ARTICLE IN PRESS

Journal of Symbolic Computation ▮ (▮▮▮▮) ▮▮▮–▮▮▮

Contents lists available at ScienceDirect

Journal of Symbolic Computation

journal homepage: www.elsevier.com/locate/jsc

Harnessing parallel disks to solve Rubik's cube☆

Daniel Kunkle, Gene Cooperman

College of Computer Science, Northeastern University, Boston, MA 02115, USA

ARTICLE INFO

Article history:
Received 29 November 2007
Accepted 22 April 2008
Available online xxxx

Keywords:
Rubik's cube
Permutation groups
Cosets
Fast multiplication
Disk-based computation
Parallel computation

ABSTRACT

The number of moves required to solve any configuration of Rubik's cube has held a fascination for over 25 years. A new upper bound of 26 is produced. More important, a new methodology is described for finding upper bounds. The novelty is two-fold. First, parallel disks are employed. This allows 1.4×10^{12} states representing *symmetrized cosets* to be enumerated in seven terabytes. Second, a faster table-based multiplication is described for symmetrized cosets that attempts to keep most tables in the CPU cache. This enables the product of a symmetrized coset by a generator at a rate of 10 million moves per second.

1. Introduction

Twenty-five years ago, Frey and Singmaster (1982) conjectured at the end their book, *Cubik Math*, that "God's number" is in the low 20's:

> No one knows how many moves would be needed for "God's Algorithm" assuming he always used the fewest moves required to restore the cube. It has been proven that some patterns must exist that require at least seventeen moves to restore but no one knows what those patterns may be. Experienced group theorists have conjectured that the smallest number of moves which would be sufficient to restore any scrambled pattern – that is, the number of moves required for "God's Algorithm" – is probably in the low twenties.

This conjecture remains unproven today. At the time of this conjecture, the best known bounds were a lower bound of 17 and an upper bound of 52 (Frey and Singmaster, 1982). The current best lower bound is 20 (Reid, 1995b). In Kunkle and Cooperman (2007), the authors demonstrated a new upper bound of 26.

☆ This work was partially supported by the National Science Foundation under Grant CNS-06-19616.
E-mail addresses: kunkle@ccs.neu.edu (D. Kunkle), gene@ccs.neu.edu (G. Cooperman).

0747-7171/$ – see front matter © 2008 Published by Elsevier Ltd
doi:10.1016/j.jsc.2008.04.013

Please cite this article in press as: Kunkle, D., Cooperman, G., Harnessing parallel disks to solve Rubik's cube. Journal of Symbolic Computation (2008), doi:10.1016/j.jsc.2008.04.013

Note that in all cases, we consider a *move* to be any quarter or half turn of a face of the cube, also known as the *face-turn metric*. We do not consider the alternative *quarter-turn metric*, which defines a half-turn to be two moves.

We present a new, algebraic approach based on the analysis of cosets for mathematical groups. Rubik's cube can be viewed as a mathematical group with each group element a permutation of the facelets of Rubik's cube. A *facelet* is one of the 54 tiles, with 9 facelets per face. The 18 natural moves are taken as the generators of the group. Since the generators always fix in place the center facelet of a face, Rubik's cube can be viewed as a permutation group acting on the 48 movable facelets. Multiplication of moves or generators is simply composition of moves. Hence, multiplication is identified with permutation multiplication for permutation groups.

Like previous efforts, we decompose Rubik's group G using a subgroup H. As is well-known in group theory, the elements of Rubik's group G are partitioned into *cosets* $Hg = \{hg : h \in H\}$, where $g \in G$. Hence, there are $|G|/|H|$ cosets. The set of cosets of H in G is denoted as G/H.

This allows a divide-and-conquer strategy. The number of configurations of Rubik's cube is $|G| \approx 4.3 \times 10^{19}$. Given an unsolved state or permutation $g \in G$, one can identify the coset Hg containing g. One searches for a shortest sequence of generators τ such that the product $g\tau \in H$. One then searches for a shortest sequence of generators of H, σ, such that $g\tau\sigma$ is the identity permutation (the home position). After finding upper bounds on the length of τ and on the length of σ, the sum of the two is then an upper bound for the number of moves to solve Rubik's cube. Traditionally, one chooses a particular subgroup R, such that $|R| \approx 2 \times 10^{10}$ and $|G|/|R| \approx 2.2 \times 10^{9}$. This makes the search feasible in RAM for both problems (after the number of states is reduced by symmetries).

Unlike previous efforts, we choose the *square subgroup* Q, the group generated by squares of generators (or 180° turns), as our subgroup. Since $|Q| \approx 6.6 \times 10^{5}$, $|G|/|Q| \approx 6.5 \times 10^{13}$ (or 1.4×10^{12} after reduction by symmetries).

1.1. Paper organization

The paper is organized as follows. In the rest of the introduction, we provide an overview of the three primary contributions of this work: a new upper bound on solutions to Rubik's Cube; the use of parallel disk-based computation; and methods for producing small, fast tables for group multiplication. Then, Section 2 briefly reviews some related work in these areas. Section 3 presents background and some basic concepts. In particular, this includes the definitions of symmetrized group element and symmetrized coset. Section 4 describes the fast multiplication algorithm, along with the perfect hash function. Section 5 shows that all elements of the square subgroup are solvable in 13 moves. Section 6 shows that all cosets are within 16 moves of the trivial coset. Finally, Section 7 presents methods for further reducing the upper bound on solutions, providing the final upper bound of 26 moves.

1.2. Upper bounds

We choose this very small subgroup for three reasons: in the limit, with a subgroup equal to the identity of the entire group, this method would produce exactly optimal results; the square subgroup is the only non-trivial subgroup that preserves all 48 symmetries of the cube; and, the use of a small subgroup allows us to more efficiently prove nearly optimal upper bounds on any coset (see Section 7).

The search for the worst case in G/Q produces 17 symmetrized cosets (equivalence classes of cosets under symmetries of the cube) that each require 16 moves to return to the subgroup Q. Any element in the subgroup Q can be "solved", using only generators of Q, in 15 moves. However, by allowing one to use any generators of Rubik's group G, an easy computation shows that 13 moves always suffice to solve for any element of Q. This immediately produces a bound of $16 + 13 = 29$ moves.

We further reduce the upper bound from 29 to 26 in two steps. First, a refinement strategy is used to analyze all group elements that are members of a symmetrized coset at level 9. The refinement computation shows that the optimal solution for any such group element is bounded above by only 20 moves, instead of the expected $13 + 9 = 22$ moves, giving elements at the furthest level a bound of $(16 - 9) + 20 = 27$. Finally, that bound of 27 is reduced to 26 by directly analyzing the 17 worst case symmetrized cosets.

1.3. Parallel disks

We briefly review the use of the parallel disks in proving that all cosets are within 16 moves of the square subgroup. The computation was parallelized using a library developed by the second author called TOP-C (Task Oriented Parallel C/C++) (Cooperman, 1996). Initially, the computation was carried out in 63 cluster hours using a high-end SAN (storage area network). Later, we duplicated the computation in 183 cluster hours using only the local disks of a commodity cluster.

To find bounds on the length of solutions among the symmetrized cosets, we use a parallel disk-based version of breadth-first search. A standard algorithm maintains a current *frontier* corresponding to all states at a given level ℓ. For each state of the frontier, one makes all moves (applies all generators) to produce a list of neighbors. Those neighbors that have already been seen at level ℓ or $\ell - 1$ are eliminated. The remaining neighbors form the next frontier at level $\ell + 1$.

The primary difficulty with creating a disk-based version of this algorithm is in the duplicate detection phase, where we must compare newly generated states to those previously seen. This typically makes use of random access data structures, such as hash tables, which are not efficient on disk. To solve this problem we delay duplicate detection and use a method similar to that of bucket sort, essentially breaking the task into RAM-sized pieces that can make use of random access efficiently.

A second issue arises when the search frontier exceeds the capacity of disk, as it does in our case. We solve this problem using an *implicit open list*. This method works by encoding the search frontier in the hash table and reconstructing it with an inverse hash function, instead of explicitly saving the non-duplicate states.

Further background on disk-based computation can be found in Section 2, and details of our method can be found in Section 6.

1.4. Small fast multiplication tables

One additional technique we employ is a fast multiplication routine that accepts the hash index of a symmetrized coset and a generator, and directly produces the hash index of the product. This allows us to avoid the costly step of working with the original representation of a symmetrized coset. While the fast multiplication has some complexity, it is conceptually simple, and based on the group theoretic decomposition of a group G into a subgroup H such that H further decomposes into a product QN, for N a *normal subgroup* of H. The definition of normal subgroup is given in Section 3.1 and further details of the fast multiplication are provided in Section 4.

1.5. Extensions to previous work

This paper is an extension of the work presented in Kunkle and Cooperman (2007), which originally proved 26 moves sufficient for Rubik's cube. Along with more general enhancements, we provide three specific contributions over that previous work.

