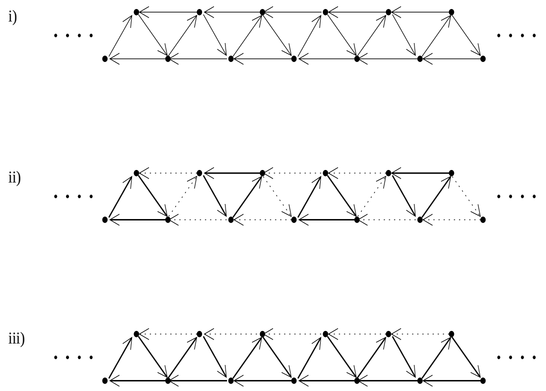

Figure 1: A tight example.

The graph in Figure 1i) shows that this approximation guarantee is the best we will be able to obtain using just this lower bound. The solution to D1, shown in Figure 1ii), is a perfect cycle cover and hence contains n arcs. There is a $\frac{3}{2}$ gap, though, between this solution

*Dept of Mathematics, M.I.T.,avetta@math.mit.edu

and the optimal solution to SCS, which is illustrated in Figure 1iii).

3 The Tools

3.1 The Cycle Contraction Method

First we present a simple cycle-contraction algorithm for solving SCS. The methods utilised by this algorithm will be applied in transforming our D1 solution into a strongly connected subgraph. Find a cycle C in G that contains a specified vertex r, called the *root vertex*. We will take C to form part of our final solution. Next consider $G' = G/C$, the graph formed by the contraction of the cycle C. Let the supervertex formed by the contraction of C be our root vertex r' in G'. Recurse until the graph contains a single vertex. At this point it is clear that the cycles found in each iteration together form a strongly connected subgraph in G.

We now show that this cycle contraction algorithm has a performance guarantee of 2. Given a contracted cycle C, charge the arcs in the cycle to the vertices of the cycle as follows. The root vertex r is not charged. One of the vertices neigbouring r in C, say v_2, is charged the two arcs adjacent to it in the cycle. The other vertices in the cycle (if C contains more than two vertices) are charged for one adjacent arc each. Notice that a vertex receives a charge only when it is contracted into the root vertex. Since every arc in our solution is charged to some vertex and every vertex is charged at most 2 arcs, we see that our solution contains at most $2n - 2$ arcs. Now n is a trivial lower bound on any strongly connected subgraph and hence this approach does indeed give a 2-approximation algorithm.

3.2 The Component Graph

We now return to task of making use of the lower bound given by D1. We have the minimum sized subgraph in which every vertex is the tail of some arc and the head of another. Note that the solution to this problem forms a subgraph that consists of disjoint components. In turn, these components are either strongly connected or they consist of strongly connected sub-components connected together by directed paths. An example of this latter type is shown in Figure 2i).

These components we will break apart as follows. We separate the strongly connected sub-components. We also remove the two end arcs of each path. These arcs, which we call the "free arcs" associated with a path, will not be forgotten, though, as they are included in our lower bound. This method is illustrated in Figure 2ii). After this process is complete we have a collection of disjoint sub-components. For clarity we will subsequently refer to these sub-components as components. These components are either paths or are strongly connected. Note that the paths may consist of just one vertex.

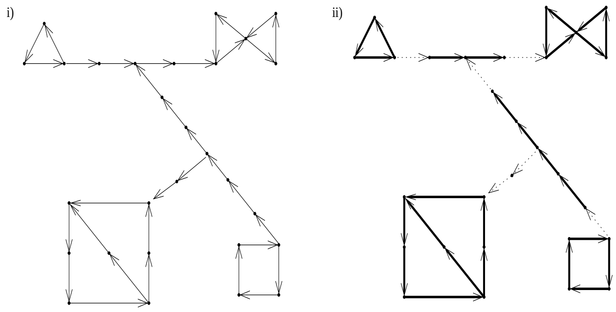

Figure 2: Finding components.

To motivate our method, assume for a moment that each of the components is strongly connected and contains at least four arcs. Now consider contracting these components so that they form the nodes of an induced graph, $\mathcal{G}$, called the *component graph*. Note that, for clarity, we refer to nodes in $\mathcal{G}$ and vertices in G. Let there be p such components and hence p nodes in our induced graph. Performing the cycle contraction algorithm outlined above gives a strongly connected subgraph on $\mathcal{G}$ containing at most $2p - 2$ arcs. Hence, at most two arcs are charged to each node or component. These arcs together with the arcs in the original strongly connected components form a strongly connected subgraph on G. Since two additional arcs are associated with each component and, in turn, each component is of size at least 4 this represents a $\frac{3}{2}$-optimal solution.

This idea forms the basis of our algorithm. Apply the cycle contraction algorithm on the component graph $\mathcal{G}$. We have seen that this approach is appropriate when the components are strongly connected and contain at least 4 arcs. The components, though, may also represent 2-cycles (digons), 3-cycles or paths. In such circumstances we need to be more careful. For example, charging two arcs to a digon results in a local blow-up factor of 2, which is unacceptable. We also need to ensure that we induce a strongly connected subgraph when path components are contracted via the cycle-contraction algorithm. These issues will be dealt with in subsequent sections.

3.3 Incorporating a Depth First Search Tree

Now we consider applying the cycle contraction algorithm to the general case where the components may be digons, 3-cycles or paths. In order to apply this method efficiently we will grow a depth first search tree $\mathcal{T}$ on the component graph $\mathcal{G}$. The tree will help us discover appropriate cycles in $\mathcal{G}$. In addition, the special structure provided by depth first search trees will help simplify the proofs concerning the performance of the algorithm.

