Optimal Permutation Routing for Low-dimensional Hypercubes

Ambrose K. Laing and David W. Krumme

Electrical Engineering and Computer Science Department

Tufts University

161 College Ave.,

Medford, MA 02155

{laing,krumme}@eecs.tufts.edu

July 11, 2003

Abstract

We consider the offline problem of routing a permutation of tokens on the nodes of a d-dimensional hypercube, under a queueless MIMD communication model (under the constraints that each hypercube edge may only communicate one token per communication step, and each node may only be occupied by a single token between communication steps). For a d-dimensional hypercube, it is easy to see that d communication steps are necessary. We develop a theory of “separability” which enables an analytical proof that d steps suffice for the case $d = 3$, and facilitates an experimental verification that d steps suffice for $d = 4$. This result improves the upper bound for the number of communication steps required to route an arbitrary permutation on arbitrarily large hypercubes to $2d - 4$. We also find an interesting side-result, that the number of possible communication steps in a d-dimensional hypercube is the same as the number of perfect matchings in a $(d + 1)$-dimensional hypercube, a combinatorial quantity for which there is no closed-form expression. Finally we present some experimental observations which may lead to a proof of a more general result for arbitrarily large dimension d.

1 Introduction

In this paper, we consider the offline problem of routing a permutation of tokens on the nodes of a d-dimensional hypercube, under the constraints that each node starts with one token and ends with one token. We use a queueless MIMD communication model, in which each hypercube edge may only communicate one token per communication step, and each node may only be occupied by a single token between communication steps. Each hypercube edge is considered to be bidirectional and therefore exchanges of tokens between adjacent nodes are possible in one communication step. The goal is to minimize the number of steps of communication required to realise an arbitrary permutation. Clearly at least d communication steps are required. This paper considers the conjecture that d communication steps suffi ce for routing arbitrary permutations on arbitrarily large hypercubes.

Our experimental results indicate that it is possible to route arbitrary permutations in d communication steps for $d = 4$. Our techniques use combinatorial properties to signifi cantly reduce the permutation space that needs to be checked computationally, by a factor of hundreds of thousands. These techniques are powerful enough provide a human-checkable proof for the case $d = 3$, as further confi rmation of a computational verifi cation by Cooperman [3, 28]. By Ramras' lemma, we therefore improve the best known result for the upper bound in arbitrarily large hypercubes to $2d - 4$ communication steps.

2 Related Work

This is a well-studied problem, with several variants. The following discussion of terminology derives from Grammatikakis et al [6]. We consider the *offline* version of the problem in which the permutation to be routed is given in advance. One processing unit has access to the entire permutation and predetermines the paths to be travelled by each token, prior to the computation. In contrast, the *online* version of the problem restricts that the permutation is not known in advance, and during any step, each processing unit in the interconnection network only has access to its memory and that its local neighbors.

The *circuit-switched* model requires that the path of edges travelled by a token from its source to its destination be all held simultaneously and exclusively by that token. Intermediate nodes along the path may be common to different source-destination paths, but each directed arc in an edge (considered as a pair of opposite directed arcs) must be exclusively held. This contrasts with the *packet-switched* model in which a token is communicated along each successive edge in the path in a subsequent communication step.

Some early hypercube designs were constrained such that communication could only occur along edges of the same dimension in a given communication step. This is sometimes referred to as a *synchronous* communication model, and computers that operate under this model are single-instruction-stream, multiple-data-stream (SIMD) computers. We note that this is also referred to in the literature by Grammatikakis et al as single-port routing [6].

Without the restriction that all communications in a given step must occur in the same dimension, one obtains a variety of models for a multiple-instruction-stream, multiple-data-stream (MIMD) computer.

For example, a more relaxed model has also been studied extensively which allows any number of dimensions in the hypercube to be used in a single communication step, but requires that each node either abstain from communication or else send and receive along the same dimension in each time step. This only permits exchanges of packets with adjacent nodes, and it follows that each node is only occupied by a single packet between communication steps. The routing theory under this model is dependent on matchings (in the graph-theoretical sense). We modify the terminology of Grammatikakis et al [6] in referring to this as the *single-port MIMD* model.

The model we study is next in level of generality. It is identical to the previous single-port MIMD model, except that there is no restriction that packets are sent and received on the same dimension by communicating nodes. Again only one packet may be received by each node and one packet transmitted by each node during a communication step, and each node can only contain a single packet between communication steps. If each node is considered to also have a communication link pointing to itself, then this model can very simply be characterized as requiring each node to send exactly one packet and receive exactly one packet during each communication step. We call this a *queueless MIMD* routing model because each node needs only a queue of size 1 to hold packets. We also note that this model is called *congestion-free* routing by Ramras [28, 27, 29].

The next model does not have the restriction that a node may only send and receive a single packet in a given timestep. Instead, a node may send tokens to all its neighbors and simultaneously receive tokens from all its neighbors. However it retains the restriction that a packet has to leave a node on the next communication step after the one during which it arrived. We call this an *all-port MIMD* communication model. Note that in the context of circuit-switching, all-port communication has a different but consistent meaning – it means that a node can be intermediate on many source-destination paths.

A fi nal model is one in which each node may receive and send packets on all dimensions simultaneously in a communication step, and a node may hold on to a packet for sending later. Buffers could be associated with the node itself, or else they could be associated with the edges of the node. A node could then contain more than d packets at a time, where d is the degree of the node. We call this the *buffered all-port MIMD* model.

Szymanski studies the offline circuit-switched all-port version of this problem [35], in contrast to our consideration of the packet switched model. Under this model he shows that for $d \leq 3$, the hypercube is *rearrangeable*, meaning that any permutation can be routed [35]. Szymanski conjectures that the d-dimensional hypercube is rearrangeable. In general, a routing under the circuit-switched model automatically provides a routing under the all-port MIMD packet-switched model but not the queueless MIMD packet-switched model since it may put multiple packets at an intermediate node at the same time. Szymanski notes that a proof of his rearrangeability conjecture for $d = 4$ appears to be challenging. Szymanski also had a still stronger conjecture, that rearrangeability is possible with shortest paths. This still stronger version was disproven by Lubiw with a 5-dimensional counterexample [19]. In passing we note that Szymanski's routings do not work for the queueless MIMD packet-switched model. Consider a 3-dimensional hypercube with a permutation in which packets at node 0 and node 7 interchange positions, and every other node stays put. Szymanski's routing sends every packet along a shortest path, which means packets in nodes 1–6 would not move, whereas some must move under the queueless MIMD packet-switched model.

Szymanski's original rearrangeability conjecture stands. He also notes that for the all-port MIMD packet-switched model, routing is possible in $2d - 1$ steps [35]. In the same model, the result was further improved by Shen et al to $2d - 3$ steps as a corollary of their main result, namely an explicit algorithm for routing arbitrary permutations on the circuit-switched hypercube such that each directed edge is used at most twice [32].

In the more restrictive queueless MIMD model, Ramras uses a bound on the optimal routing time of a cross-product graph in terms of its factors (Theorem 1), due Annexstein et al [2], to effectively prove that if permutation routing for the r-hypercube is possible in r steps, then for $d \geq r$, permutation routing is possible in $2d - r$ steps [28]. In our routing model, the result for $d = 3$ was verified computationally by Cooperman [3]. Ramras therefore establishes that permutation routing is possible in $2d - 2$ steps, based on the analytical proofs, and $2d - 3$ steps, taking into account the computational result by Cooperman. Note that this is an explicit queueless MIMD result, which uses stronger restrictions than the all-port MIMD result of Shen et al [32].

Alon et al [1] also consider the offline hypercube permutation routing, under the single-port MIMD model, where packets may only be exchanged with adjacent nodes. Under this model, they also consider many other different interconnection networks. For hypercubes in particular, they prove that routing is not possible in d steps for $d \in \{2, 3\}$. Thus their model appears to be too strong for the sort of result we seek. Further work by Høyer and Larsen in the same model considers arbitrary graphs, and shows that arbitrary permutations can be routed in $2n - 3$ steps, where n is the number of nodes [8, 7]. Zhang also proves that under this model in which only edge exchanges are allowed, routing can be done on a tree (and therefore for any graph) in $\frac{3}{2} + O(\log n)$ steps, where n is the number of nodes [38].

In the online version of the buffered all-port MIMD model, Hwang et al have shown that oblivious routing is possible in d steps for $d \in \{2, \ldots, 7\}$, using only local information [10]. In another paper [9] this is improved for up to $d \leq 12$. More recently, Vöcking obtained some very interesting results in the buffered all-port MIMD model [37]. He presents an online oblivious randomized algorithm that routes in $d + O(d/\log d)$ steps, and a matching lower bound. He also obtains a deterministic offline algorithm in the buffered all-port MIMD model that routes in $d + O(\sqrt{d \log d})$ steps with three buffers per edge. They are particularly interesting because these are the fi rst results that use $d + o(d)$ steps.

Permutation Routing has been studied in other interconnection networks with different communication paradigms [6]. There is a very wide variety of results for on-line routing. An early result was for the more general problem of sorting on a hypercube, and this was shown to be possible for the n-node hypercube in $O(\log n(\log\log n)^2)$ time in the worst case by Cypher and Plaxton [4]. Kaklamanis et al also proved an $\Theta(\sqrt{n}/\log n)$ time bound for deterministic oblivious routing on the n-node hypercube hypercube [14]. Newman and Schuster extend the store-and-forward paradigm for permutation routing to hot-potato routing for meshes, hypercubes and tori [22]. Leighton et al were the fi rst to show that it is possible to route in a two-dimensional mesh using the minimum of $2n - 2$ communication steps where n is the side of the mesh, using constant-sized communication queues [16]. Sibeyn et al also study the two-dimensional mesh, and further minimized the number of queues required. One of their algorithms routes in the optimal number of communication steps, using queues of size 32. The other algorithm improves the queue size to 12 by trading off the optimal number of communication steps by a constant [33]. Kaufmann and Sibeyn also study the $k-k$-routing problem, a variant in which each node starts and ends with k packets. In the arbitrary-dimensional mesh, they used randomization to prove that it is possible in $(1 + o(1))max\{2dn, \frac{1}{2}kn\}$ communication steps, where d is the dimension and n is the side of the mesh. Their results can be modifi ed to tori, and the communication model can be modified to the cut-through routing model [15].

Some of the preceding results have proven routing properties for all permutations on hypercubes of bounded dimension, under different communication models. An alternative approach has also been to consider routing properties of hypercubes of arbitrary dimension, but for restricted classes of permutations, again under different communication models.

An important subclass of permutations for general-purpose computing is the class of bit permute complement (BPC) permutations, which are useful for matrix transposing, vector reversal, bit shuffling and perfect shuffling [21]. Nassimi and Sahni present a routing scheme for routing BPC permutations on SIMD hypercubes in d steps, which is optimal [21]. This may be compared with the bound of $2d-1$ steps for an arbitrary permutation on the SIMD hypercube [36]. Johnsson and Ho present algorithms for optimal matrix transposition in the all-port MIMD model [11]. This is further developed to generalized shuffle permutations in the queueless and all-port MIMD models [12]. Generalized shuffle permutations are a more general form of BPCs which operate on an array of data associated with a large virtual hypercube, which is mapped onto a smaller real hypercube [12]. The BPC therefore moves data between nodes as well as changing positions of items within a node. Further work by the same authors in the all-port MIMD model considers the problem of all-to-all personalized communication [13].