First, we provide a more thorough analysis of the square subgroup, including the distribution of depths of the elements and a visual representation of the group structure.

Second, we include additional experimental results for the disk-based computation in Section 6. These new results compare the results using two different cluster computing architectures, using either global shared disk or locally attached disks. We find that the relatively cheap local disk architecture can be used to efficiently perform computations that previously were performed on a much larger supercomputer.

Finally, we present an analysis of three techniques for refining the upper bound on the group radius, including: optimal solvers; image intersection; and projection. We also provide experimental results detailing additional computations that correct an omission in the previous result, which did not provide shortened solutions for all of the necessary group elements.

2. Related work

One approach to finding bounds on solutions to Rubik's cube would be to produce the entire Cayley graph for the corresponding group. This approach was used to show that 11 moves suffice for Rubik's

$2 \times 2 \times 2$ cube (Isaacs, 1981). This smaller cube problem has also proven useful for the testing of newer methods, such as a graph representation using only two bits per element, described in Cooperman et al. (1990). For the full $3 \times 3 \times 3$ Rubik's cube, these methods are not feasible, since it has over 4.3×10^{19} states.

The first lower bound, of 17, was shown using a simple counting argument, i.e. the number of possible states achievable in 16 moves is less than the total number of possible states, so there must be some states requiring at least 17 moves (Frey and Singmaster, 1982).

It was then conjectured that a specific cube position, *super-flip*, would be a position requiring a near-maximum number of moves to solve. The super-flip position is of interest because it is the only element, other than the identity, in the center of Rubik's group. A solution of length 20 was found by Winter (1992), which was later proven optimal by Reid (1995b).

The first published upper bound was 52. Discovered by Thistlethwaite (Frey and Singmaster, 1982), it was based on solving the cube in a series of four steps, corresponding to a chain of subgroups of length four. The four steps were proven to have worst case lengths of 7, 13, 15, and 17, for the total of 52.

This algorithm was improved by Kociemba (2007) to use a subgroup chain of length two. Reid (1995a) proved the worst case for the two steps was 12 and 18, for a total upper bound of 30. Further analysis showed that the worst case never occurs, and so a bound of 29 was shown. This bound was further refined by Radu (2006) to 27, which was the best upper bound before our working showing 26 moves suffice (Kunkle and Cooperman, 2007). Recently, between the acceptance and publication of this paper, Rokicki (2008b) reduced the upper bound to 23 using 7.8 core-years of CPU time, an extension of the method described in Rokicki (2008a).

Besides work into methods with provable worst cases, several optimal solvers with no worst case analysis have been developed. The method developed by Kociemba and analyzed by Reid has a natural extension that guarantees optimal solutions. Korf (1997) used similar techniques to optimally solve ten random cube states, one in 16 moves, three in 17 moves, and the remaining six in 18 moves.

One key to our result is the use of disk-based methods for performing search and enumeration, and specifically the disk-based breadth-first search described in Section 6. Several disk-based search methods have been introduced in recent years. The common difficulty each of these methods must overcome is the significant latency of disk, which disallows the random access patterns typically used when detecting duplicate states.

Korf (2004) used sorting-based delayed duplicate detection to to solve sliding tile puzzle and Towers of Hanoi type problems. Korf and Schultze (2005) also introduced hash-based delayed duplicate detection, which can be used to avoid external sorting in some applications (we use a hash-based method for our computation). Zhou and Hansen (2004) introduced structured duplicate detection, which can utilize disk without delaying the detection of duplicates. Robinson and Cooperman (2006) introduced tiered duplicate detection as a method to speedup the enumeration of the Baby Monster sporadic simple group, and more recently applied it to the problem of the Fischer_{23} group (Robinson et al., 2007b). A comparative analysis of each of these methods, among other search techniques, can be found in Robinson et al. (2007a).

3. Notation and basic concepts

3.1. Group theory definitions

We review the formal mathematical definitions. Recall that a group G is a set with multiplication and an identity e ($eg = ge = g$), inverse ($gg^{-1} = g^{-1}g = e$), and an associative law ($(gh)k = g(hk)$). A *permutation* of a set Ω is a one-to-one and onto mapping from Ω to Ω. Composition of mappings provides the group multiplication, and the group inverse is the inverse mapping. A *permutation group* G is a subset of the permutations of a set Ω with the above operations. A *subgroup* $H < G$ is a subset H that is closed under group operations. A group G has *generators* $S \subseteq G$, written $G = \langle S \rangle$, if any element of G can be written as a product of elements of S and their inverses. The *order* of the group is the number of elements in it, $|G|$.

Please cite this article in press as: Kunkle, D., Cooperman, G., Harnessing parallel disks to solve Rubik's cube. Journal of Symbolic Computation (2008), doi:10.1016/j.jsc.2008.04.013

Given a group G and a subgroup $H < G$, a coset of H is the set $Hg = \{hg : h \in H\}$. A subgroup $H < G$ partitions the group into cosets. The set of all cosets is written G/H. The *conjugate* of h by g is defined by $h^g \stackrel{\text{def}}{=} g^{-1}hg$. $N < G$ is *normal* in G if $\forall n \in N, g \in G, n^g \in N$.

An *automorphism* α of a group G is a one-to-one and onto mapping of G such that for $g_1, g_2 \in G$, one has $\alpha(g_1g_2) = \alpha(g_1)\alpha(g_2)$. The informal idea of symmetries of Rubik's cube has its formal analogue in automorphisms.

A *Cayley graph* of a group G with generators S is a directed graph whose vertices are the elements of G and whose directed edges, (g_1, g_2), satisfy $g_1s = g_2$ for some edge label $s \in S$. Since our chosen generating set for Rubik's cube is preserved under inverses, the Cayley graph of Rubik's cube can also be considered an undirected graph. A *Schreier coset graph* of a group G with generators S and subgroup $H < G$ is a graph whose vertices are the elements of G/H and whose edges, (Hg_1, Hg_2), satisfy $Hg_1s = Hg_2$ for some edge label $s \in S$.

3.1.0.1. Symmetrized group elements and symmetrized cosets. While the above definitions are standard, we extend them to symmetrized group elements and symmetrized cosets. Let A be a group of automorphisms of a group G. A *symmetrized group element* g^A for $g \in G$ is the set $\{\alpha(g) : \alpha \in A\}$.

Given a subgroup $H < G$, let A be a group of automorphisms of G that also preserve H ($\forall h \in H, \alpha \in A, \alpha(h) \in H$). The *symmetrized coset* Hg^A is the set of elements $\{h\alpha(g) : h \in H, \alpha \in A\} = \cup_{\alpha \in A} H\alpha(g)$. (Note $\alpha(Hg^A) = Hg^A \; \forall \alpha \in A$.)

Let A be a group of automorphisms of G. Assume that A preserves the generating set S of G. Then the *symmetrized Cayley graph* for (G, S, A) is the directed graph with vertices $\{g^A : g \in G\}$ and with edges (g_1^A, g_2^A) where $\forall g_1' \in g_1^A, \exists s \in S$ such that $g_1's \in g_2^A$.

Without loss of generality, it suffices to consider only edge labels satisfying $g_1s \in g_2^A$ for a distinguished group element $g_1 \in g_1^A$. Note that the edge label in a symmetrized Cayley graph is not unique since we could equally well consider $\alpha(g_1')\alpha(s) \in \alpha(g_2)$ as defining that edge. Hence, the edge (g_1^A, g_2^A) has edge label s such that $g_1s = g_2$ and also edge label $s' = \alpha(s)$ for some $\alpha \in A$, such that $\alpha(g_1)\alpha(s) = \alpha(g_2)$. For any element $g_1's \in g_2^A$ satisfying $g_1' \in g_1^A$, there is an $\alpha \in A$ with $\alpha(g_1') = g_1$ and so the edge $((g_1')^A, (g_1's)^A) = (g_1^A, (g_1\alpha(s))^A)$.

Similarly, assuming that $G = \langle S \rangle$, $H < G$, and A preserves S, one defines the *symmetrized Schreier coset graph* for (G, H, S, A) as the directed graph with vertices $\{Hg^A : g \in G\}$ and with edges (g_1^A, g_2^A) satisfying $Hg_1' \in Hg_1^A$ and $Hg_1s \in Hg_2^A$. As before, in order to find all neighbors of g_1^A in a symmetrized Cayley graph, it suffices to choose any distinguished element $g' \in Hg_1^A$, and the set of neighbors is $\{(H(g')^A, H(g's)^A) : s \in S, g's \notin H(g')^A\}$.

Note that for identity $e \in G$, the trivial coset $He = H$ and the trivial symmetrized coset $He^A = H$ are equal. For our work, we require the following property: Let $g' \in Hg^A$. Then the distance from the symmetrized coset Hg^A to the trivial symmetrized coset H in the symmetrized Schreier coset graph is the same as the distance from the coset Hg' to the trivial coset H in the Schreier coset graph. The property is easy to prove. If there is a word w' with distinguished element $g'' \in Hg^A$ such that $g''w' \in He^A = H$, then there is an $\alpha \in A$ with $\alpha(g'') = g'$, and therefore $g'\alpha(w') = \alpha(g'')\alpha(w') \in H$. Since the automorphisms α preserve S, $\alpha(w')$ is a word in S of length d, from Hg' to the trivial coset in the Schreier coset graph.