We incorporate the tree as follows. We grow our tree from a node (component) r, henceforth termed the *root node* or *root component*. The tree, $\mathcal{T}$, will be an out-tree, a spanning tree with the property that there is a directed path from r to every other node. The tree is built via a depth first search (DFS) approach. Let the root node r be labelled v_1. Label the other vertices $v_2, v_3, \ldots, v_p$ according to the order in which they were added to $\mathcal{T}$ by the the DFS method. An important property of DFS trees is that all non-tree arcs (v_i, v_j) satisfy $j < i$, unless i lies on the directed path from r to j in the tree.

Since the nodes in $\mathcal{G}$ may contain several vertices we need to clarify exactly how to build a DFS tree on the component graph. As we grow the tree we order the vertices within each component. We then examine the vertices for new tree arcs according to that ordering. We use the notation $x \succ y$ when a vertex x is given priority over a vertex y.

1. Suppose that in growing the DFS tree we reach a component that represents a directed path. Let the path consist of the vertices $v_k, v_{k-1}, \ldots v_0$ with arcs (v_{i+1}, v_i). Use the ordering $v_0 \succ v_1 \succ \cdots \succ v_k$.
2. If the component represents a digon on the vertices v_0 and v_1, where v_0 is the vertex incident to the tree arc, use the ordering $v_1 \succ v_0$.
3. If the component represents a 3-cycle, $\{(v_0, v_1), (v_1, v_2), (v_2, v_0)\}$, where v_0 is the vertex incident to the tree arc, use the ordering $v_0 \succ v_2 \succ v_1$.
4. For a component representing a strongly connected component of size at least 4, we may order the vertices in any manner.

Given $\mathcal{T}$ and a component $C \in \mathcal{G}$, let T_C (or just T if there is no confusion) be the subtree of $\mathcal{T}$ rooted at C. If C contains the vertices $v_1, v_2, \ldots, v_k$ then $T_i \subseteq T_C$ is the subtree with root vertex $v_i \in C$. Denote by C_i the child components of v_i in T_i. There may be several such C_i for each v_i. In such circumstances, the context will make it apparent to which C_i we are referring.

We will use the following labels. We call an arc a *down arc* with respect to a component C, if it is directed from $G - T_C$ to T_C. We call an arc a *back arc* with respect to a component C, if it is directed in the opposite direction.

We call the first non-root component C, discovered in growing $\mathcal{T}$, the *target component*. Our aim will be to apply a version of the cycle contraction algorithm in which we find and contract a cycle through the root and the target component, and then recurse. This method will be described in detail in the Section 4.

3.4 Simplifying the Problem

We now show that it is sufficient to consider a smaller set of graphs. An approximation algorithm for these graphs will lead to an approximation algorithm for the general class of graphs. We begin with a decomposition lemma. A vertex v is a *cut vertex* if there are two components C_1 and C_2 such that for every pair of vertices, $u_1 \in C_1$ and $u_2 \in C_2$, each directed path from u_1 to u_2, and vice versa, passes through v.

LEMMA 3.1. *It is sufficient to consider only graphs that contain no cut vertices.*

Proof. Suppose G contains a cut vertex v. Then every strongly connected subgraph of G is the arc-disjoint union of a strongly connected subgraph on $C_1 \cup \{v\}$ and a strongly connected subgraph on $C_2 \cup \{v\}$. Hence we can decompose the problem into two and work on the pieces. Combining α-approximations for the pieces gives an α-approximation for G. $\square$

Let a *digon* be a directed cycle consisting of just two arcs, $a = (u_1, u_2)$ and its *complement* $\bar{a} = (u_2, u_1)$. We have the following lemma.

LEMMA 3.2. *Let G be a strongly connected graph containing at least three vertices. Then if G contains no cut vertices there exists a minimum strongly connected subgraph which does not contain any digons.*

Proof. Take a minimum strongly connected subgraph, S, containing the fewest possible number of digons. Suppose it contains a digon $a = (u_1, u_2)$ and $\bar{a} = (u_2, u_1)$. Notice there is no directed path from u_1 to u_2 in $S - \{a\}$. Otherwise a is superfluous and S is not minimal. Similarly there is no directed path from u_2 to u_1 in $S - \{\bar{a}\}$. So, consider an out-tree, T_1, rooted at u_1 in $S - \{a\}$. Clearly, $u_2 \notin T_1$. Similarly, $u_1 \notin T_2$, where T_2 is an out-tree rooted at u_2 in $S - \{\bar{a}\}$.

Now $A = T_1 \cap T_2$ is empty. Otherwise A induces a cut $\delta(A)$ in which the only arcs in $S \cap \delta(A)$ are all directed into A. Hence S is not a strongly connected arc set. Similarly $V = T_1 \cup T_2$. Suppose $B = V - (T_1 \cup T_2)$ is non-empty. Then $S \cap \delta(B)$ consists only of arcs directed out of B. Again, this contradicts the fact that S is strongly connected. So there are no arcs between $T_1 - \{v_1\}$ and $T_2 - \{v_2\}$. It follows that both $G - T_1$ and $G - T_2$ are strongly connected when we consider S restricted to $G - T_1$ and S restricted to

$G - T_2$, respectively. Now a and $\bar{a}$ are not the only arcs in D connecting one vertex in T_1 and another in T_2. Otherwise u_1 and/or u_2 would be cut vertices. So let a' be an arc crossing from T_1 to T_2. Without loss of generality we may assume it is directed out of T_1 and into T_2. In addition, it is incident to exactly one of u_1 or u_2. Then $S - \{a\} \cup \{a'\}$ is a minimum strongly conneceted subgraph with one less digon than S, a contradiction. □

COROLLARY 3.1. *If an arc a is in an optimal solution to G then we may consider the graph $G - \{\bar{a}\}$ instead.*