Ramras improves the optimal routing results for BPCs by proving that hypercube automorphisms (which are the same as BPCs) can be routed in d steps under the more restricted queueless MIMD model [27]. He also proves an analogous result for linear complement permutations on hypercubes of arbitrary dimension [29].

Lin considers the problem of manipulating general vectors on hypercubes and provides optimal algorithms in the synchronous (SIMD) model [18]. He defines general vectors as starting from an arbitrary node and having an arbitrary length. He provides for operations such as merging, splitting, rotation, reversal, compression and expansion (only some of which are permutations).

Given an array which has been mapped into the hypercube, the cyclic shift permutations cyclically shift the elements of the array by a given offset. Assuming that the array is mapped into the hypercube under the natural ordering which maps array cell i to hypercube node i, Ranka and Sahni have shown that in the all-port MIMD model, this form of permutation routing is possible in d steps [30]. Under the binary reflected Gray code (BRGC) order, which maps the array into a Hamiltonian path of the hypercube defined by the Gray code, Ouyang and Palem have shown that routing of cyclic shifts is possible in $\frac{4}{3}d$ steps in the all-port

MIMD model [24].

In the SIMD model, certain permutations such as the shift permutation have been shown to be routable in an optimal number of synchronous communication steps, in a way that each node always contains at most one packet between communication steps, and such that the dimension exchanges can occur in any order [25]. Under this model, Panaite identifi es a large class of hypercube permutations, called frequently used bijections (FUBs), which are similarly routable, including admissible lower triangular and admissible upper triangular permutations, and p-orderings with cyclic shifts. The motivation for this SIMD model is to be able to perform a group of up to d permutation routings simultaneously, without any requirement for local buffering of messages in intermediate nodes during a communication step [25].

Given the above results in their totality, especially those of Szymanski, Ramras, Hwang et al, and Vöcking, we are cautiously optimistic about the following conjecture which is the subject of the remainder of this paper.

Conjecture *Under the packet-switched queueless MIMD communication model, an arbitrary permutation on the d-dimensional hypercube can be offline-routed in d steps.*

3 Notation

Definition 3.1 *We denote the bounded set of nonnegative integers $\{0, \ldots, n-1\}$, which has n elements, by $[n]$.*

Definition 3.2 *Let u and v be nonnegative integers. We define $|u, v|$ as the number of bits that are different in the binary representations of u and v. Note that there is no ambiguity by omitting the number of bits in the binary representation if we assume that leading blank spaces are all zero.*

Definition 3.3 *We define the d-dimensional hypercube Q_d as the graph $([2^d], \{(u, v) : |u, v| = 1\})$.*

Definition 3.4 *S_N denotes the symmetric group of order N, the group of all permutations on the set $[N]$.*

Definition 3.5 *Let Δ_d be $\{\delta | \delta \in S_{2^d}$ and $\forall i \in [2^d], |i, \delta(i)| \leq 1\}$.*

We call Δ_d the set of *allowable* permutations, alluding to the fact that they are the set of permitted communication steps.

Definition 3.6 *Let $\phi \in S_{2^d}$. The Q_d permutation routing problem is to determine a sequence $\delta_0, \ldots, \delta_{k-1}$ of permutations in Δ_d such that $\phi = \delta_{k-1} \circ \ldots \circ \delta_0$. Such a finite sequence is known to exist [28]. If there exists a finite sequence of length k, then we say ϕ is k-factorable.*

Definition 3.7 *We will also say S_{2^d} is k-factorable if every $\phi \in S_{2^d}$ is k-factorable.*

Given these definitions, we may restate the main conjecture considered in this paper as follows:

Conjecture 3.8 *For all d, S_{2^d} is d-factorable.*

It has been shown by Ramras that the conjecture is true for $d = 2$ – every permutation in S_{2^2} is 2-factorable [28], and it has been verified by a computer program (reported in a private communication from Cooperman to Ramras[28]) that the conjecture holds for $d = 3$ [28].

We present an approach which simplifies the problem enough to permit the human verification of the case for $d = 3$ and to to permit a computer verification of the case for $d = 4$. It basically tries to use a recursive algorithm which converts a problem for dimension d into two disjoint problems of dimension $d - 1$ where possible, and verifies the exceptions explicitly. We develop this approach as follows:

Definition 3.9 *Let $b_j(v) =$ bit j of v.*

Definition 3.10 *Let $B \subseteq [d]$, where $|B| = t$. A permutation $\phi \in S_{2^d}$ is t-separable by B if there exist permutations $\delta_0, \ldots, \delta_{t-1} \in \Delta_d$ and β such that*

- $\phi = \beta \circ \delta_{t-1} \circ \ldots \circ \delta_0$.
- $\forall i \in [2^d], j \in B \Rightarrow (b_j(i \oplus \beta(i)) = 0)$,

This captures the idea that after the first t communication steps, no motion will be permitted in any of the dimensions in B (the set of dimensions that will be "broken"). Equivalently, we could say that all deliveries are completed with respect to the dimensions in B. It therefore reduces the original routing problem to 2^t separate cases of $(d - t)$-subcubes involving the dimensions $[d] \setminus B$.

Definition 3.11 *If ϕ is t-separable by B we will equivalently say ϕ is t-separable into $[d] \setminus B$.*

The following easy results highlight the relationship between the notions of separability and factorability.

Lemma 3.12 *If $\phi \in S_{2^d}$ is d-separable into $\emptyset$, then ϕ is d-factorable.*

The generalized principle of induction easily yields the following results:

Lemma 3.13 *For $t < d$, if all permutations in S_{2^t} are t-factorable and $\phi \in S_{2^d}$ is $(d-t)$-separable, then ϕ is d-factorable.*

Corollary 3.13a *If $\forall t < d$, all permutations in S_{2^t} are t-factorable and for each $\phi \in S_{2^d}$ there is a $t' < d$ such that ϕ is $(d-t')$-separable, then S_{2^d} is d-factorable.*

On applying the latter corollary with Ramras and Coopermans' results [28], we obtain the following:

Corollary 3.13b *For $d \geq 3$, if $\phi \in S_{2^d}$ is $(d-t)$-separable for some $t \leq 3$, then ϕ is d-factorable.*

The following lemma, which extends Ramras' result for $d = 2$, can be verifi ed very easily:

Lemma 3.14 *All permutations in S_{2^2} are 1-separable by D for any suitable choice of D.*

From this it follows that

Corollary 3.14a *For $d \geq 3$, if $\phi \in S_{2^d}$ is $(d-2)$-separable, then ϕ is $(d-1)$-separable, and therefore d-factorable.*

4 Characterizing 1-separability

In this section, we develop an algebraic characterization of 1-separability. This provides the basis for our treatment of Conjecture 3.8, yielding in Section 5 an analytic proof for $d = 3$ and in Section 6 an experimental verification for $d = 4$.

Definition 4.1 *For $\alpha \in S_d$, $A^\alpha : Q_d \to Q_d$ is the hypercube automorphism defined by $b_{\alpha(j)}(A^\alpha(v)) = b_j(v)$ for all $j \in [d]$ and $v \in [2^d]$.*

We will call a function like A^α a *dimension-relabelling* by α. It is easy to see that $(A^\alpha)^{-1} = A^{\alpha^{-1}}$.

Definition 4.2 *For $u \in [2^d]$, $A_u : Q_d \to Q_d$ is the hypercube automorphism defined by $A_u(v) = u \oplus v$ for all $v \in [2^d]$.*

We call a function like A_u in this definition a *zero-relocation* to node u. Note that from this definition it follows that $(A_u)^{-1} = A_u$.

Definition 4.3 *We define $A_u^\alpha = A_u \circ A^\alpha$*

The following lemma is well known [17]:

Lemma 4.4 *Every hypercube automorphism is characterized by a value $u \in [2^d]$ and $\alpha \in [S_d]$ and is obtained by applying A_u^α to the hypercube.*

We obtain the inverse of A_u^α by noting that

$$(A_u^\alpha)^{-1} = (A_u \circ A^\alpha)^{-1} = (A^\alpha)^{-1} \circ (A_u)^{-1} = A^{\alpha^{-1}} \circ A_u.$$

We also note that in general $(A_u^\alpha)^{-1} \neq A_u^{\alpha^{-1}}$. The noncommutative nature of A_u and A^α is characterized further by the following lemma:

Lemma 4.5 *For $\alpha \in S_d$ and $u \in [2^d]$, $A^\alpha \circ A_u = A_{A^\alpha(u)} \circ A^\alpha$*

PROOF For arbitrary $v \in [2^d]$, and for arbitrary dimension $j \in [d]$,

$$\begin{aligned}
b_{\alpha(j)}\left(A_{A^\alpha(u)} \circ A^\alpha(v)\right) &= b_{\alpha(j)}\left(A^\alpha(u) \oplus A^\alpha(v)\right) && \text{(by 4.2)}\\
&= b_{\alpha(j)}\left(A^\alpha(u)\right) \oplus b_{\alpha(j)}\left(A^\alpha(v)\right) \\
&= b_j(u) \oplus b_j(v) && \text{(by 4.1)}\\
&= b_j(u \oplus v) \\
&= b_{\alpha(j)}\left(A^\alpha(u \oplus v)\right) && \text{(by 4.1)}\\
&= b_{\alpha(j)}\left(A^\alpha \circ A_u(v)\right) && \text{(by 4.2)}
\end{aligned}$$

Since α is onto, all the bits of $A^\alpha \circ A_u$ and $A_{\alpha(u)} \circ A^\alpha$ are equal, thus proving the result. □

We will also need the following lemma to enable us to compose dimension relabellings:

Lemma 4.6 *For $\alpha, \beta \in S_d$, $A^\alpha \circ A^\beta = A^{\alpha\circ\beta}$*

PROOF For arbitrary $v \in [2^d]$, and for arbitrary dimension $j \in [d]$,

$$\begin{aligned}
b_{\alpha\circ\beta(j)}\left(A^\alpha \circ A^\beta(v)\right) &= b_{\alpha(\beta(j))}\left(A^\alpha \circ A^\beta(v)\right) \\
&= b_{\beta(j)}\left(A^\beta(v)\right) && \text{(by 4.1)}\\
&= b_j(v) && \text{(by 4.1)}\\
&= b_{\alpha\circ\beta(j)}\left(A^{\alpha\circ\beta}(v)\right) && \text{(by 4.1)}
\end{aligned}$$

Since α and β are both onto, $\alpha \circ \beta$ is onto, therefore all the bits of $A^\alpha \circ A^\beta$ and $A^{\alpha\circ\beta}$ are equal. □

Definition 4.7 *For $\phi : [2^d] \rightarrow [2^d]$, $\pi(\phi, j) = \{v \in V(Q_n) \mid b_j(\phi(v)) \neq b_j(v)\}$. $\pi(\phi, j)$ is called the projection of ϕ in dimension j.*

The projection of ϕ along j is the set of nodes with packets which have to cross dimension j at some point. These packets are of interest because they have to cross dimension j immediately if all movement along dimension j will be disallowed after the next computation step (as a means of recursing).