3.1.0.2. Perfect hash function. Next, a *perfect hash function* is a hash function that produces no collisions. Hence, it is one-to-one. Section 4 describes efficient perfect hash functions both for certain classes of symmetrized cosets and for symmetrized group elements. This function is nearly minimal, and has an efficiently computable *inverse hash function*.

3.2. Rubik's cube definitions

We assume the reader has seen a Rubik's cube, and we provide this description solely to fix terminology according to standard conventions (Frey and Singmaster, 1982).

A Rubik's cube is built from 26 *cubies*, each able to make restricted rotations about a core of Rubik's cube. A *face* of Rubik's cube is a side. Each face is divided into 9 *facelets*, where each of the 9 facelets is part of a distinct cubie. A cubie is either an *edge cubie* (two visible facelets), a *corner cubie* (three visible facelets), or a *center cubie* (one visible face, in the center of a side). The facelets are similarly *edge facelets*, *corner facelets*, or *center facelets*.

The states of Rubik's cube can be considered as permutations on 48 facelets (the 24 corner facelets and the 24 edge facelets). The center facelets are considered to be fixed, and all rotations of Rubik's cube are considered to preserve a fixed orientation of the cube in 3 dimensions. The *home position* or solved position of Rubik's cube is one in which all facelets of a face are the same color, and (for the sake of specificity) the blue face is downwards. We will speak interchangeably about an *element of Rubik's group*, a *state*, or a *position* of Rubik's cube. Similarly, we will speak interchangeably about the *home position of Rubik's cube* or the *identity element of Rubik's group*. Hence, a position of the cube is identified with a group element that permutes the facelets from the home position to the given position.

The *moves* of Rubik's group are conventionally denoted U, U^{-1}, U^2, D, D^{-1}, D^2, R, R^{-1}, R^2, L, L^{-1}, L^2, F, F^{-1}, F^2, B, B^{-1}, and B^2. Mnemonically, U stands for a clockwise quarter turn of the "up" face, and similarly, D, R, L, F and B stand for "down", "right", "left", "front" and "back", respectively. Any of the above moves will move exactly 20 facelets. These moves also make up the generators of the Rubik's cube group, G. Standard techniques, such as Sims's algorithm for group membership, show that the order of Rubik's group is $|G| = 43,252,003,274,489,856,000$ (approximately 4.3×10^{19}).

The generators of the *square subgroup* are given by $Q = \langle U^2, D^2, R^2, L^2, F^2, B^2 \rangle$. Standard techniques show that the order of the square subgroup is $|Q| = 663,552$ (approximately 6.6×10^5). The *index*, or number of cosets, is $[G : Q] = |G|/|Q| = 65,182,537,728,000$ (approximately 6.5×10^{13}).

3.3. Symmetries (natural automorphisms) of Rubik's cube

For Rubik's cube, we desire a subgroup of 48 automorphisms that preserve the set of generators: the *natural automorphisms* of Rubik's cube. Each automorphism can be identified with a *symmetry of a geometric cube*: one of 24 rotations of the entire cube; or a rotation followed by an inversion of the cube (mapping each corner of the cube to the opposite corner). A rotation of the cube maps the natural generators of Rubik's cube to generators. An inversion of the cube maps generators to inverse generators (clockwise quarter turns to counter-clockwise quarter turns). These 48 symmetries of the cube are known to preserve the natural generators of Rubik's group, and no other automorphisms of Rubik's group do so.

4. Fast group multiplication in the presence of symmetries

Next, the method of fast multiplication is presented. The method breaks up a group into smaller subgroups (called *coordinates* by Kociemba (2007)). The chosen subgroups are tailored for fast multiplication both within the square subgroup (group generated by squares of generators) and multiplication of the cosets by generators. Each subgroup is small enough so that the derived table-based computations fit inside CPU cache.

We will separately consider the *edge group* (the action on edge facelets) and the *corner group* (the action on corner facelets). The two actions are "linked". This will be discussed in later subsections. First, we provide the fundamental basis for our result.

4.1. Decomposition into smaller subgroups and fast multiplication

The following easy result provides the basis for our fast multiplication.

Lemma 1. *Let G be a group and N a normal subgroup of G with $G > H = QN$ and $Q \cap N = \{1\}$. Then given a set of canonical coset representatives of G/H, an element $g \in G$ uniquely defines q, $\bar{g}$ and n as follows.*

$$g = q\bar{g}n, \quad \text{where } q \in Q, \bar{g} \text{ is the canonical coset representative of } Hg, \text{ and } n \in N.$$

Proof. Clearly, $g = h\bar{g}$ for a uniquely defined $h \in H$ and $\bar{g}$ the coset representative of Hg. The equation $h = qn'$ then uniquely defines q and n' for $q \in Q$ and $n' \in N$. Define n by $n = n'^{\bar{g}}$. Then $g = h\bar{g} = qn'\bar{g} = q\bar{g}n$. $\square$

Lemma 1 provides the basis for a perfect hash function for G and Q, since g and the corresponding coset Qg can be encoded by ϕ_1 and ϕ_2 as follows.

$$\text{For } g = q\bar{g}n \text{ as above }, \phi_1(g) = (q, \bar{g}, n) \text{ and } \phi_2(Qg) = (\bar{g}, n). \tag{1}$$

Given perfect hash functions for Q, G/H and N, the above equations yield perfect hash functions for G and for G/Q through a mixed radix encoding. The importance of this particular encoding is that it adapts well to an algorithm for fast multiplication.

If the group N is also normal, then there is an efficient multiplication of a triple $(q, \bar{g}, n)$ and a pair $(\bar{g}, n)$. Recall that the *conjugate* of a group element h by g is defined as $h^g = g^{-1}hg$. Recall that a subgroup $N < G$ is a *normal subgroup* if $\forall n \in N$, and $g \in G$, $n^g \in N$. This immediately implies $ng = gn^g$ and $n_1^g n_2^g = (n_1 n_2)^g$ for $n, n_1, n_2 \in N$ and $g \in G$.

Next, consider $g = q\bar{g}n$ as above. In addition, assume that N is a normal subgroup. Let $\bar{g}s$ decompose into $q'\overline{\bar{g}s}n'$ by Lemma 1, where $\overline{\bar{g}s}$ is the canonical coset representative of $H\bar{g}s$. Note also that $H\bar{g}s = Hgs$ implies $\overline{\bar{g}s} = \overline{gs}$ for $\overline{gs}$ the canonical coset representative of Hgs. Then the product of g by a generator s can be expressed as follows.

Lemma 2. *Let $Q < G$ and let N be a normal subgroup of G. Let $Q \cap N = \{1\}$. Let $g, s \in G$. Assume the decompositions $g = q\bar{g}n$ and $\bar{g}s = q'\overline{gs}n'$, as given by Lemma 1. Then the following holds.*

$$\text{If } g = q\bar{g}n \text{ and } \bar{g}s = q'\overline{gs}n', \quad \text{then } gs = (qq')\overline{gs}(n'n^s) \text{ and } Qgs = Q\overline{gs}(n'n^s). \tag{2}$$

Providing that Q, N and $G/(QN)$ are sufficiently small, Lemma 2 provides the foundation for precomputing small tables of all the required products, for any given $s \in G$. Typically, we choose s to be a generator since there are few generators, and so the relatively small size of the tables is maintained.

Fast multiplication by an arbitrary group element, while using small tables, is also feasible. By Lemma 1, an arbitrary group element can be written as $g = q\bar{g}n$. Assuming that g is represented as the triple $(q, \bar{g}, n)$, one can successively treat each of q, $\bar{g}$ and n as the generator s for purposes of multiplication on the right.

4.2. *Fast multiplication of coset by generator for edge group*

First, consider the edge group E, the restriction of Rubik's group to act only on edge facelets. Similarly, let M_E be the restriction of the generators of Rubik's cube, M, to elements that act only on the edges. Let Q_E be the restriction of the square subgroup $Q = \langle \{s^2 \mid s \in M\} \rangle$ to act only on the edges. Hence, $Q_E = \langle \{s^2 \mid s \in M_E\} \rangle$.

Let N_E be the normal subgroup of E that fixes the edge cubies setwise, but allows the two facelets of an edge cubie to be transposed. There are 12 edge cubies, but it is not possible in Rubik's cube to transpose the facelets of an odd number of edge cubies. Group theoretic arguments then show the order of the group N_E to be 2^{11}.

It is easy to show that N_E is normal in G since for $n \in N_E$ and $g \in G$, $g^{-1}ng$ may permute the cubies according to g^{-1}, but n fixes the cubies, and g then brings the cubies back to their original position. So, $g^{-1}ng \in N$ and so N_E is normal.

