Approximate Congruence Detection of Model Features for Reverse Engineering

C. H. Gao, F. C. Langbein, A. D. Marshall and R. R. Martin

ralph@cs.cf.ac.uk

Department of Computer Science,
Cardiff University,
Wales, UK

Contents

- Problem Statement
- Previous Work
- Approximate Congruence Detection Algorithm
- Analysis
- Experimental Results
- Conclusions

Problem Statement–1

Reverse engineered models suffer from inaccuracies caused by:

- Data acquisition errors—sensors are not perfect
- Approximation and numerical errors during reconstruction
- Original model errors, such as wear of the model, manufacturing method of the model

Faces are recovered individually.

- Models built from these faces do not show expected and desired regularities.

Problem Statement–2

Our Goal

- Reconstruct an ideal model of a physical object with intended geometric regularities

Our approach:

- Beautification: improve the model in a post-processing step,
- first detect intended regularities
- then impose them on the model

This talk considers detecting congruent *features* (sub-parts of the model).

- Other work has considered simple regularities, symmetry, constraint enforcement, d-o-f analysis.

Previous Work

Exact congruence detection: Alt, Akutsu

- their algorithms calculate the transformation between two point sets
- these algorithms can be *hard to implement*

Approximate congruence detection: Alt, Schirra

- their algorithms are based on distance checking between
  - two *point* sets
  - with a *given* tolerance
- these algorithms have a *high* cost

Algorithm Overview

- Find congruent pairs of seed faces
- Expand seed pairs to neighbouring faces which keep congruence (giving congruent features)
- Avoid re-using any given congruent pair of faces
- Avoid rechecking congruence of face pairs for efficiency
- Congruence checking is done using hypothesise-and-test for efficiency
  - Find candidate isometry for 4 points from each set
  - Verify by mapping remaining points

Congruence of Two Faces

Our models have planar, cylindrical, spherical, conical and toroidal faces (and blends).
A face of given type is determined by

- a sufficient set of points to determine face parameters
- the patch boundary curves

Matching these for two patches ensures congruence.
A boundary curve of given type is determined by

- a sufficient set of points to determine curve parameters
- and end points

Face Parameters

A surface is uniquely defined by the following numbers of points (in general position):

- Planar face: 3
- Spherical face: 4
- Cylindrical face 5
- Conical face: 6
- