

Large Scale Computations in Discrete Mathematics

Anton Betten

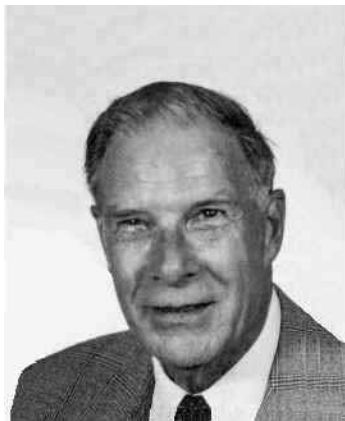
Department of Mathematics
Colorado State University

July 2008

Overview

- What are we trying to compute and how?
- What are the issues with developing software?
- What are the issues with doing the actual computation?

What are we trying to compute and how?



Richard Hamming:

The purpose of computing is insight, not numbers.

What are we trying to compute and how?

Goal:

Use computers to find new objects / examples.

Use brain/intuition to investigate once they have been found.

Find their “reason for being” (for instance by exhibiting a general construction principle) thereby gaining insight.

Often, a single new example can trigger a lot of new theory.

Nevertheless, new examples are often hard to come by.

What are we trying to compute and how?

The types of objects in Discrete Mathematics are abstract, they are hard to visualize. Expect no pictures in this talk! :-)

Areas that I have considered:

- Optimal Linear Codes
- Incidence Structures (0, 1-matrices with specific properties).
- Objects in Projective Spaces over finite fields (BLT-sets).

The types of problems that I look at are those for which computer search is needed. These problems would be very messy to solve by hand.

What are we trying to compute and how?

We wish to **classify** all objects of a certain type.

This requires computer search.

The computer traces through a **search tree**.

The tree is made up of all possible partial objects.

What are we trying to compute and how?

We are interested in the leaves of the tree.

The tree is “bushy” which makes leaves hard to find.

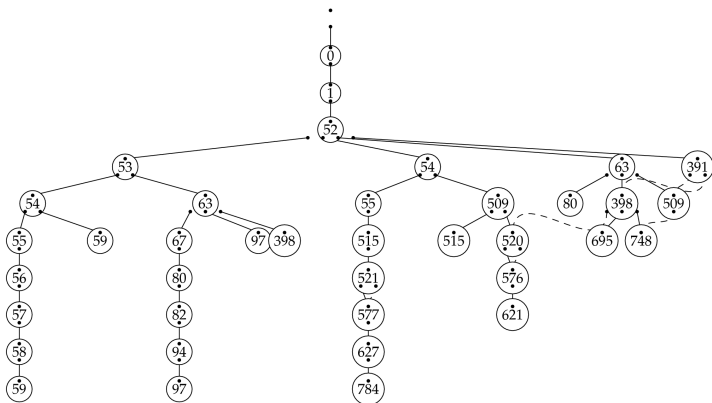
Isomorph rejection is the process of **pruning branches** once they have been identified as equivalent.

Pruning is most effective near the root of the tree.

Pruning is expensive since it requires **isomorph testing**.

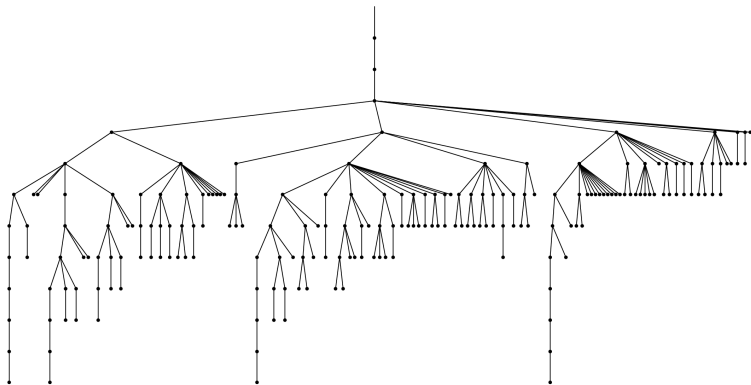
What are we trying to compute and how?

The search tree for BLT-sets in $O(5, 9)$:



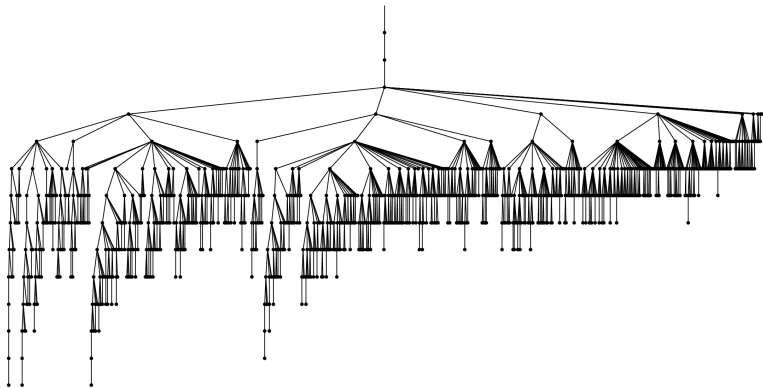
What are we trying to compute and how?

