Pairwise Edge Disjoint Shortest Paths in the n-Cube

Teofilo F. Gonzalez and David Serena
Department of Computer Science
University of California
Santa Barbara, CA 93106-5110
email: teo@cs.ucsb.edu

August 1, 2006

Abstract

Complexity issues intrinsic to certain fundamental data dissemination problems in high performance network topologies are discussed. In particular, we study the p-pairwise edge disjoint shortest paths problem. An efficient algorithm for the case when every source point is at a distance at most two from its target is presented and for pairs at a distance at most three we show that the problem is NP-complete.
Keywords: Hypercube, n-cube, edge disjoint shortest paths, NP-completeness.

1 Introduction

The n-cube is a fundamental structure for parallel computing. Several systems with this communication architecture have been built. The SGI Origin2000 and Onyx2 computing systems are platforms whose interconnection network is a variation of the n-cube. Different message routing problems arise while executing parallel programs on an n-cube connected machine. One type of these problems consists of transmitting messages concurrently from source nodes to their corresponding target nodes. In this paper we study the offline version of one such problem: the p-pairwise edge disjoint shortest paths problem. Path disjointness allows messages with different origin/destination to be routed concurrently and shortest paths are likely to minimize communication time. The above communication patterns arise when executing parallel versions of many well-known algorithms. These algorithms include: fast Fourier transform; transposing a matrix; permuting the elements stored at the nodes of an n-cube, which is the final operation in a count sort (permutation routing); concentrating (in the first k nodes of an n-cube) a sequence of elements stored in the nodes of an n-cube (data concentration), or the reverse process which is called data spreading. More complex communication operations that include the above communication patterns arise while multiplying matrices or solving systems of linear equations iteratively.

A *routing request* consists of p pairs of nodes denoted by

$$X = \{X_1, X_2, \ldots, X_p\},$$

where $X_i = (s_i, t_i)$, for $1 \leq i \leq p$, and all the s_is and t_is are distinct. Each pair $X_i = (s_i, t_i)$ consists of two *endpoints* which are called the *source* and *target*, respectively. By the above definition the number of source and target assigned to a vertex in the n-cube in a routing request is at most one. Every node in the n-cube is represented by an n-bit string and there is an edge in the n-cube between two nodes if their bit representation disagrees in exactly one bit. The distance between

the source and target nodes of pair X_i (or *pair distance*) is denoted by $d(X_i) = d(s_i, t_i)$ and in the n-cube it is simply the number of bits that differ in the bit representation of s_i and t_i. The distance $d(a, b)$ is frequently referred to as the "Hamming Distance" between the nodes a and b in the n-cube. By a *shortest path* for the pair X_i we mean any path from s_i to t_i with length equal to $d(X_i)$, i.e., the path must be a shortest path in the n-cube between the two nodes independent from any other paths. Routing requests, by definition, limit the number of source and target assigned to each vertex to at most one, therefore $d(X_i) \geq 1$ for every pair X_i.

The *p-pairwise edge disjoint shortest paths problem* for the n-cube is given a routing request $X = \{X_1, X_2, \ldots, X_p\}$, find edge disjoint shortest paths in the n-cube for all the pairs X_i. That is, for every pair X_i find a path with $d(X_i)$ edges such that no two paths have an edge in common. The *node disjoint shortest paths problem* is given a routing request X in the n-cube, find shortest paths connecting each s_i to t_i such that no two paths have a node (including the source and target) in common (see [5] for results on node disjoint shortest paths and related problems). In the context of the (undirected) n-cube the order of the source to target in the routing request is not important. Therefore, we could have used undirected pairs $X_i = \{s_i, t_i\}$ instead of the directed pairs (s_i, t_i). In what follows when we establish NP-completeness and NP-hardness results it is for the decision version of our problems.

For the 2-cube the routing request $X = \{(00, 01), (10, 11)\}$, has edge disjoint shortest paths ($00 \leftrightarrow 01$ and $10 \leftrightarrow 11$). However, the problem instance $Y = \{(00, 11\}, (10, 01)\}$ does not have edge disjoint shortest paths. The instance X has *node* disjoint shortest paths, but Y does not have *node* disjoint shortest paths. For the 3-cube the routing request $X = \{(000, 101), (010, 001), (100, 111)\}$ has edge disjoint shortest paths ($000 \leftrightarrow 001 \leftrightarrow 101$, $010 \leftrightarrow 011 \leftrightarrow 001$, $100 \leftrightarrow 110 \leftrightarrow 111$), but it does not have node disjoint shortest paths because the three adjacent vertices to node 000 are source or target nodes for other pairs.

Message routing problems are known to be computationally intractable for general graphs when one allows arbitrary length paths, rather than just shortest path ones. Madhavapeddy and Sudborough [7] show that the p-pairwise edge disjoint (arbitrary length) paths problem for the n-cube is NP-complete. Unfortunately they did not include a proof that the problem is in NP and we are unable to independently validate that proposition. Their version of the problem allows for nodes to be the source and target of many pairs [7]. In this paper we consider the shortest paths version of the problem when each vertex can be either the source or target of at most one pair. This corresponds to the case when each n-cube node either sends or receives at most one message. We show that the p-pairwise edge disjoint shortest paths problem for the n-cube is NP-complete even when every pair has pair distance at most 3 (Section 2.2), but solvable in polynomial time when all the pair distances are at most 2 (Section 2.1). Gonzalez and Serena [4] considered the extreme version of the p-pairwise edge disjoint shortest paths problem where $d(X_i) = n$, for $1 \leq i \leq p$. For the extreme version of the problem a polynomial time algorithm for all possible values of p, as long as n is odd, is given in [4].

Madhavapeddy and Sudborough [7] conjecture that the p-pairwise node disjoint paths problem is also NP-complete. In [5] a reduction, based on the construction in Section 2.2, is used to show this problem is NP-hard. The techniques used in this paper to establish our NP-completeness results have been extended to the p-pairwise edge disjoint shortest paths problem in the doubled n-cube [2]. The basic component used in [2] is a variation of Lubiw's construction [6]. Gonzalez and Serena [3] show that the node and the edge disjoint shortest paths problems in the *grid (mesh)* are NP-complete.

2 Edge Disjoint Shortest Paths

We present a simple polynomial time algorithm for routing requests where all pairs have distance at most two, and then we show that the for routing requests with pair distance at most three the p-pairwise edge disjoint shortest paths problem in the n-cube is NP-complete.

2.1 Algorithm for Pairs at Distance at Most Two

Given an instance (X, p, n) of the p-pairwise edge disjoint shortest paths problem in the n-cube, we construct an instance (U, C) of 2-SAT as follows. For each pair $X_i \in X$ there is a Boolean variable u_i. For every pair X_i with $d(X_i) = 2$, there are two possible shortest paths from s_i to t_i. Let us denote such paths by $P_{i,0}$ and $P_{i,1}$. A satisfying assignment where Boolean variable u_i has the value true means that the pair X_i is connected by path $P_{i,1}$, otherwise it is connected by the path $P_{i,0}$. For every pair X_i with $d(X_i) = 1$ we add the clause $\{u_i\}$ the path $P_{i,1}$ consists of the (only) edge between the source and target of the pair.

For every two paths $P_{i,a}$ and $P_{j,b}$, $i \neq j$, defined above with at least one edge in common, we add the clause $\{\bar{u}_i, \bar{u}_j\}$ if $a = b = 1$; $\{\bar{u}_i, u_j\}$ if $a = 1$ and $b = 0$; $\{u_i, \bar{u}_j\}$ if $a = 0$ and $b = 1$; and $\{u_i, u_j\}$ if $a = b = 0$.

It is simple to prove that the p-pairwise edge disjoint shortest paths problem in which each pair has pair distance at most two has a solution if and only if the instance constructed from it, (U, C), is satisfiable. A simple algorithm (which we call 2-EDSP) can implement the above strategy. The following theorem, which we state without a proof, formalizes this result. Implementation details for algorithm 2-EDSP and the proof of the theorem appear in [5].

Theorem 1 *Given any instance of the p-pairwise edge disjoint shortest paths problem for the n-cube, Algorithm 2-EDSP constructs a valid set of paths, whenever such paths exist, in $O(pn)$ time.*

2.2 Complexity for the Problem With Pairs at Distance at Most Three

In Theorem 2 we show that the p-pairwise edge disjoint shortest paths problem is NP-complete. We establish this result by reducing the L3-SAT problem to it. The L3-SAT problem is defined as follows.

Input: Given a set U of Boolean variables $\{u_1, u_2, \ldots, u_v\}$, and a collection of clauses $C = (c_1, c_2, \ldots, c_w)$ over U with each clause having two or three literals, such that all clauses include at most three literals corresponding to same variable, and every literal is in at most two clauses.

Question: Is (U, C) satisfiable?

The 3-SAT problem is L3-SAT without the last two conditions and the removal of 2-literal clauses. L3-SAT and 3-SAT are NP-complete problems [1].

We begin by discussing our polynomial time transformation from L3-SAT to the decision version of the p-pairwise edge disjoint shortest paths problem. There are three types of components: *setting-and-fan-out*, *conveyor*, and *clause-checking*, as well as an auxiliary component called the *edge-blocking component*. The setting-and-fan-out component assigns to each variable a value, making two copies of the variable and its negation. This component includes two edge-blocking components whose purpose is to block an edge by using only one of the edge's endpoints in its pairs. The conveyor component transports the value of a Boolean variable or its negation from one area in the n-cube to another. The clause-checking component makes sure that a clause is satisfied if at least one of its literals has the value true.

Figure 1 depicts the setting-and-fan-out component that assigns values to two copies of a Boolean variable and its complement. The construction consists of a length three pair $(s', t') = (000, 111)$ and two blocking edges $(t'', s') = (010, 000)$ and $(s'', t') = (101, 111)$. We explain in subsection 2.2.1 how we can force an edge to be a blocking edge by using an edge-blocking component. This is not a trivial matter because nodes 000 and 111 are already occupied by a setting-and-fan-out component pair. Note that when excluding the blocking edges there are only two possible edge disjoint shortest paths for the pair: $000 \leftrightarrow 100 \leftrightarrow 110 \leftrightarrow 111$ and $000 \leftrightarrow 001 \leftrightarrow 011 \leftrightarrow 111$. The construction provides two consistent copies of the value of the variable and its negation. In particular, when the edges $(000, 100)$ and $(110, 111)$ are in the path for pair $\{000, 111\}$ the Boolean variable u has the value false, and when the edges $(000, 001)$ and $(011, 111)$ are in the path for the pair then the variable u has the value true. The reason one needs only two copies of each value of a variable and its negation is that we are reducing from the L3-SAT problem which has the property that no literal is in more than two clauses. The edges labeled x or $\bar{x}$ in Figure 1 will be used later on by the conveyor components to transmit to the clause-checking component the value assigned to the variable.

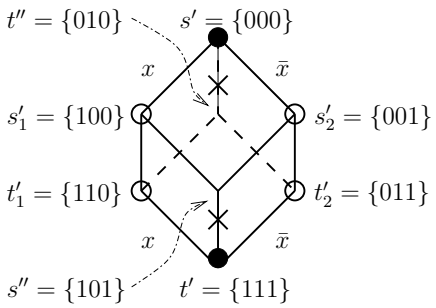

Figure 1: Setting-and-fan-out component. The symbol $\times$ represents a blocked edge.

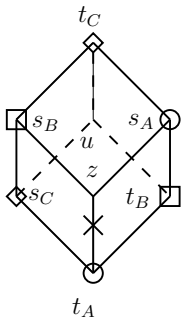

Figure 2: Edge-blocking component. The symbol $\times$ represents a blocked edge.

2.2.1 The Edge-Blocking Component

To complete the functionality of the setting-and-fan-out component we need a component to block an edge without using both of its vertices to do it.

Consider the edge-blocking component given in Figure 2. The routing request is given by $\{(s_A, t_A), (s_B, t_B), (s_C, t_C)\}$. We claim that the only possible routing must use the edge between nodes z and t_A. When the route $s_A \leftrightarrow z \leftrightarrow t_A$ is selected for the pair (s_A, t_A) then the following edge disjoint shortest paths exist for all the pairs in the 3-cube.

$$\begin{array}{lcl} s_A \leftrightarrow & z & \leftrightarrow t_A \\ s_B \leftrightarrow & t_C \leftrightarrow s_A & \leftrightarrow t_B \\ s_C \leftrightarrow & u & \leftrightarrow t_C \end{array}$$

On the other hand, when the path $s_A \leftrightarrow t_B \leftrightarrow t_A$ is used for pair (s_A, t_A), then the path for (s_B, t_B) must be either

$$s_B \leftrightarrow t_C \leftrightarrow u \leftrightarrow t_B$$

or

$$s_B \leftrightarrow s_C \leftrightarrow u \leftrightarrow t_B.$$

Since the possible paths for the pair $\{s_C, t_C\}$ are either $s_C \leftrightarrow u \leftrightarrow t_C$ or $s_C \leftrightarrow s_B \leftrightarrow t_C$, it then follows that there are no edge disjoint paths for the edge-blocking component, because each path for the pair (s_B, t_B) has an edge in common with each of the possible paths for (s_C, t_C).

Therefore, there are edge disjoint paths for all the pairs in an edge-blocking component if, and only if, the edge between node z and t_A is used by the path for the $\{s_A, t_A\}$ pair.

2.2.2 Joining the Edge-Blocking Component with the Setting-and-Fan-out Component

Now we need to incorporate two edge-blocking components (denoted by $(z, s_A, t_A, s_B, t_B, s_C, t_C, u)$ and $(z', s'_A, t'_A, s'_B, t'_B, s'_C, t'_C, u')$) with a setting-and-fan-out component. The setting-and-fan-out component is defined using the bits $F_0G_0H_0$ for the 3-cube in Figure 1. The two edge-blocking components are defined by means of bits G_0, F_1, and G_1, while keeping $F_0 = H_0 = 0$ for one edge-blocking component and $F_0 = H_0 = 1$ for the other edge-blocking component. For the nodes in the setting-and-fan-out component $F_1 = G_1 = 0$.

Each edge labeled in Figure 1 with an $\times$ symbol is coincident with the edge labeled $\times$ in one of the two edge-blocking components. More specifically, the edge (s', t'') is the same as (z, t_A), and edge (t', s'') is (z', t'_A). The setting-and-fan-out component has $s' =$ 000 00, $t'' =$ 010 00, $s'' =$ 101 00 and $t' =$ 111 00, where the first three bits are $F_0G_0H_0$ and the remaining two bits are F_1 and G_1. The edge-blocking components are defined by the as $(z = s', s_A, t_A = t'', s_B, t_B, s_C, t_C, u) =$ (000 00, 000 01, 010 00, 000 10, 010 01, 010 10, 000 11, 010 11) and $(z' = t', s'_A, t'_A = s'', s'_B, t'_B, s'_C, t'_C, u') =$ (111 00, 111 01, 101 00, 111 10, 101 01, 101 10, 111 11, 101 11). Bits F_0 and H_0 form a distinct signature that guarantees that both of the edge-blocking components are distinct. The two bits F_1 and G_1 make the edge-blocking components different from the setting-and-fan-out components, except at the two nodes where they overlap. Figure 3 shows the resulting 5-cube. The outermost dash-line quadrilateral and its interior form the setting-and-fan-out component 3-cube with the solid edges being the edges labeled $\times$ in Figure 1. The two ovals and solid dark lines on the top side of the figure represent the edge-blocking component $(z, s_A, \ldots, u)$, and the corresponding objects on the bottom side of the figure represent the edge-blocking component $(z', s'_A, \ldots, u')$.

2.2.3 The Clause-Checking Component

The clause-checking component is a single pair (s, t) in a 3-cube with edges (b_1, t), (b_2, t) and (b_3, t) which may be included in a path from either a clause-checking pair or another pair (conveyor pair). For clause checking components representing clauses with two literals there is no (b_3, t) edge. Clearly, there is no feasible shortest path from s to t if all the edges of t are in paths for pairs that are not the clause-checking pair. Note that $t = 111$ and $s = 000$ in Figure 4(a) and $t = 11$ and $s = 00$ in Figure 4(b). In other words t has one more bit set to one than b_1, b_2, or b_3.

2.2.4 The Conveyor Component

The conveyor component is used for connecting setting-and-fan-out components to the appropriate clause-checking components. At most three different conveyor components will be joined to a setting-and-fan-out component.

First we outline the basic idea and then indicate why the components do not interfere with each other. Figure 5 illustrates the basic construction. The conveyor consists of the pairs (s_1, t_1),

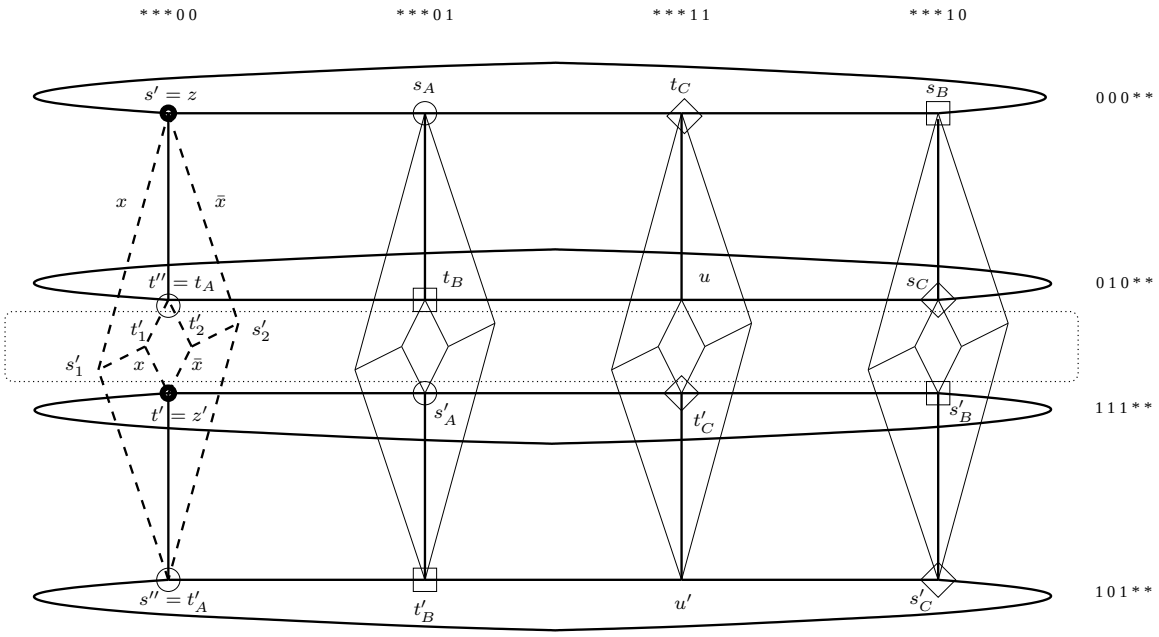

Figure 3: The 5-cube composed of one setting-and-fan-out component and two edge-blocking components.

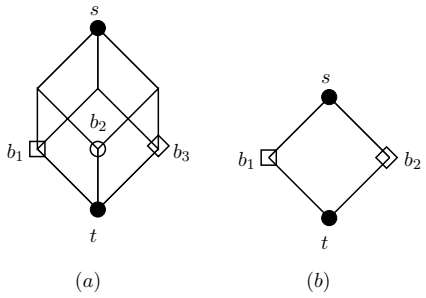

Figure 4: Clause checking component for clauses with three (a) and two (b) literals.

(s_2, t_2), ... (s_{l+r}, t_{l+r}), where r and l will be defined later on. The conveyor operates as follows. If the edge labeled A in Figure 5 is covered by a path for a non-conveyor pair, then it will always be the case that the edge labeled B will be covered by a path for the conveyor pair (s_{l+r}, t_{l+r}). However, if the edge labeled A is not covered by a path for a non-conveyor pair, then the edge labeled B may or may not be covered by the path for the conveyor pair (s_{l+r}, t_{l+r}).

The conveyor components join setting-and-fan-out components to clause-checking components as follows. The edge labeled A in the conveyor component (Figure 5) will be the same as one of the edges labeled x or $\bar{x}$ in the setting-and-fan-out component (Figure 1). That is, (u_0, t_1) will be the same as (s', s'_1), (s', s'_2), (t', t'_1) or (t', t'_2). The edge from t to b_1, b_2, or b_3 in the clause-checking components will be the same as the edge labeled B in a conveyor component (Figure 5). That is, (u_1, t_{l+r}) will be the same as (t, b_1), (t, b_2), or (t, b_3).

Consider the following bit numbering for the n-cube nodes.

$$D_0 D_1 \ldots D_{\lceil \log_2(v+1) \rceil} \;\; E_0 E_1 \ldots E_{\lceil \log_2(w+1) \rceil} \;\; F_0 G_0 H_0 \; F_1 G_1 \; \alpha_1 \alpha_2 \;\; JKL \; \beta_1 \beta_2 \beta_3$$

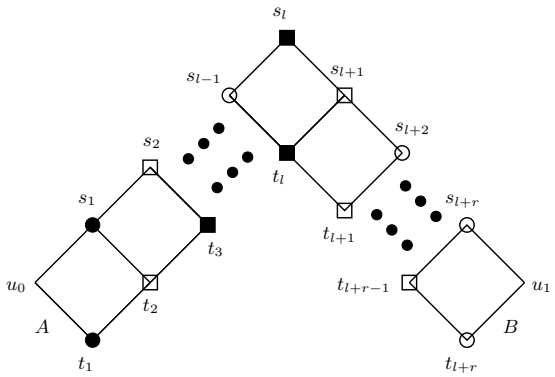

Figure 5: Conveyor component.

The bits $D_0D_1 \dots D_{\lceil \log_2(v+1) \rceil}$ represent the value of i, the index of the i^{th} variable u_i. The j^{th} clause-checking component will be set, using the bits, $E_0E_1 \dots E_{\lceil \log_2(w+1) \rceil}$. The setting-and-fan-out component plus its two corresponding 3-cube edge-blocking components use the bits $F_0G_0H_0\ F_1G_1$. The clause-checking component will use the bits JKL (3-cube or 2-cube as indicated in Figure 4). The bits α and β are used in the conveyor component and identify the edge in the i^{th} setting-and-fan-out component and the edge in the j^{th} clause-checking component that the conveyor component joins.

For each literal in a clause in the instance of L3-SAT that we start from, we make an association between the corresponding edges in the setting-and-fan-out and the clause checking components in such a way that the edges involved in the association are unique. An edge in the i^{th} setting and fan out component is defined by (y, i'), where $y \in \{s, t\}$ and $i' \in \{1, 2\}$, and it represents the edge $(y', y'_{i'})$. For example, $(y = s, i' = 1) \rightarrow (y' = s', y'_{i'} = s'_1)$, $(y = s, i' = 2) \rightarrow (y' = s', y'_{i'} = s'_2)$, $(y = t, i' = 1) \rightarrow (y' = t', y'_{i'} = t'_1)$, and $(y = t, i' = 2) \rightarrow (y' = t', y'_{i'} = t'_2)$, where $\rightarrow$ means "represents the edge". An edge in the j^{th} clause checking component is defined by (j'), where $j' \in \{1, 2, 3\}$, and it represents the edge $(t, b_{j'})$. For example, $(j' = 1) \rightarrow (t, b_{j'} = b_1)$, $(j' = 2) \rightarrow (t, b_{j'} = b_2)$, and $(j' = 3) \rightarrow (t, b_{j'} = b_3)$. Once the associations are defined, we introduce a conveyor component for every such association.

Before we describe in detail the conveyor component we need to introduce additional notation to represent bit strings. In what follows ellipses in bit representations indicate a sequence of zero or more bits all of which are zeroes. For *bit* α_i or β_j we use *bitone*(*bit*) to represent a bit with value one in the appropriate position in the bit string. By *bitrep*(*value*), where *value* is i or j, we mean the binary representation of *value* and its length is defined by $\lceil \log_2(v+1) \rceil$ or $\lceil \log_2(w+1) \rceil$ depending on whether it represents a setting-and-fan-out or a clause checking component. These bits are located at the appropriate position in the bit string. Similarly, *bitrep*(y') and *bitrep*($y'_{i'}$) (resp. *bitrep*(t) and *bitrep*($b_{j'}$)) represents a vertex in the 5-cube $F_0G_0H_0F_1G_1$ (resp. 3-cube JKL). The sequence is five (resp. three) bits long and it is located at the appropriate position in the bit string.

In what follows we discuss in detail the conveyor component for the association between the edge $(y', y'_{i'})$ in the i^{th} setting-and-fan-out component and edge $(t, b_{j'})$ in the j^{th} clause-checking component. The conveyor component consists of three sections: first, middle and last. The first section joins to a setting-and-fan-out component and the last one joins to a clause checking component.

The first section starts with the edge (u_0, t_1) which is identical to $(y', y'_{i'})$. Clearly, the difference

between nodes u_0 and t_1 is in one bit in the $bitrep(y')$ and $bitrep(y'_{i'})$. It is important to note that the number of ones in $bitrep(y')$ is either one more or one fewer than those in $bitrep(y'_{i'})$. The transition from (u_0, t_1) to (s_1, t_2) is by setting to one $\alpha_{i'}$.

$$\begin{aligned} u_0 &= bitrep(i) \ldots bitrep(y') \ldots \\ t_1 &= bitrep(i) \ldots bitrep(y'_{i'}) \ldots \\ s_1 &= bitrep(i) \ldots bitrep(y') \ldots bitone(\alpha_{i'}) \ldots \\ t_2 &= bitrep(i) \ldots bitrep(y'_{i'}) \ldots bitone(\alpha_{i'}) \ldots \end{aligned}$$

The transition (s_1, t_2), (s_2, t_3), ..., (s_{l-1}, t_l) is by changing a 0-bit to a 1-bit in such a way that we incorporate $bitrep(j)$, $bitrep(b_{j'})$ and $bitone(\beta_{j'})$ into the bit strings. Clearly, the difference between the pairs of nodes (s_1, t_2), (s_2, t_3), ..., and (s_{l-1}, t_l) is in one bit in the $bitrep(y')$ and $bitrep(y'_{i'})$. By construction we know that the value of l is the total number of ones in $bitrep(j)$, $bitrep(b_{j'})$ and $bitone(\beta_{j'})$ plus 2.

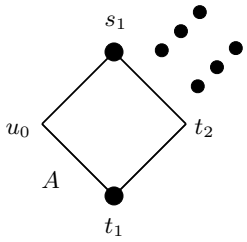

Figure 6: First section of the conveyor component (including u_0, t_1, s_1 and t_2).

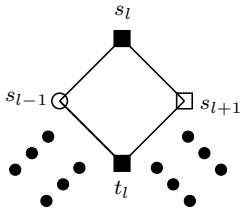

Figure 7: Middle section of the conveyor component (including nodes s_{l-1}, t_l, s_l and s_{l+1}).

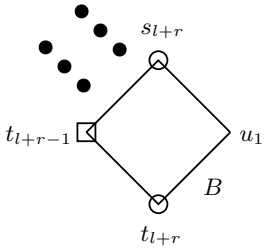

Figure 8: Last section of the conveyor component (including nodes t_{l+r-1}, s_{l+r}, t_{l+r} and u_1).

The middle section consists of the transition from (s_{l-1}, t_l) to (s_l, s_{l+1}). By definition, $bitrep(t)$ has one more bit than $bitrep(b_{j'})$. The only transition in this middle section changes $bitrep(b_{j'})$ to $bitrep(t)$.

$$\begin{aligned} s_{l-1} &= bitrep(i)bitrep(j)bitrep(y') \ldots bitone(\alpha_{i'}) \ldots bitrep(b_{j'}) \ldots bitone(\beta_{j'}) \ldots \\ t_l &= bitrep(i)bitrep(j)bitrep(y'_{i'}) \ldots bitone(\alpha_{i'}) \ldots bitrep(b_{j'}) \ldots bitone(\beta_{j'}) \ldots \\ s_l &= bitrep(i)bitrep(j)bitrep(y') \ldots bitone(\alpha_{i'}) \ldots bitrep(t) \ldots bitone(\beta_{j'}) \ldots \\ s_{l+1} &= bitrep(i)bitrep(j)bitrep(y'_{i'}) \ldots bitone(\alpha_{i'}) \ldots bitrep(t) \ldots bitone(\beta_{j'}) \ldots \end{aligned}$$

In the last section of the conveyor component the difference between the two nodes in (t_l, s_{l+1}), (t_{l+1}, s_{l+2}), ..., (t_{l+r-1}, s_{l+r}) and (t_{l+r}, u_1) is that the first element of the pair has $bitrep(b_{j'})$ and the second one has $bitrep(t)$.

Each transition from pair (t_l, s_{l+1}) to pair (t_{l+r-1}, s_{l+r}) will have one fewer bit set to one in the portion $bitrep(i)$, $bitrep(y'_{i'})$, and $bitone(\alpha_{i'})$. The last transition from, (t_{l+r-1}, s_{l+r}) to $(t_{l+r}$, $u_1)$, sets $bitone(\beta_{j'})$ to zero. So vertices t_{l+r} and u_1 correspond to the edge $(b_{j'}, t)$ in the j^{th} clause-checking component.

By construction we know that the value of r is the total number of ones in $bitrep(i)$, $bitrep(y'_{i'})$ and $bitone(\alpha_{i'})$ plus 1.

$$\begin{aligned} t_{l+r-1} &= \ldots bitrep(j) \ldots bitrep(b_{j'}) \ldots bitone(\beta_{j'}) \ldots \\ s_{l+r} &= \ldots bitrep(j) \ldots bitrep(t') \ldots bitone(\beta_{j'}) \ldots \\ t_{l+r} &= \ldots bitrep(j) \ldots bitrep(b_{j'}) \ldots \\ u_1 &= \ldots bitrep(j) \ldots bitrep(t) \ldots \end{aligned}$$

The above construction defines values for l and r. Note that l and r are normally different, and conveyor components normally have different values for l as well as different values for r.

Lemma 1 *The conveyor component which connects setting-and-fan-out component i and clause-checking component j does not overlap with any other conveyor component and nodes u_1, u_2, s_1, s_2, ..., s_{l+r}, t_1, t_2, ..., t_{l+r} of this conveyor component are unique.*

Proof: Besides the nodes incident to the edges A and B in the conveyor components, no other nodes in the conveyor components belong to more than one conveyor component nor a node is more than once in the same conveyor component. The reason for this is that in the first part of the conveyor component $((s_1, t_2), (s_2, t_3), \ldots, (s_{l-1}, t_l), (s_l, s_{l+1}))$ every node has either the signature

$$bitrep(i) \;\; bitrep(y'_{i'}) \;\; bitone(\alpha_{i'})$$

or

$$bitrep(i) \;\; bitrep(y') \;\; bitone(\alpha_{i'})$$

These signatures are unique for every conveyor. All of these nodes in the same conveyor component are unique because each one in the sequence has one additional bit set to one. The nodes t_1 and u_0 overlap with a pair of nodes of the setting-and-fan-out components. These nodes are unique in this conveyor component and both of the nodes are not assigned to another conveyor component.

In the second part of a conveyor component $((s_{l-1}, s_l), (t_l, s_{l+1}), (t_{l+1}, s_{l+2}), \ldots, (t_{l+r-1}, s_{l+r}))$ all the nodes have the signature

$$\ldots bitrep(j) \;\; bitrep(b_{j'}) \;\; bitone(\beta_{j'})$$

or

$$\ldots bitrep(j) \;\; bitrep(t) \;\; bitone(\beta_{j'}).$$

These signatures are unique for every conveyor. All of these nodes in the same conveyor component are unique because each one in the sequence has one fewer bit set to one. The only exception may be (s_{l-1}, s_l) and (t_{l+1}, s_{l+2}). This occurs when $bitrep(y')$ has one fewer bit set to one than $bitrep(y'_{i'})$. But these pairs of nodes differ in that the first pair has $bitone(\alpha_{i'})$ but the second pair does not. The nodes t_{l+r} and u_1 overlap with a pair of nodes of the clause checking components. These nodes are unique in the conveyor component and both of the nodes are not assigned to another conveyor component.

Every node in the first part of the conveyor component is different from ones in the second part of other conveyor components because of the uniqueness of these signatures. This completes the proof of this lemma.

□

Theorem 2 *The p-pairwise edge disjoint shortest paths problem for the n-cube is NP-complete.*

Proof: For any pair $X_i = \{s_i, t_i\}$ the maximum number of edges in a shortest path is n. Given a set of paths for a routing request one can check in polynomial time that every two edges in the paths are distinct and that the paths are indeed shortest paths for the pairs. Therefore the problem is in NP.

By the previous discussion we note that only a polynomial number of pairs are required in our polynomial transformation from L3-SAT to our problem. By the discussion just before the theorem it is clear that the instance of the p-pairwise edge disjoint shortest paths problem has a solution if and only if the instance of L3-SAT that we start from is satisfiable. Therefore the problem is NP-complete.

□

It is simple to see that all the pairs in the p-pairwise edge disjoint shortest paths problem constructed by the above reduction are such that their pair distance at most three. Therefore, we have the following corollary.

Corollary 1 *The p-pairwise edge disjoint shortest paths problem for the n-cube is NP-complete even when* $d(X_i) \leq 3$, *for* $1 \leq i \leq p$.

We have not been able to establish that the pairwise edge disjoint arbitrary length paths problem is NP-complete. The main problem is that we lack appropriate edge blocking components. We conjecture that this problem is NP-hard.

3 Conclusions

In this paper we examined complexity issues regarding a high performance computing topology, namely the n-cube. For the p-pairwise edge disjoint shortest paths problem in the n-cube the distance at most two pair problem is found to be solvable in polynomial time (Section 2.1). When the pair distance is at most three we established intractability (Section 2.2). The corresponding node disjoint shortest paths problem and related problems, even in the context of an approximation to the shortest paths are also computationally intractable [5].

Acknowledgments

We like to thank the referees for pointing out mistakes in earlier versions of our paper and for their suggestions on ways to improve the readability of our paper.

References

[1] Garey, M. R., and Johnson, D. S. *Computers and Intractability A Guide to the Theory of NP-Completeness.* Freeman, 1979.

[2] Gonzalez, T. F., and Serena, F. D. Complexity of k-Pairwise Disjoint Shortest Paths in the Hypercube and Grid Networks. Tech. Rep. TRCS-2002-14, University of California at Santa Barbara, May 2002.

[3] Gonzalez, T. F., and Serena, F. D. Complexity of Pairwise Shortest Path Routing in the Grid. *Theoretical Computer Science 326* (2004), 155 – 185.

[4] Gonzalez, T. F., and Serena, F. D. n-Cube Network: Node Disjoint Shortest Paths for Maximal Distance Pairs of Vertices. *Parallel Computing 30* (2004), 973 – 998.

[5] Gonzalez, T. F., and Serena, F. D. Pairwise disjoint shortest paths in the n-cube and related problems. Tech. rep., CS Technical Report 2006-04, UCSB, 2006.

[6] Lubiw, A. Counterexample to a Conjecture of Szymanski on Hypercube Routing. *Information Processing Letters 35* (June 1990), 57–61.

[7] Madhavapeddy, S., and Sudborough, I. H. Disjoint Paths in the Hypercube. In *WG* (June 1990), M. Nagl, Ed., vol. 411 of *Lecture Notes in Computer Science. Graph-Theoretic Concepts in Computer Science, 15th International Workshop, WG '89*, pp. 3–18.