For example, consider the permutation $(065)(127)(34)$, shown leftmost in Figure 1 on page 21. Its projection in dimension 2 (leftmost) contains all nodes except 1 and 6. So $\pi((065)(127)(34), 2) = \{0, 2, 3, 4, 5, 7\}$.

Two neighboring nodes in the hypercube differ in one bit (say bit j). It is easy to see that under the automorphism A_u their images also differ in bit j. Under A^α, their images differ in bit $\alpha(j)$:

Lemma 4.8 *For $w \in [2^d]$, $A^\alpha(w^j) = (A^\alpha(w))^{\alpha(j)}$.*

PROOF From the definitions, if $i \neq j$, then $b_{\alpha(i)}(A^\alpha(w^j)) = b_i(w^j) = b_i(w)$ and $b_{\alpha(i)}\left((A^\alpha(w))^{\alpha(j)}\right) = b_{\alpha(i)}(A^\alpha(w)) = b_i(w)$. Similarly, if $i = j$, then $b_{\alpha(i)}(A^\alpha(w^j)) = b_i(w^j) = \overline{b_i(w)}$ and $b_{\alpha(i)}\left((A^\alpha(w))^{\alpha(j)}\right) = \overline{b_{\alpha(i)}(A^\alpha(w))} = \overline{b_i(w)}$. Thus all bits of $A^\alpha(w^j)$ and $(A^\alpha(w))^{\alpha(j)}$ are equal. □

Given two permutations that are related under an automorphism, the following lemma shows the relationship between their projections.

Lemma 4.9 *Let $\phi \in S_{2^d}$, $\alpha \in S_d$, $u \in [2^d]$ and $j \in [d]$ be arbitrary. Then:*

$$\pi(A_u^\alpha \circ \phi \circ (A_u^\alpha)^{-1}, \alpha(j)) = A_u^\alpha(\pi(\phi, j))$$

PROOF For arbitrary $v \in [2^d]$,

$$\begin{aligned}
& v \in A_u^\alpha\left(\pi(\phi, j)\right) \\
\Leftrightarrow\ & (A_u^\alpha)^{-1}(v) \in \pi(\phi, j) \\
\Leftrightarrow\ & b_j\left(\phi \circ (A_u^\alpha)^{-1}(v)\right) \neq b_j\left((A_u^\alpha)^{-1}(v)\right) && \text{(by 4.7)} \\
\Leftrightarrow\ & b_{\alpha(j)}\left(A^\alpha \circ \phi \circ (A_u^\alpha)^{-1}(v)\right) \neq b_{\alpha(j)}\left(A^\alpha \circ (A_u^\alpha)^{-1}(v)\right) && \text{(by 4.1)} \\
\Leftrightarrow\ & b_{\alpha(j)}\left(A^\alpha \circ \phi \circ (A_u^\alpha)^{-1}(v)\right) \neq b_{\alpha(j)}\left(A^\alpha \circ A^{\alpha^{-1}} \circ A_u(v)\right) && \text{(by 4.4)} \\
\Leftrightarrow\ & b_{\alpha(j)}\left(A^\alpha \circ \phi \circ (A_u^\alpha)^{-1}(v)\right) \neq b_{\alpha(j)}(A_u(v)) \\
\Leftrightarrow\ & b_{\alpha(j)}(u) \oplus b_{\alpha(j)}\left(A^\alpha \circ \phi \circ (A_u^\alpha)^{-1}(v)\right) \neq b_{\alpha(j)}(u) \oplus b_{\alpha(j)}(A_u(v)) \\
\Leftrightarrow\ & b_{\alpha(j)}\left(u \oplus A^\alpha \circ \phi \circ (A_u^\alpha)^{-1}(v)\right) \neq b_{\alpha(j)}\left(u \oplus A_u(v)\right) \\
\Leftrightarrow\ & b_{\alpha(j)}\left(A_u \circ A^\alpha \circ \phi \circ (A_u^\alpha)^{-1}(v)\right) \neq b_{\alpha(j)}\left(A_u \circ A_u(v)\right) && \text{(by 4.2)} \\
\Leftrightarrow\ & b_{\alpha(j)}\left(A_u^\alpha \circ \phi \circ (A_u^\alpha)^{-1}(v)\right) \neq b_{\alpha(j)}(v) && \text{(by 4.3, 4.2)} \\
\Leftrightarrow\ & v \in \pi(A_u^\alpha \circ \phi \circ (A_u^\alpha)^{-1}, \alpha(j)) && \text{(by 4.7)}
\end{aligned}$$

□

Sometimes we characterize a permutation ϕ by specifying (for each dimension k), the set of nodes $P(k)$ containing packets that need to cross dimension k in order to be delivered. In the following we defi ne the $\sum$ operator to construct the permutation ϕ given these lists $P(k)$ for each dimension. This operation is in a loose sense the opposite of the projection operator π. We also show the precise sense in which they are opposite. The resulting operation is more general and can construct relations that are not functions.

Definition 4.10 *Suppose $P(k) \subseteq [2^d]$ for all $k \in [d]$. Then $\sum_k(P(k))$ describes a map from $[2^d]$ into $[2^d]$ in this way:*

$$b_j(\left(\sum_k (P(k))\right)(v)) = \overline{b_j(v)} \text{ if and only if } v \in P(j)$$

Note: sometimes the dummy variable k is omitted, so that $\sum_k(P(k))$ is written as $\sum P$.

Lemma 4.11 *Let $P(k) \subseteq [2^d]$ for all $k \in [d]$. $\pi(\sum_k P(k), j) = P(j)$.*

PROOF

$$\begin{aligned} v \in \pi(\textstyle\sum_k P(k), j) &\Leftrightarrow b_j((\textstyle\sum P)(v)) \neq b_j(v) && \text{(by 4.7)} \\ &\Leftrightarrow b_j((\textstyle\sum P)(v)) = \overline{b_j(v)} \\ &\Leftrightarrow v \in P(j) && \text{(by 4.10)} \end{aligned}$$

□

Lemma 4.12 *If ϕ is a map from $[2^d]$ into $[2^d]$, then $\sum_k(\pi(\phi, k)) = \phi$.*

PROOF Let v be arbitrary and $w = (\sum_k(\pi(\phi, k)))(v)$. Then:

$$\begin{aligned} b_j(w) = \overline{b_j(v)} &\Leftrightarrow v \in \pi(\phi, j) && \text{(by 4.10)} \\ &\Leftrightarrow b_j(\phi(v)) \neq b_j(v) && \text{(by 4.7)} \\ &\Leftrightarrow b_j(\phi(v)) = \overline{b_j(v)} \end{aligned}$$

Thus $w = \phi(v)$ for all v, so $\sum_k(\pi(\phi, k)) = \phi$. □

Whether a permutation is 1-separable by dimension k into $[d] \setminus \{k\}$ depends on $P(k)$. In the following we develop a characterization of those $P(k)$ for which the permutation is not 1-separable into $[d] \setminus \{k\}$.

Definition 4.13 *Let $\Pi(j) = \pi(S_{2^d}, j) \setminus \pi(\Delta_d, j)$.*

Lemma 4.14 *ϕ is 1-separable into $[d] \setminus \{j\}$ if and only if $\pi(\phi, j) = \pi(\delta, j)$ for some $\delta \in \Delta_d$.*

PROOF First, suppose that ϕ is 1-separable into $[d] \setminus \{j\}$, so that $\phi = \beta \circ \delta$ for some $\delta \in \Delta_d$ and $b_j(\beta(v)) = b_j(v)$ for all $v \in V(Q_d)$. Then $v \in \pi(\phi, j) \Leftrightarrow \langle v, v^j \rangle \in \chi(\beta \circ \delta) \Leftrightarrow b_j(\beta \circ \delta(v)) \neq b_j(v) \Leftrightarrow b_j(\delta(v)) \neq b_j(v) \Leftrightarrow \langle v, v^j \rangle \in \chi(\delta) \Leftrightarrow v \in \pi(\delta, j)$. Thus $\pi(\phi, j) = \pi(\delta, j)$.

Second, suppose $\pi(\phi, j) = \pi(\delta, j)$ for some $\delta \in \Delta_d$. This means that $b_j(\phi(v)) = b_j(\delta(v))$ for all $v \in V(Q_d)$, and replacing v by $\delta^{-1}(v)$ yields $b_j(\phi \circ \delta^{-1}(v)) = b_j(v)$ for all $v \in V(Q_d)$, so $\phi = (\phi \circ \delta^{-1}) \circ \delta$ is clearly a 1-separation. □

Corollary 4.14a *ϕ is 1-inseparable if and only if $\pi(\phi, j) \in \Pi(j)$ for all $j \in [d]$.*

The following lemmas essentially point out the relationship between projections $\pi(\phi, j) \in \Pi(j)$, of permutations ϕ that are not separable by dimension j into $[d] \setminus \{j\}$ and the images of the projection and the permutation under a hypercube automorphism. This will allow us to enumerate all the projections that are 1-inseparable by one dimension and modify it to obtain the set of projections that are 1-inseparable by the other dimensions.

Lemma 4.15 *Let $\alpha \in S_d$ and $p \in \Pi(j)$. Then $A^\alpha(p) \in \Pi(\alpha(j))$.*

PROOF $p \in \Pi(j)$ means there is a permutation ϕ, not allowable, such that $\pi(\phi, j) = p$. Then by Lemma 4.9, $\pi(A^\alpha \circ \phi \circ A^{\alpha^{-1}}, \alpha(j)) = A^\alpha(p)$. Now $A^\alpha \circ \phi \circ A^{\alpha^{-1}}$ cannot be allowable because otherwise ϕ would be allowable. Thus $A^\alpha(p) = \pi(A^\alpha \circ \phi \circ A^{\alpha^{-1}}, \alpha(j)) \in \Pi(\alpha(j))$. □

Corollary 4.15a $A^\alpha(\Pi(j)) = \Pi(\alpha(j))$ *for each $\alpha \in S_d$.*

PROOF The lemma implies $A^\alpha(\Pi(j)) \subseteq \Pi(\alpha(j))$. Applying the lemma for dimension $\alpha(j)$ and automorphism $A^{\alpha^{-1}}$ yields $A^{\alpha^{-1}}(\Pi(\alpha(j))) \subseteq \Pi\left(\alpha^{-1}(\alpha(j))\right)$ and thus $\Pi(\alpha(j)) \subseteq A^\alpha(\Pi(j))$. □

If all the arguments $P(k)$ of the $\sum$ operator are in the respective $\Pi(k)$, then we obtain a 1-inseparable permutation. In the following we modify the $\sum$ operator to accept all its arguments from one particular set $\Pi(J)$ where J is a fi xed dimension. This allows us to use only one copy of $\Pi(J)$ instead of constructing a different $\Pi(k)$ for each dimension k.