The search tree for BLT-sets in $O(5, 11)$:



What are we trying to compute and how?

The search tree for BLT-sets in $O(5, 13)$:



What are we trying to compute and how?

For large problems, we use a hybrid method to do the search:

We do **breadth first search** in the top part of the tree with full isomorph rejection

Using Clique finding algorithms (from graph theory), we do deep search to get to the leaves. There is no isomorph rejection in this part.

We do a **post mortem** isomorph rejection.

What are we trying to compute and how?

Compare with:

Writing a **chess program**.

We can try to spend a lot of time for evaluating any given position (intelligent search, slow)

or

we can try to simply run through all possible moves and see what could happen (simple search, fast)

What are the issues with developing software?

Our software package:

- Written in $C++$.
- 100,000 lines of code.
- Took ≥ 15 years to develop.
- Parts rewritten several times.
- External software: Brendan McKay's nauty library for graph isomorphism.

What are the issues with developing software?

Main components:

- Computer Algebra Components:
 - Finite field arithmetic
 - Finite groups (permutation groups and matrix groups)
- Polymorphy: the same code can deal with permutation groups and matrix groups.
- Efficiency: Matrices over finite fields are coded in bitfields for more efficient storage.
- Isomorphism programs (finding canonical forms)
- Orbit algorithms (finite groups acting on sets)
- Graph and network algorithms (clique finding)

What are the issues with developing software?

- $C++$ was chosen because it allows to manage large projects with many classes. It also allows to write nice code.
- Speed is a big issue: Writing efficient code is critical.
- General purpose packages (GAP, Magma, Maple, Matlab) are not efficient enough.
- We noted that some $C++$ constructs are unreasonably slow (example: virtual functions, function calls in general).
- Efficient code often tends to be somewhat ugly (and unreadable).
- Memory management is an issue (as is generally in Computer Algebra)
- Code is 64 bit compliant.

What are the issues with developing software?

Issues:

- Software development is extremely time intense (the Computer Algebra part)
- Arithmetic with finite groups not readily available elsewhere.
- Isomorphism problem is very complicated.
- Software development takes far too long for the ordinary NSF funding cycles.
- Memory management: How much memory is used? Are there any leaks?

What are the issues with developing software?

“Memory leaks:”

My feeling is that they were easier to debug in *C* than they are in *C++*.

This is because in *C*, everything regarding memory ran through `malloc / free` and it was easy to build a wrapper around these two. In *C++*, the language provides a somewhat cumbersome approach to allocation / deallocation.

See also: “My Rant on *C++*’s operator `new`” by David Mazières, Stanford Computer Science Department.

What are the issues with doing the actual computation?

“Combinatorial explosion:”

The problem size scales (super-)exponentially:

Graph isomorphism is a problem for which **no polynomial algorithm** is known (and there are no indications that any one will be found).

Large scale computations need to be performed.

What are the issues with doing the actual computation?

“All or nothing:”

We need to search **100%** of the search tree in order to get results.

This is different from simulations (in Applied Math, say), where we might get a (less accurate) solution if we invest a fraction of the time.

What are the issues with doing the actual computation?

“Perfect Parallelization:”

The problem can be parallelized trivially by slicing the search tree into pieces.

The parallelism scales perfectly.

Example 1

Classification of BLT-sets of $O(5, q)$ for q and odd prime power.

- $q = 31$: computing time: 2 days (done)
- $q = 37$: computing time: 200 days (done)
Comment of my colleague: this cannot be done.
- $q = 41$: computing time: 1700 days (estimated, in work, using parallel computing).

Example 1

What insight did we gain?

- $q = 31$: We verified that all examples were known previously (and there are no more).
- $q = 37$: We found one **new example**.
- $q = 41$: We expect to find new examples. We hope to find one with trivial stabilizer. This would be the first of its kind.

Ultimately, the hope is to come up with new infinite families by analyzing the data.

Example 2

Classification of a certain type of linear spaces on 30 points:

Computation took more than 1300 days (3.5 years).

This is work in progress.

Example 3

Classification of optimal linear codes:

Lots of parameter studies.

Found two infinite families by looking at the data.

Exhibited a construction principle: the twisted tensor product.

This was a known principle (from finite group theory), but up until now, it has not occurred in connection with optimal linear codes.

What are the issues with doing the actual computation?

I have (or had) access to

- Two double processor Intel Xeon machines in my office.
- Cluster at the Computing Center of the University of Kiel / Germany.
- APAC (Australian Partnership for Advanced Computing), a cluster of unix machines.

At this moment, I have around 25 processors running in Kiel computing BLT sets for $q = 41$.

What are the issues with doing the actual computation?

I would be interested in hearing about computing facilities on Campus.

Thank you for your attention!