We describe the method for fast multiplication of a coset of E/Q_E by a generator from M_E. We define $H_E = Q_E N_E$. The method depends on a subgroup chain

$$E > H_E > N_E, \quad \text{for } N_E \text{ normal in } E, H_E \stackrel{\text{def}}{=} Q_E N_E, \quad \text{and} \quad Q_E \cap N_E = \{1\}.$$

For the remainder of this section, we often omit the subscripts of M_E, H_E, N_E and Q_E, since we will always be concerned with the action on edges.

Please cite this article in press as: Kunkle, D., Cooperman, G., Harnessing parallel disks to solve Rubik's cube. Journal of Symbolic Computation (2008), doi:10.1016/j.jsc.2008.04.013

Edge Tables	Size	Inputs	Output
Table 1a	$69300 \times 18 \times 2B$	r_1, s	$H\overline{r_1 s} \in E/H$ for $\overline{r_1 s}$ a canonical coset representative of E/H
Table 1b	$69300 \times 18 \times 2B$	r_1, s	$\bar{r}_2 \stackrel{\mathrm{def}}{=} n^{\overline{r_1 s}} \in N$, where n is defined by setting $h \stackrel{\mathrm{def}}{=} \overline{r_1 s}(r_1 s)^{-1} \in H$ and uniquely factoring $h = \bar{q}n$ for $\bar{q} \in Q, n \in N$
Table 2	$2048 \times 18 \times 2B$	r_2, s	$r_2^s \in N$
Logical op's		r_2, r_2'	$r_2 r_2' \in N$ (using addition mod 2 on packed fields)

Fig. 1. Edge tables for fast multiplication.

By Lemma 1, for any coset $Qg \in E/Q$, $Qg = Qr_1r_2$. So, Qg is represented as a pair (r_1, r_2) for r_1 a canonical coset representative in E/H and $r_2 \in N$. Given a generator s of E, Eq. (2) is used below to multiply the pair (r_1, r_2) by s and return a new pair, $(r_1', r_2') = (\overline{r_1 s}, \bar{r}_2(r_2^s))$.

$$\text{Let } r_1 s = \bar{q}\,\overline{r_1 s}\,\bar{r}_2, \quad \text{where } \bar{q} \in Q, \bar{r}_2 \in N, \text{ and } \overline{r_1 s} \text{ is the canonical coset rep. of } Qr_1 s.$$
$$\text{Then } Qr_1r_2s = Qr_1s(r_2^s) = Q\overline{r_1 s}(\bar{r}_2(r_2^s)). \tag{3}$$

Given (r_1, r_2) and a generator s, one can compute (r_1', r_2') such that $Qr_1r_2s = Qr_1'r_2'$ primarily through table lookup. Fig. 1 describes the necessary edge tables.

Note that the logical operations can be done efficiently, because N is an *elementary abelian* 2-group. This means that the group N is isomorphic to an additive group of vectors over a finite field of order 2. In other words, multiplication in N is equivalent to addition in $\mathrm{GF}(2)^{11}$, the 11-dimensional vector space over the field of order 2. Addition in the field of order 2 can be executed by "exclusive or". Hence, it suffices to use bitwise "exclusive or" over 11 bits for group multiplication in N_E.

4.3. Extension to group action on corners

For corners, we use the same logic as previously, but with the corner group C, the restriction M_C of the generators to corners, and the restriction Q_C of the square group to corners. Let N_C be the subgroup of C that fixes in position the corner cubies, but allows the facelets of the corner cubie to be permuted. There are 8 corner cubies, but a standard group-theoretic algorithm shows that within the subgroup C, the number of group elements fixing all corner cubies is only 3^7. As before, we drop the subscripts for readability. Hence, the tables of Fig. 1 can be reinterpreted as pertaining to corners. However, there is a small difference. Since N_C is an elementary abelian 3-group, multiplication in N_C is equivalent to addition over $\mathrm{GF}(3)^7$.

4.4. Generalization to fast multiplication over symmetrized cosets

Assume that the automorphism group A acts on edge facelets and corner facelets, separately preserving edge and corner facelets. Assume also that A preserves the subgroups Q, N and $H = QN$. Therefore, A also maps the projections of Q, N and $H = QN$ into edges and into corners.

Assume that the symmetrized coset Qg^A is uniquely represented as $Q(r_1r_2)^A$, where r_1 is the canonical representative of a symmetrized coset Hg^A with $H = QN$, and $r_2 \in N$, as described in Lemma 1.

The subscript e, below, indicates the restriction of a permutation to its action only on edges. Similarly, the subscript c is for corners. The subscripts are omitted where the meaning is clear.

For edges,

$$Q\alpha(r_{1,e}r_{2,e}s) = Q\alpha(r_1s)\alpha(r_2^s) = Q\overline{\alpha(r_1s)}(\bar{r}_2\alpha(r_2^s)), \quad \text{where } \overline{\alpha(r_1s)} \text{ is defined by}$$
$$\text{Table 1a, } \alpha \text{ is chosen to minimize } \overline{\alpha(r_1s)}, \bar{r}_2 \text{ defined by Table 1b, etc.} \tag{4}$$

However for corners,

$$Q\alpha(r_{1,c}r_{2,c}s) = Q\alpha(\overline{r_1s}\,(\bar{r}_2(r_2^s))) = Q\alpha(\overline{r_1s})\alpha(\bar{r}_2(r_2^s)) = Q\overline{\alpha(\overline{r_1s})}n^{\overline{\alpha(\overline{r_1s})}}\alpha(\bar{r}_2(r_2^s)),$$

Edge Tables	Size	Inputs	Output
Table Aut	$1564 \times 18 \times 1$B	$r_{1,e}, s$	$\alpha \in A$ for $\overline{\alpha(r_1 s)}$ a canonical coset rep. of H in E
			(We choose α such that $\overline{\alpha(r_1 s)} = \min_{\beta \in A} \overline{\beta(r_1 s)}$.)
Table 1a	$1564 \times 18 \times 2$B	$r_{1,e}, s$	$H\overline{\alpha(r_1 s)} \in E/H$ for α defined in terms of r_1 and s by Table Aut.
(coset rep.)			(Note that $H^A = H$.)
Table 1b	$1564 \times 18 \times 2$B	$r_{1,e}, s$	$\bar{r}_2 \stackrel{\text{def}}{=} n'^{\overline{\alpha(r_1 s)}} \in N$, where $h' \stackrel{\text{def}}{=} \alpha(r_1 s)\overline{\alpha(r_1 s)}^{-1} \in H$ and $h' = \bar{q}' n'$
(N)			for $\bar{q}' \in Q$, $n' \in N$ for α defined in terms of r_1 and s by Table Aut
Table 2	$2048 \times 18 \times 2$B	$r_{2,e}, s$	$r_2^s \in N$
Table 5	$2048 \times 48 \times 2$B	$n_e \in N, \alpha$	$\alpha(n) \in N$, where α is the output of Table Aut
			n is defined by $n = r_2^s$ (output of Table 2 for edges)
Logical op's		$r_{2,e}, r'_{2,e}$	$r_2 r'_2 \in N$ (using addition mod 2 on packed fields)

Fig. 2. Edge tables for fast multiplication of symmetrized coset by generator.

Corner Tables	Size	Inputs	Output
Table 1a	$420 \times 18 \times 2$B	$r_{1,c}, s$	$H\overline{r_1 s} \in C/H$ for $\overline{r_1 s}$ a canonical rep. of a coset of C/H
Table 1b	$420 \times 18 \times 2$B	$r_{1,c}, s$	$\bar{r}_2 \stackrel{\text{def}}{=} n'^{\overline{r_1 s}} \in N$, where n' is defined by setting $h \stackrel{\text{def}}{=} r_1 s \overline{r_1 s}^{-1} \in H$
			and uniquely factoring $h = \bar{q} n'$ for $\bar{q} \in Q$, $n' \in N$
Table 2	$2187 \times 18 \times 2$B	$r_{2,c}, s$	$r_2^s \in N$
Table 4a	$420 \times 48 \times 2$B	$H\overline{r_{1,c} s} \in C/H, \alpha$	$H\overline{\alpha(\overline{r_1 s})} \in C/H$, where $H\overline{r_1 s}$ is the output of Table 1a,
(coset rep.)			and α is the output of Table Aut on edges
Table 4b	$420 \times 48 \times 2$B	$H\overline{r_{1,c} s} \in C/H, \alpha$	$n^{\overline{\alpha(\overline{r_1 s})}} \in N$, where $H\overline{r_1 s}$ is the output of Table 1a,
(N)			and n is defined by setting $h = \alpha(\overline{r_1 s})\,\overline{\alpha(\overline{r_1 s})}^{-1} \in H$,
			and uniquely factoring h into qn for $q \in Q$, $n \in N$
Table 5	$2187 \times 48 \times 2$B	$n_c \in N, \alpha$	$\alpha(n) \in N$, where α is the output of Table Aut on edges, and n defined
			by computing $\bar{r}_2 = n'^{\overline{r_1 s}}$ (as in Table 1b), and r_2^s computed
			as in Table 2, and $\bar{r}_2 r_2^s$ computed by logical op's on corners
Logical op's		$r_{2,c}, r'_{2,c}$	$r_2 r'_2 \in N$ (using addition mod 3 on packed fields)

Fig. 3. Corner tables for fast multiplication of symmetrized coset by generator.

$$\text{where } \alpha \text{ is chosen as in Eq. (4), } \overline{r_1 s} \text{ defined by Table 1a, } \overline{\alpha(\overline{r_1 s})} \text{ defined by Table 4a,}$$
$$n^{\overline{\alpha(\overline{r_1 s})}} \text{ defined by Table 4b, } \bar{r}_2 \text{ defined by Table 1b, } r_2^S \text{ by Table 2, etc.} \qquad (5)$$

Note that the choice of $\alpha \in A$ for edges above depends on there being a unique such automorphism that minimizes $\overline{\alpha(r_1 s)}$. In fact, this is not true in about 5.2% of cases for randomly chosen r_1 and s. These unusual cases can be easily detected at run-time, and additional tie-breaking logic is generated. We proceed to describe tables for fast multiplication for the common case of unique $\alpha \in A$ minimizing $\overline{\alpha(r_1 s)}$, and discuss the tie-breaking logic later.

The tables that implement the above formulas follow. While it is mathematically true that we can simplify $\overline{\alpha(\overline{r_1 s})}$ into $\overline{\alpha(r_1 s)}$, we often maintain the longer formula to make clear the origins of that expression, which is needed for an implementation. As before, the subscripts e and c indicate the restriction of a permutation to its action only on edges and only on corners. Figs. 2 and 3 describe the following edge tables, among others.

Ideally, one would use only the simpler formula and tables for edges, and copy that logic for corners. Unfortunately, this is not possible. We must choose a representative automorphism $\alpha \in A$ for purposes of computation. We choose α based on the projection $r_{1,e}$ of r_1 into E (action of r_1 on edges). Hence, Tables 1a and 1b for edges take input r_1 and s, then compute α as an intermediate

Edge Tables	Size	Inputs	Output
Table Mult Aut	$48 \times 48 \times 1$B	α, β	the product $\alpha\beta \in A$
Table 1c (A)	$1564 \times 18 \times 1$B	$r_{1,e}, s$	$\{\beta \in A\colon\ \overline{\beta(\overline{\alpha(r_1 s)})} = \overline{\alpha(r_1 s)}\}$
Table 3 (N)	$2048 \times 48 \times 2$B	$H\overline{r_{1,c}s} \in C/H, \alpha$	$\bar{r}'_2 \stackrel{\text{def}}{=} n''^{\overline{\beta(r_1)}} \in N$, where β taken from Table 1c, and where $h'' \stackrel{\text{def}}{=} \beta(r_1)\overline{\beta(r_1)}^{\,-1} \in H$, and $h'' = \bar{q}''n''$, for $\bar{q}'' \in Q, n'' \in N$

Fig. 4. Edge tables for fast multiplication of symmetrized coset by generator, adjusted to break ties.

computation, then return $H\overline{\alpha(r_1 s)}$. A similar computation for corners is not possible, because the intermediate value α depends on $r_{1,e}$ and not on the corresponding element of the corner group $r_{1,c}$.

4.4.0.3. Tie-breakers: When the minimizing automorphism is not unique. Table Aut in the previous table for edges defines an automorphism α that minimizes $\overline{\alpha(r_1 s)}$. Unfortunately, there is not always a unique such α. In such cases, one needs a tie-breaker, since different choices of α will in general produce different encodings (different hash indices).

For each possible value of $\overline{\alpha(r_1 s)}$, with α chosen to minimize the expression, we precompute the stabilizer subgroup $B \le A$ defined by $B = \{\beta \in A\colon\ \overline{\beta(\overline{\alpha(r_1 s)})} = \overline{\alpha(r_1 s)}\}$ and use the formulas and additional table below to find the unique $\beta \in B$ such that the product $\alpha\beta$ minimizes the edge pair result $(r'_{1,e}, r'_{2,e})$. Where even this is not enough to break ties, we compute the full encoding, while trying all possible tying automorphisms. This latter situation arises only 0.23% of the time, and does not contribute significantly to the time. The tables of Fig. 4 suffice for these computations.

For edges,

$$Q\beta(\alpha(r_{1,e}r_{2,e}s)) = Q\beta(\alpha(r_1 s))\beta(\alpha(r_2^s)) = Q\beta(\overline{\alpha(r_1 s)})\left(\beta\left(\bar{r}_2\alpha(r_2^s)\right)\right) =$$
$$Q\overline{\alpha(r_1 s)}\bar{r}'_2\left(\beta\left(\bar{r}_2\alpha(r_2^s)\right)\right),$$

where $\overline{\alpha(r_1 s)}$ is defined by Table 1a, α is chosen to minimize $\overline{\alpha(r_1 s)}$,

$$\beta \in A \text{ satisfies } Q\overline{\beta(\overline{\alpha(r_1 s)})} = Q\overline{\alpha(r_1 s)},\ \beta(\overline{\alpha(r_1 s)}) = \overline{\alpha(r_1 s)}\bar{r}'_2\ (r'_2 \text{ defined in Table 3}),$$

and $\bar{r}_2$ defined by Table 1b for edges. (6)

However for corners,

$$Q\beta(\alpha(r_{1,c}r_{2,c}s)) = Q\beta(\alpha(\overline{r_1 s}\,(\bar{r}_2(r_2^s)))) = Q\beta(\alpha(\overline{r_1 s}))\beta(\alpha(\bar{r}_2(r_2^s))) =$$
$$Q\overline{\beta(\alpha(\overline{r_1 s}))}n^{\overline{\beta(\alpha(\overline{r_1 s}))}}\beta(\alpha(\bar{r}_2(r_2^s))), \text{ where } \alpha \text{ and } \beta \text{ are chosen as in Eq. (6)},$$

and other quantities based on the previous Corner Tables using $\alpha\beta$. (7)

Table 1c is implemented more efficiently by storing the elements of each of the possible 98 subgroups of the automorphism group, and having Table 1c point to the appropriate subgroup $B \le A$, stabilizing $r_{1,e}, s$.

4.5. Optimizations

In the discussion so far, we produce the encoding or hash index of a group element based on an encoding of the action of the group element on edges, along with an encoding of the action of the group element on corners. We can cut this encoding in half due to parity considerations.

Consider the action of Rubik's cube on the 12 edge cubies and the 8 corner cubies, rather than on the facelets. We define the *edge parity* of a group element to be the parity (even or odd) in its action on edge cubies. (Recall that the parity of a permutation is odd or even according to whether the permutation is expressible as an odd or even number of transpositions.) The *corner parity* is similarly defined.

The edge and corner parity of a symmetrized coset, Hg^A, are well-defined, and are the same as the edge and corner parity of g. This is so because $H = QN$, and elements of Q and N have even edge parity and even corner parity. Parity is unchanged by the action of an automorphism.

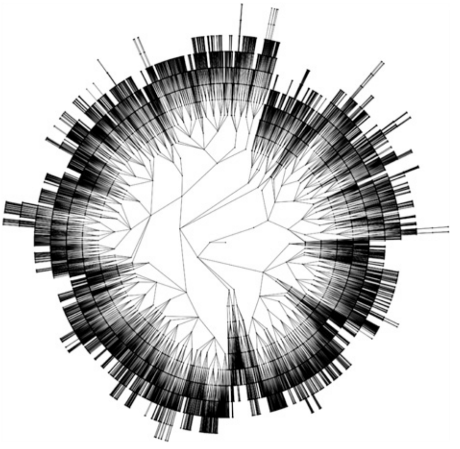

Fig. 5. Breadth-first enumeration of the symmetrized Cayley graph of the square subgroup.

For Rubik's cube, the natural generators have the edge parity equal to the corner parity. So this property extends to all group elements, and hence to all symmetrized cosets Hg^A. Therefore, our encoding can assume that edge and corner parities of symmetrized cosets are equal. The size of the corresponding hash table is thus reduced by half.

4.5.0.4. Nearly minimal perfect hash function. If we were to only use cosets instead of symmetrized cosets (no automorphism), then the perfect hash function that we have described implicitly would also be a *minimal perfect hash function*. However, there are examples for which $Q\alpha(g) = Qg$ for α not the identity automorphism.

A computation demonstrates that the perfect hash function of Section 4.4, with the addition of the parity optimization, has 1471074877440 $\approx 1.5 \times 10^{12}$ possible values. In Section 6, we show that there are 1357981544340 $\approx 1.4 \times 10^{12}$ symmetrized cosets. So, the ratio of these two values give the hash function an efficiency of 92.3%.

5. Square subgroup elements are within 13 moves of the identity

Because of the small size of the square subgroup (663,552 elements, or 15,752 after reduction by symmetries) these computations are small when compared to the corresponding computations for the cosets.

First, we constructed the Cayley graph of the square subgroup by breadth-first search, using the square generators. This computation took only seconds, even using a simple implementation on a single computer. It was found that all elements of the subgroup are within 15 moves of the identity.

Fig. 5 presents a visual representation of this breadth-first search. Each dot represents a symmetrized element and each line represents one of the six squared generators. The dot in the center is the identity element, and elements are grouped into concentric circles based on their distance from the identity. The plot shows a certain structure of the group, where initial moves away from the identity help dictate the maximum depth of large branches of the tree.

Next, we found an optimal solution for all elements of the square subgroup when any of the 18 generators are allowed to be used. Performing a breadth-first search to depth 15 using all 18 generators would be computationally expensive: an increase from enumerating 6^{15} to 18^{15} states (both numbers are smaller in practice due to duplicate detection). Because of this, we chose to use bidirectional search when allowing all 18 generators.

The computation is performed in two phases. First, we performed a forward search to depth 7, starting from the identity. Then, we performed one backward search for each of the 15,752 elements of the square subgroup, halting when the search frontier intersected the forward search. These

Square Generators			All Generators		
Dist.	Sym. Elts.	Elts.	Dist.	Sym. Elts.	Elts.
0	1	1	0	1	1
1	1	6	1	1	6
2	2	27	2	2	27
3	5	120	3	5	120
4	18	519	4	18	519
5	56	1932	5	62	2124
6	162	6484	6	214	8188
7	482	20310	7	693	27636
8	1258	55034	8	1871	78644
9	2627	113892	9	4093	174521
10	4094	178495	10	5394	233504
11	4137	179196	11	2774	116010
12	2231	89728	12	620	22228
13	548	16176	13	4	24
14	114	1488			
15	16	144			
Total	15752	663552	Total	15752	663552

Fig. 6. Distribution of elements in the square subgroup.

backwards searches required anywhere from milliseconds to a few hours in the worst case. Overall, this optimization took less than one day, requiring no parallelization.

This showed that all elements of the square subgroup have a solution of 13 or fewer moves. Fig. 6 shows the distribution of distances of (symmetrized) elements in the square subgroup, using either just the square generators or the full set of generators.

6. Cosets are within 16 moves of the trivial coset

We constructed the symmetrized Schreier coset graph, with respect to the square subgroup, through breadth-first search. However, the computation is significantly more complex than typical breadth-first search due to the scale of the computation: approximately 1.4 trillion symmetrized cosets. To handle this large scale computation we developed fast multiplication of symmetrized cosets by generators (see Section 4), and developed new methods for disk-based search and enumeration.

Robinson et al. (2007a) presents a comparative analysis of several different methods for parallel disk-based search and enumeration. The two methods from that comparison which we use here are *hash-based delayed duplicate detection* (hash-based DDD) with an *implicit open list*.

The primary data structure is an almost perfectly dense hash array of approximately 1.5×10^{12} entries, corresponding to the range of hash indices for all symmetrized cosets. The hash function is provided by our method of fast multiplication. Each entry of the table holds a four bit value describing the depth at which the corresponding symmetrized coset occurs in the breadth-first search, for a total array size of 685 GB.

Note that we could have used only two bits per state, encoding which of the states are on the current search frontier, instead of representing the exact depth of each element. We chose to use the more detailed representation to allow for efficient post-processing of the data (such as extracting all of the states at a given level).

Algorithm 1 describes how this array is used to perform the search. It uses an iterative process with two phases, *generating* and *merging*.

The generating phase produces new states, buffering them to disk, and continues until either an entire level of the breadth-first search is complete, or until available disk space is exhausted. The merge phase reads the buffered states, and uses the hash values to do duplicate detection in RAM.

Dist.	Elements	Approx.	Dist.	Elements	Approx.
0	1		9	80741117	8.1×10^7
1	1		10	1028869318	1.0×10^9
2	3		11	12787176355	1.3×10^{10}
3	23		12	140352357299	1.4×10^{11}
4	241		13	781415318341	7.8×10^{11}
5	3002	3.0×10^3	14	421980213679	4.2×10^{11}
6	38336	3.8×10^4	15	330036864	3.3×10^8
7	490879	4.9×10^5	16	17	
8	6298864	6.3×10^6			
			Total	1357981544340	1.36×10^{12}

Fig. 7. Distribution of symmetrized cosets of the square subgroup.

Note that the non-duplicate values are not saved, but are encoded directly in the hash table and reconstructed using the efficient inverse hash function provided by our fast multiplication.

6.0.1. *Parallelization and optimizations*

In each phase, the computation is split into *buckets*, based on contiguous ranges of hash values. By doing so, the computation can be performed in parallel, with each parallel process handling a separate bucket. We call each of these bucket-based computations a *task*. Because of this natural task structure, we parallelized the computation using a library developed by the second author called TOP-C (Task Oriented Parallel C/C++) (Cooperman, 1996).

As an optimization, the breadth-first search is initially conducted using only main memory, as in the traditional case, and switches to the parallel disk-based version once RAM limitations are met (in this case, at depth 9).

Further, when storing new states on disk, we must store the associated hash index, which is 41 bits long. However, we store all of the states that fall into a particular bucket into a separate file or directory. So, the highest 11 bits of the hash value are stored implicitly, and only 30 bits per state must be stored explicitly, which fits into a single 4-byte value.

For efficiency, wherever possible, we use *double buffering* for reading or writing large amounts of data on disk. Basically, this involves transferring half of the data, while continuing computations with the other half.

For a long-running computation, checkpointing is essential. Notice that the end of phase two, the merging phase, is a natural time to checkpoint. At this point, all data is contained in the level array (on disk). We simply record the last level completed and the portions of the level array which have already been generated from (in the first phase). Given this, we can restart the computation after a failure, losing only the work since the end of the last merge phase. Notice that the update of the level array is idempotent, so that re-doing some generating and merging does not effect correctness.

6.1. *Experimental results*

The breadth-first search showed that all coset are within 16 moves of the trivial coset. Fig. 7 shows the distribution of depths for cosets in the symmetrized Schreier coset graph.

We have now performed this computation twice. The primary reason for repeating this experiment was to compare two different cluster computing architectures, namely: those using shared disk, such as a SAN (Storage Area Network); and those using distributed disk, with one or more disks locally attached to each compute node. Secondarily, repeating the computation gives more assurance of the correctness of the result.

For the shared disk architecture, we used the DataStar cluster at the San Diego Supercomputer Center (SDSC). We used 16 compute nodes in parallel, each with 8 computing cores and 16 GB of main memory. For external storage, we used DataStar's attached GPFS (IBM General Parallel File System).

Please cite this article in press as: Kunkle, D., Cooperman, G., Harnessing parallel disks to solve Rubik's cube. Journal of Symbolic Computation (2008), doi:10.1016/j.jsc.2008.04.013

Algorithm 1 Construct Symmetrized Schreier Coset Graph

```
Initialize array of symmetrized cosets with all levels set to unknown (four bits per coset).
Add trivial coset to array; set level ℓ to 0.
while previous level had produced new neighbors, at next level do
  {Generate new elements from the current level}
  Let a segment be those nodes at level ℓ among N consecutive elements of the array.
  Scan array starting at beginning.
  while we are not at the end of the array, extract next segment of array and do
    for each node at level ℓ (representing a symmetrized coset) do
      for each generator do
        Compute product by fast multiplication.
        Compute hash index of product.
        Save hash index in bucket b, where b is the high bits of the hash index. Note, we only save
        the low order bits of the hash index not encoded by the bucket number. (This value fits
        in four bytes.)
        If bucket b is full, transfer it (write it) to a disk file for bucket b.
      end for
    end for
    Transfer all buckets to corresponding disk files.
  end while
  {Now merge buckets into array of symmetrized cosets.}
  for each bucket b on disk do
    Load portion of level array corresponding to bucket b into main memory.
    for each buffered element on disk for this bucket do
      Read value into RAM and delete from disk (in large chunks).
      Look up corresponding level value in array.
      If a value already exists for the element, it is a duplicate. Otherwise, set its level to ℓ.
    end for
    Write portion of level array back to disk.
  end for
  Increment level ℓ.
end while
```

We used up to 7 terabytes of storage at any given time, as a buffer for newly generated states in the breadth-first search.

The fast multiplication algorithm allowed us to multiply a symmetrized coset by a generator at a rate of approximately 5 million times per second.

The computation using shared disk required 63 cluster hours, or approximately 8000 CPU hours.

For the distributed disk architecture, we use a recently purchased cluster at Northeastern University, called TeraCluster. Here, we used 30 nodes, using one computing core per node. For external storage, we used up to 80 GB per node, on a locally attached disk (or up to 2.3 TB of aggregate storage).

We used just one core per node because we found experimentally that executing more than one task simultaneously on a single node could significantly reduce performance. We hypothesize that this occurs when the two tasks disrupt the large contiguous disk accesses of the other, reducing them to smaller, less efficient disk operations. This was not seen in the shared disk architecture, where the SAN was composed of hundreds or thousands of disks. As future work, we intend to examine the use of multiple disks per node, in hopes that each additional disk could be used by a separate task.

The fact that we were using fewer CPUs in the distributed disk architecture is somewhat offset by the fact that the fast multiplication algorithm was twice as fast, performing approximately 10 million multiplications per second per computing core. This speed differential is likely due in part to the different cache architecture of the two CPUs.

Please cite this article in press as: Kunkle, D., Cooperman, G., Harnessing parallel disks to solve Rubik's cube. Journal of Symbolic Computation (2008), doi:10.1016/j.jsc.2008.04.013

The computation using distributed disk required 183 cluster hours, or around 5500 CPU hours. So, even though the total computation took longer, it was faster per CPU. This is because the computation is bound by the speed of the network and disks, not by CPU speed. It is encouraging that the distributed disk time was within a factor of three of the shared disk time, considering that our local cluster is more modest, and less expensive, than the large supercomputer at SDSC.

7. Refining the upper bound

Here, we present three separate methods for further reduction of the upper bound on the radius of Rubik's group, and of groups in general when using a two stage solution.

So far, we have shown that, from an arbitrary cube position, one can reach the square subgroup in no more then 16 moves, and can then solve the cube in no more than 13 moves, for an upper bound of 29 moves. The methods we present here were used to show that all cube configurations that require 27 or more moves by this method have an alternate solution of length 26 or less. Often, these alternate solutions require taking non-optimal paths in the first phase (reaching the square subgroup), in trade for a reduced solution length in the second phase.

The three methods we present are:

(1) *Optimal solvers*: used on a single group element and guaranteed to find the shortest possible solution. This operation is typically expensive, and can only be used on relatively few positions.
(2) *Image Intersection:* acts on an entire coset, but does not guarantee optimal results.
(3) *Projection:* uses the structure of breadth-first search to refine the upper bound on all cosets based on refinement at earlier depths.

7.1. Goal of refinement of coset upper bound

First, we define the process of refinement, and fix our terminology for this section.

Having constructed the Schreier coset graph, one wishes to test individual cosets, and prove that all group elements of that coset are expressible as words in the generators of length at most u.

Recall that G is the group of Rubik's cube, and Q is the square subgroup. Consider a coset Qg at a depth ℓ (distance ℓ from the home position, or trivial coset, in the coset graph). Let $d = 13$ be the diameter of the subgroup Q. For any group element $h \in Qg$, clearly its distance from the identity element is at least ℓ and at most $\ell + d$. We describe a computation to produce a finer upper bound on the distance of any $h \in Qg$ from the identity element.

Note that in the original computation proving bounds of 13 for the square subgroup and 16 for the cosets we reduced both of these spaces by the 48 symmetries of the cube. However, when refining the upper bound, we cannot make use of both of these symmetry reductions. In general, for $q \in Q$, $g \in G$, and $\alpha_1, \alpha_2, \alpha_3, \alpha_4 \in A$, $\mathrm{dist}(\alpha_1(q)\alpha_2(g)) \neq \mathrm{dist}(\alpha_3(q)\alpha_4(g))$, where $\mathrm{dist}(h)$ is the length of the shortest path from h to the identity. So, to refine the upper bound on a coset Qg, we must refine the bound on all elements in that coset, where Q is not reduced by symmetry. We still make use of the symmetry reduction for the cosets, by refining only one canonical coset Qg from each symmetrized coset Qg^A.

7.2. Refinement by optimal solver

Over the years since Rubik's cube was introduced, many solvers have been introduced for the puzzle, including a number of solvers that guarantee optimal solutions (given enough execution time).

The idea is straight forward: to refine the upper bound on a coset Qg, and prove that all elements have a solution of length u or less, one applies the solver to all elements qg, where $q \in Q$ and $\mathrm{dist}(q) > u - \ell$. If the resulting solution length for each of these elements is less than u, the refinement has succeeded. Otherwise, the refinement is not possible.

When using an optimal solver, this method is guaranteed to prove the refinement if it is possible. However, finding the optimal solution can take anywhere from seconds to hours or more, depending on the speed of the solver and the distance of the element from the identity. Because of this, we

Algorithm 2 Refinement by Image Intersection

Input: a subgroup Q of a group G, a coset Qg; a desired upper bound u; and a set of words $w_1, w_2, \ldots$ in generators of G such that $Qgw_i = Q$.

Output: a demonstration that all elements of Qg have solutions of length at most u or else a subset $S \subseteq Qg$ such that all elements of $Qg \setminus S$ are known to have solutions of length at most u.

1: Let $k = u - \text{len}(g)$. Let $U_0 = \{q \in Q \mid \text{dist}(q) > k\} \subseteq Q$. Then $(Q \setminus U_0)g$ is the subset of elements in the coset Qg which are known to have solutions of length at most u. The set $U_0 g$ is the "unknown set", for which we must decide if they have solutions of length u or less.

2: For each $i \geq 1$, let $U_i = U_{i-1} \setminus \{q \in U_{i-1} \mid \text{dist}(qgw_i) \leq u - \text{len}(w_i)\}$. (Note that $qgw_i \in Q$). By $\text{dist}(qgw_i)$, we mean the shortest path in the full set of generators of G. If $\text{dist}(qgw_i) \leq u - \text{len}(w_i)$, then qg has a solution of length at most u. The solution for qg is given by a path length $\text{len}(w_i)$ followed by a path of length $u - \text{len}(w_i) = \text{dist}(qgw_i)$.

3: If $U_i = \emptyset$ for some $i \leq j$, then we have shown that all elements of Qg have solutions of length at most u. If $U_j \neq \emptyset$, then we have shown that all elements of $(Q \setminus U_j)g$ have solution length at most u.

typically only use optimal solvers when we wish to refine the bound only on a small number of cosets and/or when there are few elements to consider per coset.

We made use of two solvers previously developed by the community. The first is Cube Explorer, developed by Kociemba (2007). This is currently the most efficient, and most developed solver for Rubik's cube.

The second solver we made use of is based on software developed by Winter (1992). This solver uses a method similar to Cube Explorer, but does not make use of some of the more advanced features or the large precomputations performed by Cube Explorer. However, it is a simple C program, which we relatively easily modified to run in parallel on our cluster computer.

7.3. Refinement by image intersection

The method we introduce here, *image intersection*, attempts to overcome the expense of considering each individual element in a coset in isolation. It is typically much more efficient than optimal solvers, especially when there are many elements to solve per coset.

Note that for the coset Qg, there can be many paths in the coset graph from the identity coset to Qg. In terms of group theory, there are multiple words, $w_1, w_2, \ldots$, where each word is a product of generators of Rubik's group, and $Qw_1 = Qw_2 = \cdots = Qg$. Note that in general, the words are distinct group elements: $w_1 \neq w_2$, and $qw_1w_2^{-1} \neq q$, for $q \in Q$. Nevertheless, $w_1w_2^{-1} \in Q$. The different images of Q produced through multiplication with the different words is the key to finding a refined upper bound.

Next, suppose our goal is to demonstrate an upper bound $k + \ell$ for all of the elements in the coset Qg (at depth ℓ), where $\ell \leq k + \ell < d + \ell$.

Let $Q_k \stackrel{\text{def}}{=} \{q \in Q \mid \text{dist}(q) \leq k\}$, the subset of Q at distance from the identity at most k, and let $Q_k g \stackrel{\text{def}}{=} \{qg \mid q \in Q_k\}$. Let w_i be words such that $Qw_1 = Qw_2 = \cdots = Qg$. Assume the words are of length ℓ in the generators of Rubik's group, i.e., they are shortest words for g.

Note that for all elements of $Q_k w_1$, there is an upper bound, $k + \ell$. Similarly, for all elements of $Q_k w_2$, there is an upper bound, $k + \ell$. Therefore, the elements of $Q_k w_1 \cup Q_k w_1$ have an upper bound of $k + \ell$. More compactly,

$$\text{dist}(Q_k w_1 \cup Q_k w_2) \leq k + \ell.$$

More generally, for w_i a word in the generators of G of length $\text{len}(w_i)$, let $x_i = \text{len}(w_i) - \ell$ represent the number of additional moves used by w_i beyond a shortest path. Then

$$\text{dist}(Q_{k-x_1} w_1 \cup Q_{k-x_2} w_2) \leq k + \ell$$

since the length of any word in $Q_{k-x_1} w_1$ is at most $(k - x_1) + \text{len}(w_1) = k + \ell$ and similarly for w_2.

Define the complement $Q_j' \stackrel{\text{def}}{=} Q \setminus Q_j$. We can now write:

$$Q'_{k-x_1} w_1 \cap Q'_{k-x_2} w_2 \supseteq \{h \in Qg \mid \text{dist}(hg) > k + \ell\}.$$

Clearly, the above equation can be generalized to the intersection of multiple words, $w_1, w_2, \ldots$.

$$\cap_i Q'_{k-x_i} w_i = \emptyset \Longrightarrow \forall i, \qquad \text{dist}(Q w_i) = \text{dist}(Qg) \le k + \ell.$$

For purposes of computation, Algorithm 2 captures these insights.

Finally, a few notes on implementation details for the above algorithm:

- Note that $gw_i \in Q$. So, for $q \in Q$, qgw_i can be computed by fast multiplication within the subgroup Q.
- Determining whether $\text{dist}(qgw_i) \le u - \text{len}(w_i)$ can be done by maintaining a hash table mapping all elements $q \in Q$ to $\text{dist}(q)$.
- Finding words from the target coset to the identity can be achieved in many ways, including depth-first, breadth-first, and bidirectional search.

7.4. *Refinement by projection*

Refining by *projection* refers to the use of new upper bounds discovered at cosets within ℓ moves of the trivial coset to refine the bounds on all cosets further than ℓ moves from the trivial coset.

In its simplest form, consider the case where all of the cosets at depth ℓ have a refined upper bound of u, where $\ell \le u < d + \ell$ (proven by one of the above methods). Then, all subsequent cosets at depth $m > \ell$ have a refined upper bound of $m - \ell + u$. This is because any of the elements in a coset at depth m can reach a coset at depth ℓ in $m - \ell$ moves, and then complete the solution in u moves.

We can also use this same logic when the cosets at depth ℓ have only been *partially refined*. For example, assume we are attempting to refine the upper bound of Qg, at depth ℓ, to u, but there remain some unrefined elements. There is a set of elements $S \subset Qg$, where for all $s \in S$ there is a known solution proving $\text{dist}(s) \le u$. Then, any element $sw \in Qgw$, where $s \in S$, w is a word in the generators, and Qgw is a coset at depth $m = len(w) + \ell$, has a refined upper bound of $u + len(w)$.

In other words, all elements in cosets beyond depth ℓ that have a shortest path to a coset in level ℓ resulting in a refined element, is also refined. So, by recording the set of refined elements for all cosets at a given depth, we can easily determine the corresponding set of refined elements at the next depth by performing one step in a breadth-first search, recording any elements that are the neighbor of a refined element.

Using this technique, we can use expensive refinement techniques (e.g. optimal solvers, or finding many words for image intersection) at early depths where there are very few cosets, and project those results to later depths, which have exponentially many more cosets.

7.5. *Experimental results*

We used two steps to reduce the upper bound on solutions to Rubik's cube from 29 to 26. First, we refined the bound on all cosets at depth 9 by 2 moves, reducing the overall bound to 27. Then, we refined all cosets at depth 16, the furthest depth, by an additional move, proving 26 moves suffice overall.

We did not directly refine the cosets at depth 9 by 2 moves, as this would have required considering 22252 elements (the number of elements at depths 12 or 13 of the square subgroup) for each of the 80741117 depth 9 cosets (or almost 2 trillion total elements). Instead, we started by attempting to refine cosets at depth 2 by 2 moves, and projected those results to later levels. By interleaving the use of refinement by optimal solvers and image intersection with projection at each level, we were able to minimize the total number of elements considered.

Both the optimal solver and image intersection methods were parallel computations, simultaneously refining several cosets. Typically, we used 60 computing cores across 30 nodes of a cluster.

Please cite this article in press as: Kunkle, D., Cooperman, G., Harnessing parallel disks to solve Rubik's cube. Journal of Symbolic Computation (2008), doi:10.1016/j.jsc.2008.04.013

Depth	Cosets	Elts. Left	Method	Depth	Cosets	Elts. Left	Method
2	3	66756	Starting Elts.	6	38336	37833	Projection
		222	Image Inters.			26065	Image Inters.
						12991	Optimal Solver
3	23	2652	Projection	7	490879	166425	Projection
		1311	Image Inters.			77555	Optimal Solver
		44	Optimal Solver				
4	241	586	Projection	8	6298864	990419	Projection
		482	Image Inters.			152940	Optimal Solver
		293	Optimal Solver				
5	3002	3756	Projection	9	80741117	2752920	Projection
		2891	Image Inters.			0	Optimal Solver

Fig. 8. Number of remaining elements at each depth during refinement.

In some cases, the optimal solver was given an execution time limit, after which it would halt execution on a given element. Due to this, the optimal solver may have failed to refine some elements which it could have at some earlier depths.

The experimental results of this refinement are shown in Fig. 8. The table shows: the total number of cosets at depths 2 through 9; the total number of remaining unrefined elements across all cosets at each depth; and the resulting number of elements left after each refinement technique was used. The decision of which technique to use at each point was based on timing estimates of the various methods.

The entire process was completed over the course of approximately two weeks, though that period includes down time between computations to analyze results and setup the next step. If the three methods of refinement were more tightly integrated and automated, the entire process would likely take just a few days on a cluster of computers.

Lastly, we refined the cosets at depth 16, showing that all elements there can be solved in 26 or fewer moves. We did this by applying an optimal solver to the elements corresponding to the last three levels of the square subgroup across each of the cosets, and verifying that a solution of 26 of fewer moves was found. With 138262 elements at depth 11 or greater in the square subgroup, and 17 cosets at depth 16, this resulted in 2350454 elements to solve. This is roughly equivalent to the last refinement, at depth 9, of the previous series of computations.

References

Cooperman, G., 1996. TOP-C: A Task-Oriented Parallel C interface. In: 5th International Symposium on High Performance Distributed Computing. HPDC-5. IEEE Press, pp. 141–150. software at: http://www.ccs.neu.edu/home/gene/topc.html.

Cooperman, G., Finkelstein, L., Sarawagi, N., 1990. Applications of Cayley graphs. In: AAECC: Applied Algebra, Algebraic Algorithms and Error-Correcting Codes, International Conference. In: LNCS, Springer-Verlag, pp. 367–378.

Frey Jr., A.H., Singmaster, D., 1982. Handbook of Cubik Math. Enslow Publishers.

Isaacs, S., 1981. Lower bounds. http://www.math.rwth-aachen.de/~Martin.Schoenert/Cube-Lovers/Stan_Isaacs__lower_bounds.html.

Kociemba, H., 2007. Cube Explorer. http://kociemba.org/cube.htm.

Korf, R., 1997. Finding optimal solutions to Rubik's cube using pattern databases. In: Proceedings of the Workshop on Computer Games (W31) at IJCAI-97, Nagoya, Japan, pp. 21–26.

Korf, R.E., 2004. Best-first frontier search with delayed duplicate detection. In: AAAI. pp. 650–657.

Korf, R.E., Schultze, P., 2005. Large-scale parallel breadth-first search. In: AAAI. pp. 1380–1385.

Kunkle, D., Cooperman, G., 2007. Twenty-six moves suffice for Rubik's cube. In: ISSAC '07: Proceedings of the 2007 International Symposium on Symbolic and Algebraic Computation. ACM, New York, NY, USA, pp. 235–242.

Radu, S., 2006. Rubik can be solved in 27f. http://cubezzz.homelinux.org/drupal/?q=node/view/53.

Reid, M., 1995a. New upper bounds. http://www.math.rwth-aachen.de/~Martin.Schoenert/Cube-Lovers/michael_reid__new_upper_bounds.html.

Reid, M., 1995b. Superflip requires 20 face turns. http://www.math.rwth-aachen.de/~Martin.Schoenert/Cube-Lovers/michael_reid__superflip_requires_20_face_turns.html.

Please cite this article in press as: Kunkle, D., Cooperman, G., Harnessing parallel disks to solve Rubik's cube. Journal of Symbolic Computation (2008), doi:10.1016/j.jsc.2008.04.013

Robinson, E., Cooperman, G., 2006. A parallel architecture for disk-based computing over the Baby Monster and other large finite simple groups. In: Proc. of International Symposium on Symbolic and Algebraic Computation, ISSAC '06. ACM Press, pp. 298–305.
Robinson, E., Kunkle, D., Cooperman, G., 2007a. A comparative analysis of parallel disk-based methods for enumerating implicit graphs. In: PASCO '07: Proceedings of the 2007 International Workshop on Parallel Symbolic Computation. ACM, New York, NY, USA, pp. 78–87.
Robinson, E., Müller, J., Cooperman, G., 2007b. A disk-based parallel implementation for direct condensation of large permutation modules. In: ISSAC'07: Proceedings of the 2007 International Symposium on Symbolic and Algebraic Computation. ACM, New York, NY, USA, pp. 315–322.
Rokicki, T., 2008a. Twenty-five moves suffice for Rubik's cube. http://arxiv.org/abs/0803.3435.
Rokicki, T., 2008b. Twenty-three moves suffice. http://cubezzz.homelinux.org/drupal/?q=node/view/117.
Winter, D.T., 1992. Kociemba's algorithm. http://www.math.rwth-aachen.de/~Martin.Schoenert/Cube-Lovers/Dik_T._Winter__Kociemba's_algorithm_(3).html.
Zhou, R., Hansen, E.A., 2004. Structured duplicate detection in external-memory graph search. In: AAAI. pp. 683–689.