Proof. If a is in an optimal solution then by Lemma 3.2 its complement $\bar{a}$ (if it is present in G) is not in some optimal solution. Hence we may solve the problem on the smaller graph $G - \{\bar{a}\}$. □

We say that an arc a that is not present in some optimal solution is *redundant*. If ever we find a redundant arc then we may instead consider the graph $G - \{a\}$. For example, if an arc is in a digon and we can show that its complement is forced to be in every optimal solution then by Corollary 3.1 the arc is redundant. More generally, redundant arcs may also occur in the presence of forced paths. We say that directed path, P, consisting of the vertices $v_0, v_1, \ldots v_r$, is *forced* if it is in some optimal solution. In addition, suppose that we have a vertex u and arcs (u, v_0) and (u, v_r). Choosing the arc (u, v_0) will always be preferable to (u, v_r). This follows as, given that P is present, (u, v_0) also creates a path from u to v_k as well as from u to v_0. Hence (u, v_r) is redundant and we may consider the smaller graph $G - (u, v_r)$. Similarly, given such a forced path P and arcs (v_0, u) and (v_r, u), we see that (v_0, u) is redundant.

Now consider a cycle in G consisting of three arcs, $(v_0, v_1), (v_1, v_2)$ and (v_2, v_0). First, note that the nodes v_0, v_1 and v_2 may induce up to six arcs in G, the three cycle arcs and their complements. Suppose there is an optimal minimum strongly connected subgraph containing at least 2 of these six possible arcs. Then we may choose the three cycle arcs, contract the cycle, and find a $\frac{3}{2}$-optimal subgraph on the resultant graph. The following lemma shows that this approach will be sufficient to produce a $\frac{3}{2}$-approximation factor.

LEMMA 3.3. *If three nodes induce two arcs is some optimal solution then we may choose any 3-cycle through the nodes and consider the resultant sub-problem.*

Proof. Let the optimal solution have OPT arcs, two of which are induced by v_0, v_1 and v_2. So $k = OPT - 2$ arcs are used on the arc set induced by $G' = G/\{v_0, v_1, v_2\}$, the graph formed by contracting the 3-cycle. Let OPT' be the size of the optimal solution to G'. Clearly $OPT' \leq k$. So if we can find $\frac{3}{2}$-optimal solution to G' and then add the three arcs in the cycle we will have a solution to G. This solution will use at most $\frac{3}{2}OPT' + 3 \leq \frac{3}{2}k + 3 = \frac{3}{2}(k+2) = \frac{3}{2}OPT$ arcs. Hence we have a $\frac{3}{2}$-optimal solution to G. □

We will apply Lemma 3.3 in the following manner. If we ever find a 3-cycle that induces two arcs in some optimal solution then we will consider the graph resultant from contracting the cycle.

There is one other operation that we will use in our algorithm, that of *Component Merging*. Suppose we find a cycle $\mathcal{C}'$ in $\mathcal{G}$ that strongly connects together several components (which do not include the root component). In such circumstances we may treat the group formed by such additions as one mega-component. We may define a new component graph $\mathcal{G}'$ on the same original components, with the exception that we will use the mega-component rather than its many base-components. This process of refining our component graph will be termed component merging. We will then grow a new tree $\mathcal{T}'$ on $\mathcal{G}'$ and begin again. Here we must bear in mind that we must also charge the arc set $\mathcal{C}'$ to the the mega-component, on top of any other arcs that may subsequently be charged to it.

Note that contractions, deletions and component mergings can only be carried out a polynomial number of times. As such, repeatedly carrying out these operations (and solving on the resultant graphs) will not affect the polynomial nature of our algorithm.

4 The Algorithm and an Analysis

We are now ready to describe how to implement our version of the cycle contraction algorithm. Recall that we wish to efficiently contract a cycle (or circuit), in the component graph, through the root and the target component, and repeat. Given such a cycle, $\mathcal{C}$ in $\mathcal{G}$, suppose that the contraction of $\mathcal{C}$ results in the contraction of a set of vertices $V' \subseteq V$. In order to show our algorithm has the desired performance, it is sufficient to prove the following theorem.

THEOREM 4.1. *Suppose that the algorithm calls for the contraction of a cycle $\mathcal{C}$. Let this induce the contraction of the vertex set V'. If V' induces r arcs in our final solution then V' induces at least $\frac{2}{3}r$ arcs in the lower bound.*

Proof. Our proof of Theorem 4.1 will be split into several parts. We consider first the case in where the cycle, $\mathcal{C}$ in $\mathcal{G}$, that we contract passes only through strongly connected components. We then partition our proof depending upon whether the the target component contains at least 4 arcs or is a digon or 3-cycle. Finally we examine the situation in which the root component, target component or other component contained in $\mathcal{C}$ is a path.

So we wish to contract a cycle through the root and the target component. In addition, we desire the property that the vertices contracted in this process induce enough arcs in the lower bound to cover, with our $\frac{3}{2}$ performance guarantee, the number of arcs they induce in the final solution. We have seen that this is easy when the target component is strongly connected and contains at least four arcs, since this gives us two extra arcs to play with. We now show that Theorem 4.1 holds for the other types of target component.

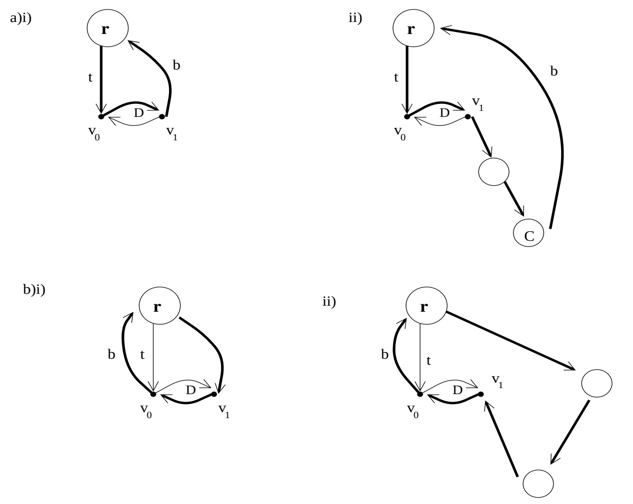

Figure 3: A target digon.