Definition 4.16 *Let a fixed $J \in [d]$ be given, and for all $k \in [d]$ let $\varrho_k \in S_d$ be given, satisfying $\varrho_k(J) = k$. Then if $P(k) \subseteq [2^d]$ for all $k \in [d]$, $\sum_k{}^{\varrho}(P(k))$ is defined as:*

$$\sum_k{}^{\varrho}(P(k)) = \sum_k \left(A^{\varrho_k}\left(P(k)\right)\right)$$

Theorem 4.17 *ϕ is 1-inseparable if and only if:*

$$\phi = \sum_k{}^{\varrho}(p_k) \ \mathit{where}\ p_k \in \Pi(J) \ \mathit{for\ all}\ k \in [d]$$

PROOF $\Rightarrow$: Let 1-inseparable ϕ be given. For any $k \in [d]$, Lemma 4.15 says $A^{{\varrho_k}^{-1}}(\pi(\phi, k)) = p_k$ for some $p_k \in \Pi(\varrho_k^{-1}(k)) = \Pi(J)$. Thus $\pi(\phi, k) = A^{\varrho_k}(p_k)$ for all $k \in [d]$. By Lemma 4.12,

$$\phi = \sum_k(\pi(\phi, k)) = \sum_k(A^{\varrho_k} \circ p_k) = \sum_k{}^{\varrho}(p_k)$$

$\Leftarrow$: Consider $\pi(\sum_k{}^{\varrho}(p_k), j)$ for arbitrary j:

$$\begin{aligned}
\pi(\textstyle\sum_k{}^{\varrho}(p_k), j) &= \pi(\textstyle\sum_k(A^{\varrho_k} \circ p_k), j) && \text{(by 4.16)}\\
&= A^{\varrho_j} \circ p_j && \text{(by 4.11)}\\
&\in \Pi(\varrho_j(J)) = \Pi(j) && \text{(by 4.15a)}
\end{aligned}$$

Thus ϕ is 1-inseparable by Corollary 4.14a. □

Our procedure for constructing $\Pi(J)$ for a fi xed dimension J is more easily understood based on a slightly different view of $P(k)$. This is developed in the following:

Definition 4.18 *The profile $\sigma_j(\phi)$ of permutation ϕ in dimension j is a map from the half-cube comprising nodes v with $b_j(v) = 0$ into $[4]$ defined this way:*

$$\sigma_j(\phi)(v) = 2\left(b_j(v) \oplus b_j(\phi(v))\right) + \Big(b_j(v^j) \oplus b_j(\phi(v^j))\Big)$$

The profi le is simply a way to encode the information from $\pi(\phi, j)$, as the following lemma shows:

Lemma 4.19

$$\sigma_j(\phi)(v) = \begin{cases} 0 & \textit{if } v \notin \pi(\phi, j) \textit{ and } v^j \notin \pi(\phi, j) \\ 1 & \textit{if } v \notin \pi(\phi, j) \textit{ and } v^j \in \pi(\phi, j) \\ 2 & \textit{if } v \in \pi(\phi, j) \textit{ and } v^j \notin \pi(\phi, j) \\ 3 & \textit{if } v \in \pi(\phi, j) \textit{ and } v^j \in \pi(\phi, j) \end{cases}$$

PROOF The result follows directly from the definitions. $\square$

Lemma 4.20 *The number of vertices v for which $\sigma_j(\phi)(v) = 1$ is the same as the number for which $\sigma_j(\phi)(v) = 2$.*

PROOF Let H_0 and H_1 be two disjoint half-cubes not containing any edges in dimension j. Since ϕ is a permutation, the number of v in H_0 for which $\phi(v) \in H_1$ must be the same as the number of v in H_1 for which $\phi(v) \in H_0$. This means that the number of v for which $\sigma_j(\phi)$ is 2 or 3 must be the same as the number for which $\sigma_j(\phi)$ is 1 or 3. Eliminating those for which $\sigma_j(\phi)$ is 3 gives the result. $\square$

The following property is satisfied by every profile σ which causes ϕ to be 1-inseparable, and it is easy to visualize and verify computationally when expressed this way:

Definition 4.21 *A profile $\sigma = \sigma_j(\phi)$ is said to be regular if there is a vertex-disjoint set of directed paths lying within the half-cube in which the profile is defined satisfying the following conditions:*

- *v appears at the head of a path if and only if $\sigma(v) = 1$.*
- *v appears at the end of a path if and only if $\sigma(v) = 2$.*
- *If v appears as an intermediate node in a path, then $\sigma(v) = 0$.*

Lemma 4.22 *$\sigma_j(\phi)$ is regular if and only if ϕ is 1-separable by $\{j\}$.*

PROOF In light of Lemma 4.14, we need only show that $\sigma_j(\phi)$ is regular if and only if $\pi(\phi, j) = \pi(\delta, j)$ for some $\delta \in \Delta_d$.

Suppose first that $\delta \in \Delta_d$ and $\pi(\phi, j) = \pi(\delta, j)$. By Lemma 4.19, this means that $\sigma_j(\phi) = \sigma_j(\delta)$. Consider the set of directed edges $\langle v, \delta(v)\rangle$ such that $b_j(v) = b_j(\delta(v)) = 0$. A straightforward application of the definitions show that $\sigma_j(\delta)$ is regular.

Second, suppose $\sigma_j(\phi)$ is regular. Define δ this way: if $\langle v, w\rangle$ is in some path, let $\delta(v) = w$ and $\delta(w^j) = v^j$; if v is at the head of a path, let $\delta(v^j) = v$; if v is at the end of a path, let $\delta(v) = v^j$. Clearly $\delta \in \Delta_d$. It is straightforward to verify, using Lemma 4.19, that $\pi(\delta, j) = \pi(\phi, j)$, and thus $\sigma_j(\delta) = \sigma_j(\pi)$.
$\square$

5 The case $d = 3$.

We characterize the 1-inseparable permutations in Q_3 using Theorem 4.17 with $J = 2$, $\varrho_0 = (201)$, $\varrho_1 = (21)$, and ϱ_2 the identity. Our goal is to enumerate the equivalence classes of 1-inseparable permutations, where two permutations are considered equivalent if one can be obtained from the other by a hypercube automorphism. Formally, $\phi_1 \approx \phi_2 \Leftrightarrow \phi_1 = A_u^\alpha \circ \phi_2 \circ (A_u^\alpha)^{-1}$ for some A_u^α.

The first step is to enumerate the members of $\Pi(2)$. By Lemma 4.22, we enumerate all $\sigma_2(\phi)$ that are not regular. The half-cube on which $\sigma_2(\phi)$ is defined comprises the four-cycle on $[4]$. By Lemma 4.20, there are zero, one, or two vertices for which $\sigma_2(\phi)$ is 1, and zero, one, or two, respectively, for which it is 2. In the first case, $\sigma_2(\phi)$ is regular vacuously. In the third case, edges connecting each vertex labeled 1 to a neighbor labeled 2 serve as the two disjoint paths establishing regularity.

In the second case, if the vertex labeled 2 is adjacent to the vertex labeled 1, then a directed edge connecting them establishes regularity. If the vertex labeled 2 is not adjacent to the vertex labeled 1, and if there is a vertex labeled 0, then a path of length two through the vertex labeled 0 suffices. Finally, if there is no vertex labeled 0, then no suitable path is possible, and in that case $\sigma_2(\phi)$ is not regular. There are four ways this final case can happen, according to which vertex $v \in [4]$ has $\sigma_2(\phi)(v) = 1$.

Lemma 4.19 can now be used to calculate the members of $\Pi(2)$, the inseparable projections in dimension 2. They are shown in the table below, together with their images under ϱ_1 and ϱ_2.

p	$A^{(201)}(p)$	$A^{(21)}(p)$
$p_0 = \{1, 2, 3, 4, 5, 6\}$	p_0	p_0
$p_1 = \{0, 2, 3, 4, 5, 7\}$	p_2	p_1
$p_2 = \{0, 1, 3, 4, 6, 7\}$	p_3	p_3
$p_3 = \{0, 1, 2, 5, 6, 7\}$	p_1	p_2

Now we face the task of examining all $\sum^\varrho(a, b, c)$ for $a, b, c \in \Pi(2)$, where we adopt the convention that a corresponds to dimension 2, b to dimension 1, and c to dimension 0. Instead of considering all 64 such cases, we use automorphisms of Q_3 to reduce those to a list of six, based on the following property of Q_3:

Lemma 5.1 *Suppose a set of six directed edges of Q_3 is formed by taking for each dimension j, a directed edge $\langle v, w\rangle$ that crosses dimension j together with its "opposite" which connects the antipode of v to the antipode of w. Then some automorphism of the hypercube will carry that set to one of the following:*

$\langle 2,0\rangle, \langle 0,1\rangle, \langle 1,5\rangle$ *and their opposites*

$\langle 2,0\rangle, \langle 0,1\rangle, \langle 5,1\rangle$ *and their opposites*

$\langle 2,0\rangle, \langle 0,1\rangle, \langle 0,4\rangle$ *and their opposites*

$\langle 2,0\rangle, \langle 0,1\rangle, \langle 4,0\rangle$ *and their opposites*

$\langle 2,0\rangle, \langle 1,0\rangle, \langle 4,0\rangle$ *and their opposites*

$\langle 0,2\rangle, \langle 0,1\rangle, \langle 0,4\rangle$ *and their opposites*

PROOF Suppose the set of edges contains some path of length two. Use A_m to move the midpoint m of that path to vertex 0, and then use a dimension relabeling A^α so that the path is $\langle 2,0\rangle, \langle 0,1\rangle$. There are four possible ways to choose a pair of edges for the other dimension, yielding the first four lines in the above list.

Suppose there is no path of length two. Since the six edges have twelve endpoints, some vertex m must be touched by two edges. Use A_m to move m to vertex 0, followed by A^α so that the other two edges are $\langle 2,0\rangle$ and $\langle 1,0\rangle$ or $\langle 0,2\rangle$ and $\langle 0,1\rangle$. In the first case, adding $\langle 5,1\rangle$, $\langle 1,5\rangle$, or $\langle 0,4\rangle$ would create a path of length two. In the second case, adding $\langle 5,1\rangle$, $\langle 1,5\rangle$, or $\langle 4,0\rangle$ would create a path of length two. The remaining choices produce the final two entries in the above list. $\square$

The above six possibilities are essentially different, which can be seen by observing the following distinguishing characteristics. The first comprises a directed cycle of length six. The second defines an undirected cycle of length six. The remaining four are characterized by having a degree-3 vertex with out-degree two, one, zero, and three respectively.

Lemma 5.2 *Given any map $\phi = \sum^\varrho(a,b,c)$ for $a,b,c \in \Pi(2)$, $\phi \approx \phi'$ where ϕ' is one of the following:*

ϕ'	*Projections*
$\sum^\varrho(p_1,p_3,p_0)$	(p_1,p_2,p_0)
$\sum^\varrho(p_2,p_3,p_0)$	(p_2,p_2,p_0)
$\sum^\varrho(p_0,p_3,p_0)$	(p_0,p_2,p_0)
$\sum^\varrho(p_3,p_3,p_0)$	(p_3,p_2,p_0)
$\sum^\varrho(p_3,p_3,p_3)$	(p_3,p_2,p_1)
$\sum^\varrho(p_0,p_0,p_0)$	(p_0,p_0,p_0)

PROOF Select a set of directed edges of Q_3 in the following way. For $j = 0,1,2$, include $\langle v, v^j\rangle$ in the set if and only if $v \notin \pi(\phi, j)$. Each p_i in $\Pi(2)$, and likewise each $\varrho_j(p_i)$, omits exactly a pair of vertices which are antipodes, so Lemma 5.1 applies to this set. It is straightforward to verify that the six sets listed in Lemma 5.1 correspond to the projections shown in the second column of the above table. Applying $A^{(21)^{-1}}$ in dimension 1 and $A^{(201)^{-1}}$ in dimension 0 yields the first column. □

Thus the six maps from Lemma 5.2, three of which are permutations, include representatives of all equivalence classes of 1-inseparable permutations. The following table shows the permutation determined, or in the case of no permutation, two vertices that are mapped to the same vertex under the map.

	Permutation or reason
$\sum^{\varrho}(p_1, p_3, p_0)$	$(065)(127)(34)$
$\sum^{\varrho}(p_2, p_3, p_0)$	$0 \rightarrow 6, 1 \rightarrow 6$
$\sum^{\varrho}(p_0, p_3, p_0)$	$(0275)(16)(34)$
$\sum^{\varrho}(p_3, p_3, p_0)$	$0 \rightarrow 6, 1 \rightarrow 6$
$\sum^{\varrho}(p_3, p_3, p_3)$	$1 \rightarrow 6, 2 \rightarrow 6$
$\sum^{\varrho}(p_0, p_0, p_0)$	$(16)(25)(34)$

Since the three listed permutations have cycles of different lengths, they are inequivalent. In light of Theorem 4.17, we have established:

Theorem 5.3 *There are three essentially different 1-inseparable permutations of Q_3:* $(065)(127)(34)$, $(0275)(16)(34)$, *and* $(16)(25)(34)$.

These permutations are illustrated in Figure 1.

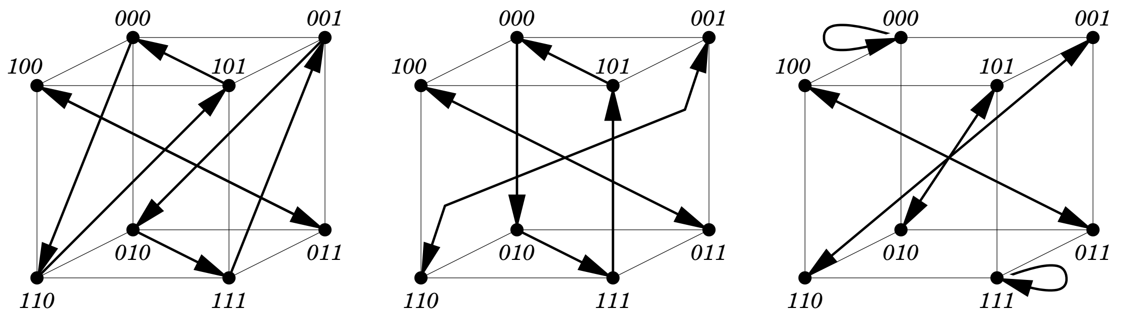

Figure 1: Three canonical 1-inseparable permutations

By factoring these three, we establish the main result for $d = 3$:

Theorem 5.4 *All permutations on eight elements can be factored into three allowable permutations of Q_3.*

PROOF Here are factorizations.

Permutation	Factorization
$(065)(127)(34)$	$(23)(45) \circ (0264)(1375) \circ (023754)$
$(0275)(16)(34)$	$(01)(23)(45)(67) \circ (023754) \circ (0264)(1375)$
$(16)(25)(34)$	$(132645) \circ (132645) \circ (132645)$

□

Corollary 5.4a *Every permutation of Q_3 which is neither 1-separable nor 2-separable is equivalent to* $(16)(25)(34)$.

PROOF Any permutation not equivalent to one of the above three is 1-separable. The factorizations shown for the first two of these provide 2-separations into dimension 0, since their third factors $(23)(45)$ and $(01)(23)(45)(67)$ use only dimension 0. Regarding the third permutation, without loss of generality assume dimension 0 is chosen for a 2-separation. Then two factors would have to achieve the permutation $(0176)(24)(35)$ which is easily seen to be impossible. Thus the third permutation is not 2-separable. □

6 The case $d = 4$.

In this section we present an experimental verification of the following results:

Result 6.1 *All permutations in S_{2^4} can be factored into four allowable permutations of Q_4.*

A slightly stronger result has also been verified:

Result 6.2 *All 1-inseparable permutations in S_{2^4} are 3-separable in all possible ways.*

This shows that the basic conjecture is true with "room to spare".

6.1 Basic Approach

The set of 1-inseparable permutations on Q_4 may be constructed similarly to the case for $d = 3$. For $d = 4$, we use $J = 3$, and a set $\{\varrho_j | j \in [4]\}$ defined by

$$\varrho_j(i) = \begin{cases} i & \text{for } i < j \\ j & \text{for } i = 3 \\ i+1 & \text{for } j \leq i < 3 \end{cases}$$

Again, we desire to consider the equivalence classes of 1-inseparable permutations and factorise one member of each equivalence class, as in Theorem 5.3. In order to do this, we experimentally determine the set $\Pi(3)$, based on the characterization in Lemma 4.22. To determine whether a possible profile λ is regular, we use a brute-force depth-first search algorithm. The algorithm checks whether there exists a set of paths that satisfy the conditions in Definition 4.21.

Algorithm 6.3 *EnumeratePi*

```
Output: $\Pi(3)$
For each labelling $\lambda : V(Q_3) \to [4]$
  If $|\{u|\lambda(u) = 1\}| = |\{u|\lambda(u) = 2\}|$
    If $\lambda$ is regular
      enumerate($\lambda$)
```

This enumeration yields 2120 labellings of $V(Q_3)$ which are not regular, and each of these determines a unique member of $\Pi(3)$. It is therefore necessary to consider 2120^4 4-tuples in an enumeration of 1-inseparable mappings, and not all of these are permutations.

We further reduce the space of permutations that need to be examined by restricting our attention to permutations that are canonical in the following sense (assuming an arbitrary total ordering on the 2120 labellings):

Definition 6.4 *A permutation $\phi = \Sigma^\varrho(m_3, m_2, m_1, m_0) \in S_{2^d}$ is canonical if the tuple (m_3, m_2, m_1, m_0) is lexicographically less than or equal to all tuples (m_3', m_2', m_1', m_0') such that $\Sigma^\varrho(m_3', m_2', m_1', m_0') = A_u^\alpha \circ \phi \circ (A_u^\alpha)^{-1}$ for some $u \in [2^d]$ and $\alpha \in S_d$.*

The basic idea of the algorithm is summarized in the following pseudocode:

Algorithm 6.5 *EnumerateOneInseparablePermutations*

```
Output: Enumeration of all essentially different 1-inseparable permutations
For each 4-tuple $(m_3, m_2, m_1, m_0) \in (\Pi(3))^4$
  If $\phi = \Sigma^\varrho(m_3, m_2, m_1, m_0)$ is a permutation
    If $\phi$ is canonical
      enumerate($\phi$)
```

Testing whether a mapping is a permutation is trivial. Testing whether ϕ is canonical is a little more interesting: Given a 4-tuple (m_3, m_2, m_1, m_0), for each A_u^α we need to find the equivalent tuple (m_3', m_2', m_1', m_0') defined by $\phi' = \Sigma^\varrho m_j' = A_u^\alpha \circ \phi \circ (A_u^\alpha)^{-1}$, where $\phi = \Sigma^\varrho m_j$.

The calculation of m_j' is based on the following equation:

$$
\begin{aligned}
m_j' &= A^{\varrho_j^{-1}}(\pi(\phi', j)) \\
&= A^{\varrho_j^{-1}}(A_u^\alpha(\pi(\phi, \alpha^{-1}(j)))) \\
&= A^{\varrho_j^{-1}}(A_u^\alpha(A^{\varrho_{\alpha^{-1}(j)}}(m_{\alpha^{-1}(j)}))) \\
&= A_{A^{\varrho_j^{-1}}(u)} \circ \left(A^{\varrho_j^{-1}} \circ A^\alpha \circ A^{\varrho_{\alpha^{-1}(j)}}\right)(m_{\alpha^{-1}(j)}) \\
&= A_{A^{\varrho_j^{-1}}(u)} \circ A^{\varrho_j^{-1} \circ \alpha \circ \varrho_{\alpha^{-1}(j)}}(m_{\alpha^{-1}(j)})
\end{aligned}
$$

This computation can be done very quickly using the following tables:

$$
\begin{aligned}
\mathbf{M}[m][\theta][u] &= A_u^{\theta^{-1}}(m) \in \Pi(3), \text{ for all } (m,\theta,u) \in \Pi(3) \times S_3 \times [2^4] \\
\mathbf{B}[j][\theta][u] &= \text{ blank, for all } (j,\theta,u) \in [4] \times \Pi(3) \times [2^4] \\
\mathbf{P}[\alpha][u][j] &= \&\mathbf{B}[\alpha^{-1}(j)][\varrho^{-1}_{\alpha^{-1}(j)} \circ \alpha^{-1} \circ \varrho_j][A^{\varrho_j^{-1}}(u)]
\end{aligned}
$$

Note that we would normally need to define θ over the range S_4, but we use S_3 because

$$\varrho^{-1}_{\alpha^{-1}(j)} \circ \alpha^{-1} \circ \varrho_j(3) = 3.$$

We are essentially focussing on 4-element permutations which map 3 to 3, and there is a very natural correspondence between these and the set S_3.

Once these tables are defined we can calculate m'_j very quickly, on average, by batch-processing a number of these computations which depend the same value of m_j. This is done by copying $\mathbf{M}[m_j][\theta][u]$ into $\mathbf{B}[j][\theta][u]$ for all $(j,\theta,u) \in [4] \times \Pi(3) \times [2^4]$. Then:

$$
\begin{aligned}
*\mathbf{P}[\alpha][u][j] &= \mathbf{B}[\alpha^{-1}(j)][\varrho^{-1}_{\alpha^{-1}(j)} \circ \alpha^{-1} \circ \varrho_j][A^{\varrho_j^{-1}}(u)] \\
&= \mathbf{M}[m_{\alpha^{-1}(j)}][\varrho^{-1}_{\alpha^{-1}(j)} \circ \alpha^{-1} \circ \varrho_j][A^{\varrho_j^{-1}}(u)] \\
&= A_{A^{\varrho_j^{-1}}(u)} \circ A^{\varrho_j^{-1} \circ \alpha \circ \varrho_{\alpha^{-1}(j)}}(m_{\alpha^{-1}(j)}) \\
&= m'_j
\end{aligned}
$$

This process of copying the 96 values in $\mathbf{M}[m_j][\theta][u]$ into $\mathbf{B}[j][\theta][u]$ for all $(j,\theta,u) \in [4] \times \Pi(3) \times [2^4]$ effectively defines $384 = 24 \times 16$ cells in the $*\mathbf{P}$ "virtual" array in one step. Moreover the fact that we compute the m'_j in lexicographical enumeration implies that m'_j can be used for a few levels of iteration if it is not m'_0. As explained in the following subsection, further optimizations are possible that leverage this approach.

6.2 Optimizations

In practice, we use predictive tests to rule out some cases when only some of the entries in a 4-tuple have been selected. If a partial tuple cannot be completed in a way that translates into a permutation, then none of the possible completions of the tuple need to be checked. Similarly, if a partial tuple can not be completed into a canonical permutation, then none of the possible completions need be generated. We use * to denote tuple entries that have not been fi lled in.

The noncanonical prediction works by using the array $\mathbf{P}$ to determine as many of (m'_3, m'_2, m'_1, m'_0) as possible for each $\alpha \in [S_d]$ and $u \in [d]$, and making the obvious inferences from those results.

Algorithm 6.6 *PotentialCanonical*

```
For $\alpha \in [S_d]$
  Let $j$ be the smallest value such that none of $m_{\alpha^{-1}(3)}, \ldots, m_{\alpha^{-1}(j)}$ is *
  For $u \in [2^d]$
    If $(m'_3, \cdots, m'_j) <_{lex} (m_3, \cdots, m_j)$, Return False
Return True
```

In practice the determination of j in the second line is hard-coded into the program so that only appropriate table references are performed.

The partial permutation test is given a list of k projections $m_3, \cdots m_{4-k} \in \Pi(3)$, and it is understood that the remaining elements $m_{4-k-1} \cdots m_0$ are unspecifi ed. Given a binary number v, let H_v be the set of nodes in $V(Q_4)$ whose leftmost k bits are the binary equivalent of v. For the purposes of computing $\sum^{\varrho}$, we can use any value for the unspecifi ed elements (including the null set).

Algorithm 6.7 *PotentialPermutation*

```
Output: True when there exists a way to specify the unspecified values to realise a permutation
Let $k$ be the number of $m_i$ whose values are specified.
Compute the mapping $\phi = \sum^{\varrho}(m_3, \ldots, m_{4-k}, \emptyset, \ldots, \emptyset)$
For each $v \in [2^k]$
  If $|\{u \in [2^4] | \phi(u) \in H_v\}| \neq 2^{4-k}$ Return False
Return True
```

These tests based on partial information are combined in the following top level algorithm which prunes the unnecessary branches of the search tree:

Algorithm 6.8 *EnumerateOneInseparablePermutations*

```
Output: Enumeration of all essentially different 1-inseparable permutations
For each $m_3 \in \Pi(3)$
 If PotentialCanonical$(m_3, *, *, *)$
  For each $m_2 \in \Pi(3)$
   If PotentialPermutation$(m_3, m_2, *, *)$
    If PotentialCanonical$(m_3, m_2, *, *)$
     For each $m_1 \in \Pi(3)$
      If PotentialPermutation$(m_3, m_2, m_1, *)$
       If PotentialCanonical$(m_3, m_2, m_1, *)$
        For each $m_0 \in \Pi(3)$
         If PotentialPermutation$(m_3, m_2, m_1, m_0)$
          If PotentialCanonical$(m_3, m_2, m_1, m_0)$
           enumerate($\phi = \Sigma^{\varrho}(m_3, m_2, m_1, m_0)$)
```

Note that no $PotentialPermutation$ test is performed at the outermost level because every projection in $\Pi(3)$ is known to be a potential permutation if it is the only specified mapping, by Algorithm 6.3. Also the tests at the innermost level determine whether in fact the specified 4-tuple of projections determines a permutation (and a canonical permutation).

Together, these partial tests enable us to identify 31,275,077 canonical permutations on Q_4 that need to be factored. This is 0.00015% of the total number of permutations. This savings includes a factor of 16 reduction for exploiting zero-relocation symmetries and a factor of 24 reduction for exploiting dimension-relabelling symmetries. Further reduction results from the decision to focus on explicitly constructed 1-inseparable permutations.

Cost savings from these partial tests at different levels are detailed in the following table:

	Number of tuple elements specifi ed			
Cutoff type	1	2	3	4
Nonpermutation	0	56,793	21,133,804	2,443,217,046
Noncanonical	2,078	21,259	987,583	12,634,637

Note that a cutoff after specifying k tuple elements results in a savings of $2120^{(4-k)}$ tuples. There are 2.02×10^{13} tuples that are not examined as a result of these cutoffs. Coincidentally, this is very close to the number (2.09×10^{13}) of permutations. Evidently, there are almost as many tuples to examine as there would be permutations to factor if we attempted a brute-force factorization of all permutations. However our approach of enumerating 1-inseparable permutations is much faster because it is much faster to test that a tuple is not a permutation (or not canonical) than it is to factor a permutation. No search and backtracking is required in the former.

In practice, we have found it possible to enumerate the canonical 1-inseparable permutations in an hour or two of run time. The rate at which we can factor random permutations (most of which are 1-separable) is 2.5×10^4 per second, versus 2.5×10^5 per second for for testing tuples.

6.3 Factoring

The basic structure of the factorization procedure is a depth-fi rst exhaustive search, controlled by a 5×16 array P. The task is best visualized as a one-person game with the following rules. Initially, "tokens" from the set $[16]$ are distributed to the vertices of Q_4, one token per vertex. Then a series of "moves" is allowed, where in each move, any subset of the tokens may be moved, each to some adjacent vertex, as long as no two tokens ever occupy a vertex simultaneously after the move is completed. At the end of four such moves, each token v must occupy vertex v. Thus, by placing token $\phi(v)$ at vertex v initially, the task of factoring ϕ is embodied in this game.

More generally, the same structure is used to fi nd a factorization showing that a permutation is separable into a given set of dimensions. The procedure simply searches for a sequence of moves that deliver each token to the appropriate subcube. For example, in testing for 3-separability into dimension 3, token 0 must be at either vertex 0 or vertex 8 after three steps, token 1 must be at 1 or 9, etc. We let $s_{\max}$ denote the number of steps in the factorization, so in testing separability into dimension-set D, $s_{\max} = d - |D|$.

The array P, at the end of a successful search process, records the complete history of one of these games: entry $P[s][v].\texttt{token}$ records the token present at vertex v immediately after step s, with $P[0][v].\texttt{token}$ recording the initial placement of tokens.

To begin the search, P is initialized with $P[0][v].\texttt{token} = \phi(v)$ and $P[s][v].\texttt{token} = \epsilon$ for $1 \leq s \leq s_{\max}$, where ϵ means "empty." The search process then fi lls in the empty entries one by one. Although it might seem natural to fi ll in all entries $P[1][v]$ fi rst, then $P[2][v]$, and so on, the search process instead chooses $P[s][v]$ one at a time using various criteria, so it may skip around through various values of s and v. The fact that the time steps are represented explicitly means the search process has the flexibility to do this.

During the search, entry $P[s][v]$ is said to be on the "frontier" if for every $s' \leq s$, there is some v' with $P[s'][v'].\texttt{token} = P[s][v].\texttt{token}$. If the progress of a particular token is conceptualized as an unbroken path forward through space and time, then the frontier is the point on that path that is furthest forward in time. In the search process, some frontier entry $w = P[s][v].\texttt{token}$ is chosen, and, through backtracking search, the consequences of setting $P[s+1][v'].\texttt{token} = w$ are evaluated for all feasible v'.

Sometimes a token is also placed at step $s_{\max}$ at the same time a frontier entry is fi lled in. This is done when all the paths available to reach the destination vertex or subcube must pass through a given vertex at step $s_{\max}$. For example, for a 3-separation into dimension 3, token 9 must be at vertex 1 or 9 after step 3; if it is initially at vertex 6, then it cannot reach vertex 9 in three moves, and thus $P[3][1].\texttt{token} \leftarrow 9$. Let us designate this particular operation a "prediction" by the algorithm. Predictions can trigger backtracking whenever multiple predicted or actual tokens would be required at any $P[s_{\max}][v]$.

Three tests are applied to determine whether a particular possible choice $P[s+1][v']$ is feasible. (Of course, only values v' which differ from v in at most one bit are even looked at.) (1) The token must still be within range of its destination; that is, its remaining distance to travel must not exceed the number of steps remaining. (2) If $s+1 < s_{\max}$, then there must be at least one way to move the given token forward from $P[s+1][v']$ toward its destination, using an entry $P[s+2][v'']$ not fi lled in with another token. (3) If a prediction is possible, then the token's predicted arrival at vertex w must not contradict $P[s_{\max}][w].\texttt{token}$. (In other words, two tokens cannot be predicted to occupy the same vertex after the last step.)

The running time of a backtracking search is strongly influenced by the order in which choices are attempted. In choosing an entry at the frontier, the algorithm chooses the entry which is the most "constrained," meaning that it has the fewest choices that will satisfy tests (1) and (2). The intent of this is to evaluate preferentially those choices that are most likely to lead to an impossible situation, thus accelerating the consequential backtracking. One side-effect is the immediate taking of forced actions, cases where there is only one feasible way to advance a token.

During the program's development, two other ways to speed up the search algorithm were tried but found to be ineffective. One was checking whether a potential placement would block another token's only feasible path, abandoning the placement if so. This method produced a small but inconsequential improvement in the run time. The other method was dynamic programming: recording in compressed form the non-empty part of the entire P array in a separate table, and avoiding evaluating the same situation twice. It turned out that the search algorithm rarely generated such repeated situations; due to the extra overhead, including access times for its large table in the presence of hardware cache and virtual memory effects, the dynamic programming version was only half as fast as the plain version.

Several items of information are stored for each $P[s][v]$ to support the above-described calculations:

$P[s][v]$.`goes_to`: move from v for step $s+1$, i.e. the v' such that $P[s][v]$.`token` $= P[s+1][v']$.`token`;

$P[s][v]$.`comes_from`: move to v for step s, i.e. the v' such that $P[s][v]$.`token` $= P[s-1][v']$.`token`;

$P[s][v]$.`predict`: predicted arrival location for $P[s][v]$.`token`, i.e. the v' such that $P[s][v]$.`token` $=$ $P[s_{\max}][v']$.`token`;

$P[s][v]$.`options`: number of feasible step-$(s+1)$ moves from v.

Algorithm 6.9 expresses the above ideas. It uses an auxiliary stack, rather than recursion, to support backtracking. The frontier is maintained as a doubly linked list, ordered by $P[s][v]$.`options`. Although the algorithm is formally organized just to determine whether a permutation is separable, as a side-effect it leaves information in the P array showing the factorization it discovered.

Algorithm 6.9 *Factor*

```
Output: whether permutation $\phi$ is separable into dimension(s) D
For each $v \in [2^d]$
  $P[0][v]$.token $\leftarrow \phi(v)$
  For each $s$, $1 \leq s \leq s_{\max}$, $P[s][v]$.token $\leftarrow \epsilon$
  Set all other fields of $P[s][v]$ to appropriate initial values
GoForward:
If every $(s, v)$ on Frontier satisfies $s = s_{\max}$, Return True
For $s < s_{\max}$, choose most constrained $(s, v)$ on Frontier /*C,F*/
$P[s][v]$.goes_to $\leftarrow \epsilon$
ContinueForward:
Choose next feasible move (after $P[s][v]$.goes_to) satisfying (1), (2), (3)
If there is no such move, GoTo GoBack
Record move in $P[s][v]$.goes_to and $P[s+1][v']$.comes_from
To predict arrival at $w$, set $P[s+1][v']$.predict and $P[s_{\max}][w]$.token /*P*/
Set $P[s+1][v']$.options to the number of feasible options satisfying (1), (2)
Push $(s+1, v')$ onto Stack
GoTo GoForward
GoBack:
If Stack is empty, Return False
Pop $(s+1, v')$ from Stack
Use $P[s+1][v']$.predict to retract any prediction
Set $(s, v)$ using $P[s+1][v']$.comes_from
GoTo ContinueForward
```

To show the effectiveness of the above-described techniques, versions of the factoring program were prepared which embodied subsets of the heuristics, and the programs'running times were measured. The tests were conducted on a Sun Microsystems 400 MHz UltraSPARC-II processor. The results are presented in Table 1.

Number of permutations tested per second				
Method	1-sep. (fails)	3-sep. (fails)	3-sep. (succeeds)	Fully factor
base	8,244.023	0.058	2.694	44.014
+F	8,223.684	0.001	0.144	0.149
+C	8,210.181	0.058	2.701	43.860
+P	7,942.812	1.262	64.185	42.955
+F+P	8,156.607	0.002	0.593	0.142
+C+F	13,495.277	1.645	37.760	37.841
+C+P	7,936.508	1.267	91.603	57.296
+C+F+P	13,531.800	2.150	48.788	36.532

Table 1: Run time consequences of the search-pruning heuristics.

When the program manages to find a factorization, it quits without attempting to enumerate all factorizations, and avoids much of the searching and backtracking that is required in a unsuccessful attempt. This may result in differing dynamics and run times. Thus, in preparing the test data, all permutations were first tested to determine their separability, and the program's performance was measured separately for batches of successful and unsuccessful factorization attempts.

Each table entry shows the number of permutations factored per second. (The number of permutations used in each test varied from 12 for the slowest versions of the program to 100,000 for the fastest.) The columns represent different types of problems: unsuccessful tests for 1-separability, unsuccessful tests for 3-separability, successful tests for 3-separability, and successful full factorizations without regard to separability. The table omits data for successful tests of 1-separability; those tests ran twice as fast as the fastest tests presented in the table and had no dependence on the heuristics. Most of the permutations were randomly selected; the full factorization test used just 1-inseparable permutations because they were not quite as easy as random ones, requiring about 50% more time.

The rows represent different versions of the program, corresponding to allowing the frontier to grow irregularly rather than by steps (denoted **+F**), and incorporation of the "prediction" (**+P**) and "constrained" (**+C**) heuristics. In versions lacking the "constrained" heuristic, the choices are made by the reverse of the heuristic: the choice with the largest number of remaining options is taken. The lines in the algorithm where these heuristics are implemented are marked with comments `/*C,F*/` and `/*P*/`.

It can be seen from this performance data that the run time of the program depends strongly on the heuristics. Except in the first column, the best versions of the program were about three orders of magnitude faster than the worst. No version of the program was uniformly best; those in the bottom rows of the table did generally better across all tests, and the program with all heuristics did best on the hardest problems.

This data predicts that 31,000,000 1-inseparable permutations which happen to be 3-separable in all four ways can have their 3-separabilities verifi ed in approximately $(4 \times 31,000,000) \div (48 \times 3600) \approx 720$ hours. The computation can be performed in parallel, so this computation can be carried out, for example, in one day using thirty computers.

The timing tests also rule out verifying Conjecture 3.8 by enumerating all permutations and attempting to factor. At a rate of 25,000 factorizations per second, approximately the rate at which 1-separability can be discovered, it would take $2 \times 10^{13} \div (9 \times 10^{7})$ hours to test all $16!$ permutations, or about 25 years of computer time.

6.4 Code Checking

Our algorithms are quite complicated, and therefore we ensure correctness by having two completely independent permutation enumeration programs, one by each author. First, we verify that the same total number of canonical permutations is obtained by the two programs. To further verify that we are actually producing the same set of permutations, they are enumerated in lexicographic order, and each of them is expressed in a mutually agreed binary format, which is then hashed using the MD5 checksum algorithm. Our algorithms agree on the checksum obtained for the entire set of permutations.

We have one version of the factoring algorithm, which is well tested. We have also run the program in a mode where every factorization is checked by composing the allowable permutations to verify that we obtain the factored permutation. The latter check is extremely simple to implement and debug, and it verifi es the factoring algorithm.

6.5 Problem Scaling

In this subsection we argue that this computational approach does not scale to the case $d = 5$. The following parameters of the problem increase very rapidly with the size of the hypercube dimension. Overall these rapidly increasing parameters discourage an exhaustive computational verifi cation of Conjecture 3.8 for $d = 5$.

Parameter Formula	Parameter Values		
Dimension			
d	3	4	5
Hypercube size			
2^d	8	16	32
Number of Permutations			
$\|S_{2^d}\|$	8!	16!	32!
Approximate Number of Permutations			
$\approx 2^d!$	4×10^4	2×10^{13}	3×10^{35}
Allowable Permutations (Note 1)			
$\|\Delta_d\|$	272	589,185	16,332,454,526,976 [34]
Number of Non-regular Profiles			
$\|\Pi(d-1)\|$	4	2120	167,983,476
Number of Canonical 1-inseparable permutations			
	3	31,275,077	$\approx 10^{30}$ (Note 2)

Note 1. This number is discussed in Subsection 6.6.

Note 2. This estimate is discussed in Section 7 on page 39.

6.6 On the number of allowable permutations

We determined the sequence of the numbers of allowable permutations (for increasing dimension starting from 0) to be:

$$1, 2, 9, 272, \cdots$$

We then searched the Sloane's Online Encyclopaedia of Integer Sequences [34] to determine higher numbers in the sequence. This sequence is characterized as the number of perfect matchings in a hypercube [5], and the next two (and only other known) terms in the sequence are 589,185 and 16,332,454,526,976 [34, 20, 26]. We also verified the value 589,185 computationally, and it turns out that the relationship between these two quantities is provable:

Theorem 6.10 *The number of perfect matchings in Q_{d+1} is the same as the number of allowable permutations on Q_d.*

PROOF In this proof, we use greek letters α, β, and γ to represent d-bit strings, and the roman letters a, b, c and d to represent bit variables. Let $\mathcal{M}_{d+1}$ be the set of perfect matchings on Q_{d+1}. Recall that Δ_d is the set of allowable permutations on Q_d. We will refer to the *parity* of a binary string as *even* or *odd* (rather than its numeric value). Given a matching M we will write $\vec{M}(a\alpha, b\beta)$ when $\{a\alpha, b\beta\} \in M$ and $a\alpha$ is even.

Defi ne the function $\Phi : \mathcal{M}_{d+1} \rightarrow \Delta_d$ by

$$\Phi(M) = \sigma, \text{ where } \sigma(\alpha) = \beta \text{ if } \vec{M}(a\alpha, b\beta)$$

Note that in the defi nition we always map the rightmost d bits of the even node in any edge of the matching to the rightmost d bits of the odd node. From this it is clear that the number of vertices on which σ is defi ned is the same as the number of edges in M. Hence to prove that σ is bijective, it suffi ces to prove that σ is onto.

For every perfect matching M, we claim that $\Phi(M) = \sigma \in \Delta_d$. Let β be given. Choose b such that $b\beta$ is odd. Suppose $\vec{M}(a\alpha, b\beta)$. Since $a\alpha$ is even, $\sigma(\alpha) = \beta$, so σ is onto, and therefore bijective.

Finally suppose that σ is not allowable. There exist α and β such that $|\alpha, \beta| \geq 2$ and $\sigma(\alpha) = \beta$. But this requires that $\vec{M}(a\alpha, b\beta)$ for some a and b, where $|a\alpha, b\beta| \leq 1$, which is not possible. Hence σ is allowable, thus proving that $\sigma \in \Delta_d$.

Next we claim that Φ is bijective. Given $\sigma \in \Delta_d$, we will construct a mapping $M \in \mathcal{M}_{d+1}$ such that $\Phi(M) = \sigma$ in the obvious way as follows: if $\sigma(\alpha) = \alpha$ then $\{0\alpha, 1\alpha\} \in M$. On the other hand consider the case that $\sigma(\alpha) = \beta \neq \alpha$ and $\sigma(\gamma) = \alpha$, where $\gamma \neq \alpha$. If α is even, then we ensure $\vec{M}(0\alpha, 0\beta)$. We also note that 1α is odd so γ is even and we can ensure $\vec{M}(1\gamma, 1\alpha)$. Similarly if α is odd, we can ensure $\vec{M}(1\alpha, 1\beta)$. Again we note that 0α is odd so γ is even and we can ensure $\vec{M}(0\gamma, 0\alpha)$. In order to show that M is a perfect matching we need to show for every α, 0α and 1α are both in the matching, and this is clearly the case from the construction, so it remains to show that M is a matching.

Suppose M as constructed is not a matching. Then there are three distinct nodes $a\alpha$, $b\beta$ and $c\gamma$ such that $\{a\alpha, b\beta\} \in M$ and $\{a\alpha, c\gamma\} \in M$. If $a\alpha$ is even, then $\sigma(\alpha) = \beta$ and $\sigma(\alpha) = \gamma$ imply that $\beta = \gamma$

since σ is one-to-one. Since $b\beta$ and $c\gamma$ are adjacent to $a\alpha$, they must be the same parity, so $b = c$, which is a contradiction. On the other hand if $a\alpha$ is odd, then $\sigma(\beta) = \alpha$ and $\sigma(\gamma) = \alpha$, which imply that $\beta = \gamma$ and $b\beta$ and $c\gamma$ are different parity, so both can not be adjacent to $a\alpha$. This proves that M is a matching and that Φ is onto.

It remains to show that Φ is one-to-one. Suppose $\Phi(M_1) = \Phi(M_2) = \sigma$. Also suppose that $\{a\alpha, b\beta\} \in M_1$ and $\{a\alpha, c\gamma\} \in M_2$, where $b\beta \neq c\gamma$. If $a\alpha$ is odd then $\sigma(\beta) = \alpha$ and $\sigma(\gamma) = \alpha$, so either σ is not one-to-one (a contradiction), or else $\gamma = \beta$. In the latter case $b \neq c$, to ensure distinctness, but that implies that $b\beta$ and $c\gamma$ are not the same parity, which is also a contradiction.

On the other hand if $a\alpha$ is even then $\sigma(\alpha) = \beta$ and $\sigma(\alpha) = \gamma$, which implies that either $\beta \neq \gamma$ and σ is not a function (a contradiction) or $\beta = \gamma$ and again parity constraints cannot be satisfi ed. Hence Φ is one-to-one and this completes the proof that $\Phi : \mathcal{M}_{d+1} \to \Delta_d$ is bijective. $\square$

This establishes our claim on the number of allowable permutations for $d = 5$, and shows a little more how the problem size increases with dimension. We also note that a closed form expression is not known for the number of perfect matchings in a hypercube [31, 26], hence for the number of allowable permutations in a hypercube.

7 Other Experimental Results

From Result 6.2, one might wonder whether there are any permutations that are not 3-separable in at least one dimension. (The answer seems to be that there are not.) To shed light on this, and to explore possible interconnections among the various kinds of separability, we tabulated statistics in several ways.

We tested three distinct sets of permutations: (a) randomly generated permutations; (b) the canonical 1-inseparable permutations; and (c) the inverses of the canonical 1-inseparable permutations. Random permutations were generated using the “ranper” function [23], using random numbers from the Solaris 5.7 operating system’s “random” function.

For each of these sets, we tested for 1-separability, 2-separability, and 3-separability, and tallied the occurrences of each combination. These results are presented in the following table. All table entries connote 3-separability because every permutation tested was 3-separable in some way. The results fail to disclose any compelling pattern. It appears that inseparability is rare in general.

Permutations (number tested)	3-sep only	1-sep & 3-sep	2-sep & 3-sep	1-sep 2-sep 3-sep
(a) random (2,100,000)	.0004%	.006%	.057%	99.94%
(b) 1-inseparable (31,275,077)	.79%		99.20%	
(c) inverses of 1-insep. (31,275,077)	.11%	.15%	5.21%	94.51%

Table 2: Separability frequencies for various sets of permutations.

In another experiment on 100 million randomly generated permutations, we addressed the question of whether 1-inseparability by a given dimension makes 1-inseparability or 3-inseparability of either the permutation or its inverse using either the given dimension or some other dimension more or less likely. We found little correlation. The strongest effect was that a permutation was twice as likely (30%) to be 1-inseparable by dimension d if its inverse was 1-inseparable by dimension d than it was in general (16.5%).

We also explored experimentally some possible underpinnings of Result 6.2. These explorations provided counterexamples to two conjectures, each of which would imply Result 6.2. The first conjecture was that if a permutation was 1-inseparable by dimension d, then it would be 3-separable into dimension d. We generated 446 counterexamples. A weaker conjecture was that the number of dimensions by which a permutation was 1-inseparable plus the number of dimensions into which it was 3-inseparable would never exceed 4. For this, we found exactly one counterexample: $(1\ 3\ 12\ 8\ 7\ 5\ 10\ 15\ 14)(2\ 13)(4\ 11)(6\ 9)$. It is 1-separable only by dimension 1, and it is 3-separable into just dimensions 1 and 2. Furthermore, it is not 2-separable at all, and its inverse is neither 1-separable nor 2-separable. It is the most difficult-to-factor permutation we know. We know of no permutations that are not 3-separable.

We estimated the frequency of 1-inseparability in 5 dimensions in two ways. First, we generated permutations at random and applied our factorization routine. We found that around 0.15% of 500 billion randomly generated permutations were 1-inseparable. This is double the rate in 4 dimensions. Second, we enumerated all 167,983,476 irregular profiles and stored them in a file. (Through data compression techniques, we were able to store them using an average of 3 bits per profile.) Then we generated 5-tuples of inseparable projections at random and tested whether they defined permutations. Using several hundred hours of computer time, we generated 3 billion such tuples and found 8 that defined permutations. This confirms the first estimate, since a rough calculation predicts 9: there should be $.0015 \cdot 32!$ inseparable permutations, there are 167,983,476^5 tuples, and

$$\frac{.0015 \cdot 32!}{167983476^5} \cdot 3 \times 10^9 \approx 8.8$$

Each canonical permutation corresponds on average to almost $5!$ distinct equivalent ones. (The correspondence would be exact if there were no permutations invariant under some hypercube automorphism; there are relatively few such instances.) This means there should be close to $.0015 \cdot 32! \div 5! \approx 3.3 \times 10^{30}$ canonical 1-inseparable permutations in 5 dimensions.

The theoretical results of Section 4 provide a reasonable level of understanding of 1-separability. As rare as 1-inseparability is, 2-inseparability is rarer and 3-inseparability rarer still. It appears likely that all permutations in four dimensions are 3-separable. The experimental results in five dimensions suggest a higher rate of 1-inseparability, but since we expect 4-inseparability to be rarer than 3-inseparability, the overall trend is unclear. We speculate that $(d-1)$-separability in d dimensions may be a productive subject for future research.

8 Conclusions

In this paper we have considered the offline permutation-routing problem on a hypercube, under the queueless MIMD packet-switched model, which uses one message buffer per node and assumes a communication capacity of one message per directed edge during each communication step. We have looked at the conjecture that d communication steps are sufficient to route an arbitrary permutation on a d-dimensional hypercube.

We have developed a theory of separability, and used it to prove the conjecture for the 3-dimensional hypercube. We have also used it to conduct a computer verification of the 4-dimensional case. The latter result improves the upper bound for the number of communication steps required to route an arbitrary permutation on arbitrarily large hypercubes to $2d - 4$.

In the process, we discovered a bijection between the possible communication steps in a d-dimensional hypercube and the number of perfect matchings in a $(d + 1)$-dimensional hypercube. It is interesting to note that there is no closed-form expression (or asymptotic characterization) for the latter combinatorial quantity. We have also presented experimental observations on the four- and five-dimensional cases. We propose 1-separability and $(d - 1)$-separability as ways to approach the general result.

References

[1] N. Alon, F. Chung, and R. Graham. Routing permutations on graphs via matchings. *SIAM Journal of Discrete Math.*, 7(3):513–530, August 1994.

[2] F. Annexstein and M. Baumslag. A unifi ed framework for off-line permutation routing in parallel networks. *Mathematical Systems Theory*, 24:233–251, 1991.

[3] G. Cooperman. Personal Communication to Mark Ramras.

[4] R. Cypher and C. Plaxton. Deterministic sorting in nearly logarithmic time on the hypercube and related computers. In *22nd. Annual Symposium on Theory of Computing*, pages 193–203, 1990.

[5] N. Graham and F. Harary. The number of perfect matchings in a hypercube. *Appl. Math. Lett.*, 1:45–48, 1988.

[6] M. Grammatikakis, D. Hsu, M. Kraetzl, and J. Sibeyn. Packet routing in fi xed-connection networks: A survey. *Journal of Parallel and Distributed Computing*, 54:77–132, 1998.

[7] P. Høyer and K. S. Larsen. Parametic permutation routing via matchings. *Nordic Journal of Computing*, 5(2):105, Summer 1998.

[8] P. Høyer and K. S. Larsen. Permutation routing via matchings. *Univ. Southern Denmark, IMADA (Department of Mathematics and Computer Science) preprint*, PP-1996-16. 13 pages.

[9] F. Hwang, Y. Yao, and B. Dasgupta. Some permutation routing algorithms for low dimensional hypercubes.

[10] F. Hwang, Y. Yao, and M. Grammatikakis. A d-move local permutation routing for the d-cube. *Discrete Applied Mathematics*, 72:199–207, 1997.

[11] S. L. Johnsson and C. T. Ho. Algorithms for matrix transposition for boolean n-cube confi gured ensemble architectures. *SIAM J. Matrix Appl.*, 9(3):419–454, 1988.

[12] S. L. Johnsson and C. T. Ho. Generalized shuffle permutations on boolean cubes. *J. Parallel Distrib. Comput.*, 16:1–14, 1992.

[13] S. L. Johnsson and C. T. Ho. Optimal communication channel utilization for matrix transpose and related permutations on boolean cubes. *Disc. Appl. Math.*, 53(1-3):251–274, 1994.

[14] C. Kaklamanis, D. Krizanc, and T. Tsantilas. Tight bounds for oblivious routing in the hypercube. In *Proc. 2nd ACM Symp. Parallel Algorithms and Architectures*, pages 31–44, 1990.

[15] M. Kaufmann and J. Sibeyn. Randomized multipacket routing an sorting on meshes. *Algorithmica*, 17:224–244, 1997.

[16] F. T. Leighton, F. Makedon, and I. G. Tollis. A $2n - 2$ step algorithm for routing in an $n \times n$ array with constant size queues. In *Proceedings of the 1989 ACM Symposium on Parallel Algorithms and Architectures*, pages 328–335, June 1989.

[17] T. Leighton. *Introduction to Parallel Algorithms and Architectures: Arrays, Trees, Hypercubes*. Morgan Kaufmann Publishers, San Mateo, CA, 1992.

[18] W. Lin. Manipulating general vectors on synchronous binary n-cube. *IEEE Trans. Comput.*, C-42(7):863–870, 1993.

[19] A. Lubiw. Counterexample to a conjecture of Szymanski on hypercube routing. *Information Processing Letters*, 35:57–61, June 1990.

[20] P. H. Lundow. Computation of matching polynomials and the number of 1-factors in polygraphs. *Research Reports of Department of Mathematics, Umeå University*, 12, 1996.

[21] D. Nassimi and S. Sahni. Optimal BPC permutations on a cube connected SIMD computer. *IEEE Trans. Comput.*, C-31(4):338–341, 1982.

[22] I. Newman and A. Schuster. Hot-potato algorithms for permutation routing. *IEEE Trans. Parallel and Dist. Systems*, 6(11), November 1995.

[23] A. Nijenhuis and H. S. Wilf. *Combinatorial Algorithms*. Academic Press, New York, 2nd edition, 1978.

[24] P. Ouyang and K. V. Palem. Very efficient cyclic shifts on hypercubes. *J. Parallel Distrib. Comput.*, 39(1):79–86, 1996.

[25] P. Panaite. Hypercube permutations routable under dimension orderings. *Inf. Proc. Letters.*, 59(4):185–189, 1996.

[26] J. Propp. *Enumeration of Matchings: Problems and Progress*, volume 38 of *Mathematical Sciences Research Institute Publications*, pages 255–291. Cambridge University Press, Amsterdam, 1999.

[27] M. Ramras. Congestion-free optimal routings of hypercube automorphisms.

[28] M. Ramras. Routing permutations on a graph. *Networks*, 23:391–398, 1993.

[29] M. Ramras. Congestion-free routings of linear complement permutations. *SIAM Journal of Discrete Math.*, 11(3):487–500, 1998.

[30] S. Ranka and S. Sahni. Odd even shifts in SIMD hypercubes. *IEEE Trans. Parallel Distrib. Systems*, 1(1):77–82, Jan 1990.

[31] H. Sachs. Problem 298, how many perfect matchings does the graph of the n-cube have? *Discrete Math.*, 191:251, 1998.

[32] X. Shen, Q. Hu, and W. Liang. Realization of an arbitrary permutation on a hypercube. *Information Processing Letters*, 51(5):237–243, September 1994.

[33] J. F. Sibeyn, B. S. Chlebus, and M. Kaufmann. Deterministic permutation routing on meshes. *Journal of Algorithms*, 22(1):111–141, Jan 1997.

[34] N. J. Sloane. *Sloane's On-Line Encyclopaedia of Integer Sequences*. WWW, http://www.research.att.com/˜njas/sequences/index.html, 2000.

[35] T. Szymanski. On the permutation capability of a circuit-switched hypercube. In *Proceedings of the 1989 International Conference on Parallel Processing*, pages I–103–I–110, 1989.

[36] C. D. Thompson. Generalized connection networks for parallel processor intercommunication. *IEEE Transactions on Computers*, C-27(12):1119–1125, December 1978.

[37] B. Vöcking. Almost optimal permutation routing on hypercubes. In *Proceedings of the thirty-third annual ACM symposium on Theory of computing*, pages 530–539. ACM Press, 2001.

[38] L. Zhang. Optimal bounds for matching routing on trees. In *Proceedings of the Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 445–453, New Orleans, Louisiana, January 1997.