DIGONS. Suppose the target component is a digon D. We have seen that it is too costly to charge 2 extra arcs to this component. Recall that we begin with the assumption that all of the components which we encounter in this section are strongly connected. The situation in which some of the components may be paths will be dealt with subsequently. We begin with some easy situations.

a) There is a back arc b emanating from the subtree T_1. Suppose b is incident to v_1, see Figure 3a)i). In this case we may add the arc b and remove the arc (v_1, v_0). Notice that the removal of the arc (v_1, v_0) implies that the net charge to the digon is just one arc, giving our desired $\frac{3}{2}$ factor. Suppose, instead, that b emanates from a component C in $T_1 - \{v_1\}$, where T_1 is the subtree in $\mathcal{T}$ rooted at v_1. See Figure 3a)ii). In this case we may add the arc b and remove the arc (v_1, v_0). Along with the appropriate tree arcs this creates a cycle $\mathcal{C}$ to contract. This cycle also connects into the root the all components between D and C in the subtree T_1. Note each such component receives a net charge of one arc. Since these components contain at least two arcs, we have at worst a $\frac{3}{2}$ blow-up factor.

b) There is a back arc b emanating from v_0. Now since v_0 is not a cut vertex there must be a path from r to v_1 that does not pass through v_0. This path may consist of a single arc from r to v_1, see Figure 3a)i). The general case is shown in Figure 3a)ii). Combining this path with the arc b allows us to drop the arc (v_0, v_1) and still create a cycle $\mathcal{C}$. Again the net charge to the digon (and any other component in the cycle) is just one.

Observe that Figures 3a)ii) and b)ii) generalise Figures 3a)ii) and b)ii). In particular, Figure 3a)ii) contains a path from v_1 to the root passing through various components of $\mathcal{G}$. In Figure 3a)i) this path is a trivial path consisting of a single arc from v_1 to the root. The situation with regards to Figures 3b)i) and ii) is similar, with the path, this time, being directed from the root to v_1. Note that (since we are assuming that each node in $\mathcal{G}$ is strongly connected and contains at least two arcs) if non-trivial paths are found then the charge associated with each additional component is just one arc, giving a blow-up factor of at most $\frac{3}{2}$. Observe that the general case of non-trivial paths is just a combination of the trivial path case and a path through these additional components. Hence, it will be the case that the trivial path case provides a worst case analysis. If we can show that the blow-up associated with the trivial path case is at most $\frac{3}{2}$, then it will also follow for the general case. As a result, unless otherwise stated, we will make this worst case assumption that connections between specific vertices will be via a single arc rather than via a path in the component graph.

c) The only possible situation remaining is the case of a back arc b going from $T_0 - \{v_0\}$ to the root component. Again, for a worse case scenario we may assume it emanates from C_0 a child component of v_0 in T_0. In such circumstances we will be forced to contract into the root several components together. We will find a circuit, rather than a cycle, to contract. In particular we will contract two cycles (one of which does not contain the root component) and argue that the process can be done efficiently.

Now suppose there is a back arc path from v_1 to v_0, or vice versa, via C_1. In the worst case this path consists of just two arcs. See Figure 5a). So by adding both these path arcs whilst removing (v_0, v_1) we connect up, into the root component, the digon and C_0 and C_1. Thus if C_0 and C_1 both contain at least three arcs then we use 12 arcs in total for the three components. Since these three components contain 8 arcs this gives a $\frac{3}{2}$ blow-up. Notice that we do indeed contract a circuit which consists of two cycles that intersect at D.

What, though, if C_0 or C_1 are themselves digons? We will eliminate digons via a recursive process. In order to show how this will be done we first introduce some more notation. Given a digon D_j, label the vertices of the digon as follows. Let the $v_{ja} \in D_j$ be incident to arcs that currently connect T_{D_j} to $G - T_{D_j}$. Let the v_{jb} be the other vertex in D_j. We say that a child component C of D_j is a *rung* if there is a path in the subtree T_{D_j} from v_{ja} to v_{jb}, or vice versa, that passes through C. In the worst case this path will contain just the single component C. In this case we will label the path arcs p_1 and p_2, see Figure 4a) and b). Note that the use of such paths allows the removal of either the arc (v_{ja}, v_{jb}) or the arc (v_{jb}, v_{ja}). Hence the path forms a 3-cycle through the digon D_j, and its three arcs will be charged to the digon. This observation will be used in a form of recursion that enables us to "ignore" digons.

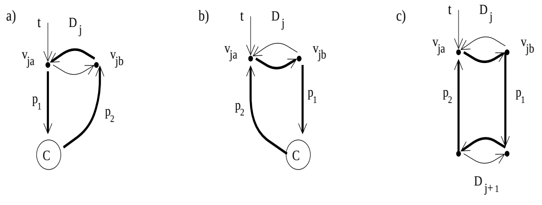

Figure 4: Rungs.

In addition, we say that a rung C is a *base rung* if either C contains at least 3 arcs or if C is a digon D_{j+1} and the path arcs are incident to different vertices in D_{j+1}. Notice that in the latter case there is a 4-cycle through the vertices of D_j and D_{j+1}. One such possibility is shown in Figure 4c). Should we find a digon D_{j+1} that is a base rung and a child of another digon D_j we will then perform our component merging operation. We replace these two digon components in the component graph $\mathcal{G}$ by the 4-cycle that passes through them.

Thus in the previous case, where we assumed that C_0 and C_1 both contain at least three arcs, the component C_1 is a base rung child of the digon D (and C_0, whilst not technically a base rung, plays a similar role).

We now illustrate the basic recursive argument that will be applied on a specific example.. Let C_1 be a digon D_1. Hence C_1 is a rung. Suppose also that it is not a base rung. Suppose there are arcs $p_1 = (v_1, v_{1a})$ and $p_2 = (v_{1a}, v_0)$. The reader may observe that were we to contract the 3-cycle formed by p_1, p_2 and (v_0, v_1) then we would essentially be in the same situation as in Figure 5a) but with the digon D replaced by the digon D_1. Suppose then that D_1 has a child component

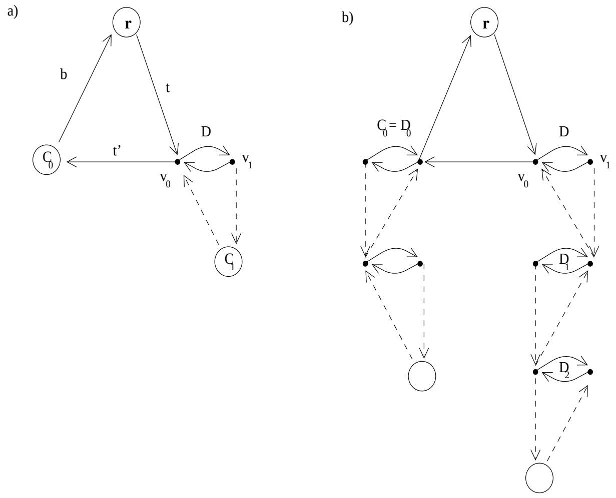

Figure 5: Growing a Pair of Digon Ladders

C_2 that is a base rung. Now if C_2 contains at least three arcs then the blow-up factor of merging the circuit illustrated is again at most $\frac{3}{2}$. If $C_2 = D_2$ is a digon that is a base rung then there is a 4-cycle through D_1 and D_2 and we consider a new component graph $\mathcal{G}'$ with the 4-cycle as a node. There is one final case to consider. If the digon D_2 has a back arc that goes beyond D_1 and into D then it is easy to verify that such a circumstance can only be beneficial.

We now return to the general case and formalise our arguments. We will successively eliminate digons, until we reach a base rung. Assume that C_0 is a digon and consider the picture shown in Figure 5b).

We take a child component of D that is a rung (if there are many such child components we may choose the one with the highest priority in $\mathcal{T}$). If this component is a base rung then we stop. Otherwise, if this rung is a digon D_1 that is not a base rung, we recurse on the children of D_1. This process forms a "ladder of digons", $D, D_1, D_2, \ldots D_p$. Similarly we grow another ladder of digons, by the same method, beneath D_0. Such ladders are shown in Figure 5b). If both of these processes terminate successfully then we have a guarantee of the desired blow-up factor.

So suppose one of the ladders does not terminate with a base rung. Consider the last digon D_p in the ladder. Then either D_p has no children or it has no child components that are rungs. Consider the former case. Now a back arc b must emanate from v_{pb} otherwise the arc (v_{pb}, v_{pa}) is forced and we may consider the graph $G - (v_{pa}, v_{pb})$. In addition, the head of b must also be incident to the tail of p_1 otherwise D_p is a base rung. Again its is only beneficial if b goes beyond

D_{p-1}. Now there must be a down arc d entering into v_{pb}, otherwise the arc (v_{pa}, v_{pb}) is forced and we may consider the graph $G - (v_{pb}, v_{pa})$. In the worst case d emanates from the digon D_{p-1}. Here we may construct a 4 cycle through the vertices of D_{p-1} and D_p. This is shown in Figure 6a) and b). We then apply the merging components method, using the 4-cycle instead of the two digons.

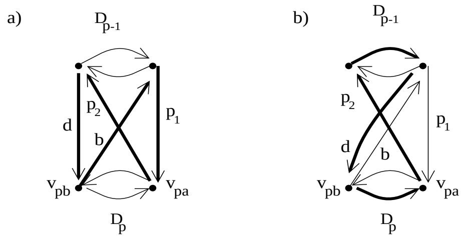

Figure 6: Merging digons.

Similar arguments apply in the latter case where D_p has child components but none are rungs. Hence the ladder growing process terminates and we are done.

Finally, note that any component that is incident to an arc a that emanates from the root component may be chosen as a target node. This follows as we may force a to be chosen as the first arc in $\mathcal{T}$. In fact, any component that is adjacent to the root (via an in-arc or an out-arc) may be chosen as the target node as we may as well have chosen $\mathcal{T}$ to be an in-tree rooted at r. By symmetry, the arguments that follow with respect to an out-tree $\mathcal{T}$ would also apply to an in-tree. This observation allows us to now assume that no components adjacent to the root component are digons (or, in fact, strongly connected with at least 4 arcs).

3-Cycles. Let our target component be a 3-cycle. Again, we will assume that none of the components which we encounter in this section are paths. In addition, should we encounter a component that represents a digon then the "ladder of digons" argument may be applied in the same manner as before. This observation allows us to assume that all of the components which we encounter are strongly connected and contain at least three arcs.

We may assume that all cycles in component graph $\mathcal{G}$ that pass through the root component and the target component are digons. Suppose otherwise and such a cycle passes through three or more components. The two components adjacent to the root component contain at least three arcs each. The blow-up factor is then at most $\frac{3}{2}$. This factor occurs for the case in which there are three components (i.e. three arcs are added) and the non-root components are both 3-cycles. This is illustrated in Figure 7i).

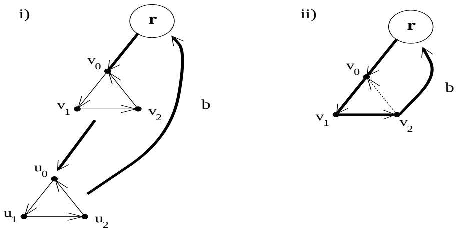

Figure 7: Some 3-cycles.

Let the 3-cycle consist of the nodes v_0, v_1 and v_2 with the tree arc, from the root to the cycle, being incident to v_0. By the arguments given above, there must be a back arc b emanating from the 3-cycle. If b is incident to v_2 then we can choose b and remove the arc (v_2, v_0). This situation is shown in Figure 7ii). Thus, we need only 4 arcs in total and they can all be accounted for by the 3 arcs in the cycle. Hence, since v_0 is not a cut vertex there must be a back arc b emanating from v_1. This implies that there is no arc from the root component to v_2, otherwise we may omit the arc (v_1, v_2) in a similar manner to that of Figure 7ii).

There is another relatively straightforward situation. Suppose a tree arc t emanates from the node v_2 and that it leads to a child component C_2. If for such a subtree, T_2, there emanates a back arc b' into v_0 then we can connect up the 3-cycle and component C_2 for a cost of 3 arcs. This follows as in choosing d and b' we may remove arc (v_2, v_0). Since C_2 contains at least 3 arcs this gives the desired blow-up factor. Note that if b' emanates from a component lower than C_2 in T_2 then this only helps us, as we connect up any additional component for a charge of just one arc. This situation is shown in Figure 8i). The corresponding cases for subtrees T_1 and T_0, with child components C_1 and C_0, are shown in Figures 8ii) and iii).

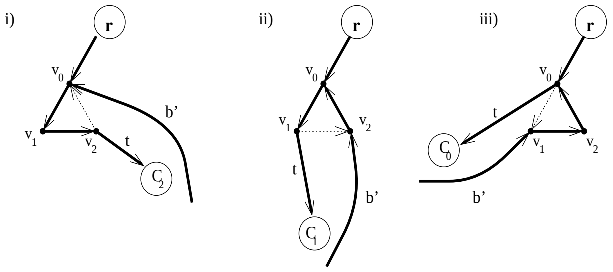

Figure 8:

Now consider the remaining possibilities. Recall there is a back arc b out of v_1. By the ordering given by DFS method, there are no arcs from vertices in T_0 to vertices in either T_1 or T_2. There are also no arcs from vertices in T_2 to vertices in T_1.

a) There is a back arc b' coming into v_2 from T_0. Choosing b' allows us to drop the arc (v_1, v_2). The net charge is $4-1=3$ arcs which we charge to the cycle and C_0. See Figure 9a). Since we can deal with the case of back arcs entering into v_1, or going beyond the 3-cycle, we may now assume that for all the C_0, every back arc goes into v_0.

b) There is a back arc b' coming into v_1 from T_2. This situation is dealt with analogously to the previous part, and is shown in Figure 9b).

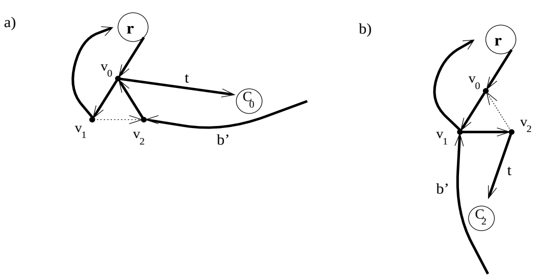

Figure 9:

c) There is an arc a leaving T_2 and entering T_0. Since the only back arcs leaving T_0 enter into v_0 itself, a creates a path between T_2 and v_0. Thus a acts like an arc entering v_0 and can be dealt with in the same way (Figure 8i)), allowing the removal of (v_2, v_0). Hence we may assume that for all the C_2, all the back arcs go into v_2.

d) There is an arc a leaving T_1 and entering T_2. Now a acts like an arc entering v_2 and can be dealt with in the same way (Figure 8ii)), thus allowing the removal of (v_1, v_2).

e) In what follows we are interested in the possible routes from v_1 to v_2. Suppose there is an arc a from v_2 to a vertex $u \in T_0 - \{v_0\}$. Now in any optimal solution there must be a directed path P from u to v_0 consisting only of vertices in that subtree. As such, there must be an optimal solution that does not contain the arc (v_2, v_0). This follows, from a forced path argument, as the arc a will also connect v_2 to v_0 via P. Using a then is at least as good as using (v_2, v_0) since it also provides a path from v_2 to the vertices of P. Hence we may remove the arc (v_2, v_0) and resolve the problem on $G - (v_2, v_0)$. By a similar argument there is no arc a from v_1 to a vertex in T_2.

f) Otherwise the only way to reach v_2 from v_1 is via the arc (v_1, v_2), or if it is present (v_0, v_2). In addition, the only way up the tree from v_2 is via the arc (v_2, v_0) or, if it is present, (v_2, v_1). Hence, at least two arcs induced by the vertices v_0, v_1 and v_2 are forced to be in an optimal solution. So by Lemma 3.3 we may contract the 3-cycle and recurse on the smaller graph.

Paths. We now consider the situation in which a component may represent a path P. There are three cases to analyse. The cases where the path is the root component, the target component or a component in a contracted cycle $\mathcal{C}$ associated with a different target component. Let P consist of the vertices $v_k, v_{k-1}, \ldots v_0$ with k arcs of the form (v_{i+1}, v_i). We will draw the path so that the arcs are directed from right to left. We will call an arc (v_i, v_j), $i \leq r < j$, a *forward* arc wrt v_r. We call an arc (v_j, v_i), $i \leq r < j$, a *back* arc wrt v_r. We aim to connect up the path with at most a factor $\frac{3}{2}$ blow-up. To do this we will start at the leftmost vertex of the path and add arcs to strongly connect it up to its immediate neighbours. We continue this process rightwards until all the vertices on the path are strongly connected. An example will be of use here, see Figure 10a). Let v_0 be the leftmost vertex. Suppose there is a forward arc d going to v_2. Then adding d strongly connects v_0, v_1 and v_2. Now these three vertices induce two path arcs, (v_2, v_1) and (v_1, v_0). We may charge d to these two arcs to give a blow-up factor $\frac{3}{2}$. We may then think of v_0, v_1 and v_2 as being contracted together to form a super-node on the path. Thus our general picture remains the same as that of Figure 10a) after each step.

We will show that the charge associated, in strongly connecting up the path, is at most half of the length of the path. Consider the first case, where the component representing the path is the root component. Hence we only need to ensure that the vertices on the path become strongly connected, and not worry about connecting the path upwards into the root component. We will also assume that for each T_i the C_i will represent strongly connected components. Again, the case where the C_i are paths will be dealt with later.

During the following connection process the vertex v_0 may represent multiple nodes but $v_1, v_2 \ldots v_k$ can not. From now on, let d be the forward arc reaching deepest along the path from v_0. Since v_0 may represent multiple nodes, there may be several arcs reaching the same node. Whether the choice between the arcs is important will be made apparent. Note also that, by the DFS ordering, there are no arcs from a vertex in T_i to a vertex in T_j, where $i < j$.

a) Take v_0 and consider d. If d goes to v_2 or beyond then we add d and charge it to the (at least) two path

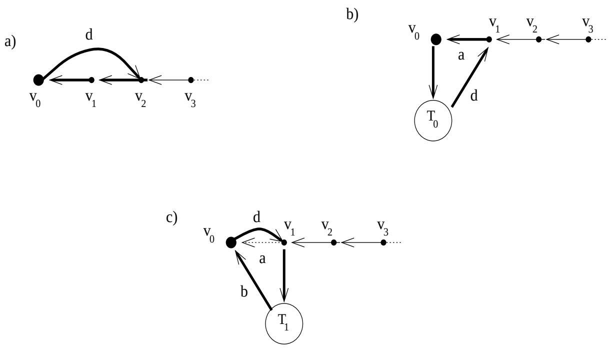

Figure 10: Some paths.

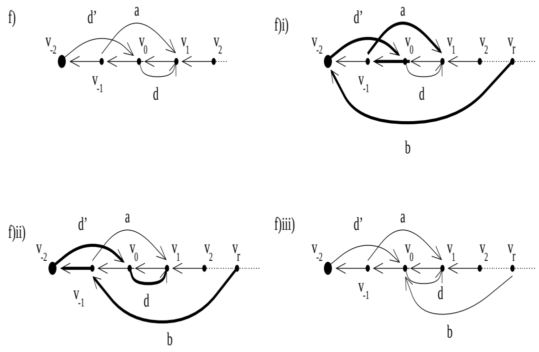

Figure 11: Unravelling a path.

arcs. See Figure 10a).

b) So suppose d only reaches v_1. Now if d comes from T_0 then we consider the component C_0. Suppose that C_0 contains at least three arcs. Now adding d strongly connects together v_0, v_1 and T_0. See Figure 10b). We need to charge the tree arc a_0 and d. We charge them to C_0 *and* (v_1, v_0). Hence we charge these two arcs to at least four arcs contained in the lower bound, a $\frac{3}{2}$ factor. Suppose instead that C_0 is a digon. Here we may apply the "ladder of digons" argument to obtain the same factor.

c) There is a back arc b from T_1 into T_0. See Figure 10c). We may add d, b and the tree arc a_1 whilst removing (v_1, v_0). Similar arguments to those of part b) now apply.

d) Otherwise unravel our process one step. For clarity we use negative indices to label the vertices of the path after unravelling the process. If the previous deepest arc d' covered at least 3 path arcs then we can charge d and d' to the (at least) four path arcs.

e) If the previous deepest arc d' covered just one path arc then d is a forced arc as it is the only way in which to construct a directed path from v_0 to v_1. Hence we may consider the graph $G - (v_1, v_0)$.

f) So assume that d' covered 2 path arcs. In addition, there is an arc a from v_{-1} to v_1. Otherwise d is forced and we again consider $G - (v_1, v_0)$. See Figure 11f). Note also that since v_1 is not a cut vertex, there must be a directed path P', that does not contain v_1, from a vertex $v_r, r \geq 2$ to a vertex $v_s, s \leq 0$. First assume that P' consists of a single arc b between vertices in P.

i) If there is an arc b into v_{-2} then use the arcs $b, d', (v_0, v_{-1})$ and a. See Figure 11f)i). These four arcs are covered by the lower bound of the three path arcs from v_{-2} to v_1. We don't yet have a path from v_1 to v_0 but later in connecting up the rest of the path we must find a directed path from v_1 to v_r. This will complete a cycle through v_1 and v_0. In fact, the next stage in the process will then be similar to our basic case except that, in effect, we get the edge b for free (since it has already been charged to the arcs on the path between v_{-2} and v_1). This, of course, can only help us and our iterative process continues as before.

ii) If there is an arc b into v_{-1} then we proceed as in Figure 11f)ii) and again use 4 arcs associated with the three path arcs.

iii) Finally suppose that all the type b arcs are incident to v_0. See Figure 11f)iii). Consider one such arc (v_r, v_0). Now the only way to reach v_r from a vertex $v_s, s \leq 0$ in the path P is via the vertex v_1. Hence in any optimal solution there is a directed path from v_1 to v_r that does not use any of the vertices $v_s, s \leq 0$. Hence, by a forced path argument, using the arc (v_r, v_0) will always be preferable to using the arc (v_1, v_0). Thus (v_1, v_0) is a redundant arc and we may consider the graph $G - (v_1, v_0)$.

Now consider the case where P' does not consist of a single arc but goes through other components in the component graph $\mathcal{G}$. The arguments above still apply with the other components incident to P' being charged just one arc.

It follows that in the specific case where the path is the root component we use at most an additional $\lfloor \frac{k}{2} \rfloor$ arcs.

Now consider the general case where P is a non-root component i.e. it is the target component or is a component in a cycle $\mathcal{C}$ contracted for a different target component. Let P be the root of a subtree T. For a worst case analysis, we may assume that there are no back arcs from a component in $T - P$ to $G - T$. First remove the DFS tree arc connecting $G - T$ and P. Add an arc a from P to $G-T$ with the property that a originates as far to the left on P as possible, say from v_j. Figure 12i) shows the resulting situation. Here the other possible arcs connecting P to $G-T$ are labelled b, b' and d. This notation is used as the b arcs may be used as back arcs and the d arcs may be used as forward arcs as before. The b' arcs are slightly different. Adding such an arc does not create a "back path" via $G-T$ unless an arc from $G-T$ to P (i.e. a type b or type d arc) has already been chosen.

This allows us to proceed as before adding forward and back arcs to strongly connect up the path. We make one proviso. Should a b' arc from P to $G-T$ be required as a back arc, and no arc from $G-T$ to P has currently been selected, then we add an appropriate arc and remove a. An example where the addition of b' leads to the removal of a and addition of b is shown in Figure 12ii). Note that such a method still leads to a net charge of one when b' is added.

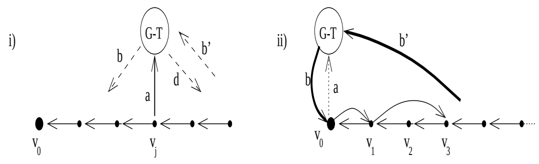

Figure 12: A non-root path.

The previous arguments, used for the specific case of the path being the root component, may now be applied to this general case. Hence $\lfloor \frac{k}{2} \rfloor$ arcs are required to internally strongly connect the path. Now we have two possible situations after we have internally connected up the path.

Type I) No arcs between P and $G-T$ were chosen in the process (except for our original choice of a). In this case we need to add an extra arc a' to connect the path upwards. So our total cost is at most $\lfloor \frac{k}{2} \rfloor + 2$ (including one each for a and a'). Notice that a and a' play no part in internally connecting up the path. This observation will be of use later.

Type II) Some arcs, apart from a, between P and $G-T$ were chosen in the process. These arcs are included in the $\lfloor \frac{k}{2} \rfloor$ charge to the path but they also succeed in strongly connecting the path upwards. Hence the charge is $\lfloor \frac{k}{2} \rfloor + 1$ (including one for a).

Recall that associated with each path were two "free" arcs and so we actually have $k+2$ arcs to charge these additional arcs to. It follows that if the path is a target component then the total number of arcs used is at most $k + \lfloor \frac{k}{2} \rfloor + 2 < \frac{3}{2}(k+2)$, as required.

So consider the case in which the path is a component in a cycle C for which the target component was a digon or 3-cycle. Again associated with the target path are two free arcs. Thus, we have at least three arcs (due to the $\frac{3}{2}$ performance guarantee one extra arc is also allowed with the two free arcs) which we can use in strongly connecting up a path component. It now follows easily that it would only have been beneficial to encounter any paths in any of the circuits we found in the previous sections, where the target components were digons and 3-cycles.

Finally, consider the case in which the path is a component in a circuit for which the target component is another path. Recall that this occurs when we reallocate path arcs to a child component C_i below. We postponed the discussion of the situation in which these child components were themselves paths. We deal with this topic here. Call the upper path P' and the lower path P. In this situation we allocate one arc from P' to the P and force two other arcs to be chosen that strongly connect upwards P. A total of three extra arcs. Now, if the lower path, P, is of type I) the arcs a and a' are superfluous so the charge to P is $\lfloor \frac{k}{2} \rfloor + 3$. Whereas, if P is of type II) then the arc a was needed in connecting up the path so with these extra three arcs we have a charge of $\lfloor \frac{k}{2} \rfloor + 4$. Given that we also reassigned one arc from P' to P we now have $k+3$ arcs associated with P. Now since P has a single parent such extra arcs can be reallocated from P' to P at most once. This completes the proof that the blow-up factor for paths is at most $\frac{3}{2}$ since $k + \lfloor \frac{k}{2} \rfloor + 4 < \frac{3}{2}(k+3)$. The theorem follows. $\square$

References

[1] H. Hsu, An Algorithm for Finding a Minimal Equivalent Graph of a Digraph, *Journal of the Association for Computing Machinery*, **22(1)**, 11-16, (1975).

[2] S. Khuller, B. Raghavachari and N. Young, Approximating the Minimum Equivalent Digraph, 5^{th} *Annual ACM-SIAM Symposium on Discrete Algorithms*, **41(4)**, 859-72, (1995).

[3] S. Khuller, B. Raghavachari and N. Young, On Strongly Connected Digraphs with Bounded Cycle Length, *Discrete Applied Mathematics*, **69(3)**, 281-89, (1996).

[4] D. Moyles and G. Thompson, An Algorithm for Finding a Minimum Equivalent Graph of a Digraph, *Journal of the Association for Computing Machinery*, **16(3)**, 455-60, (1969).

[5] S. Vempala and A. Vetta, Factor $\frac{4}{3}$ Approximations for Minimum 2-Connected Subgraphs, in K. Jansen and S. Khuller (eds.), *Approximation Algorithms for Combinatorial Optimization*, Springer-Verlag, Lecture Notes in Computer Science, **1913**, 262-73, (2000).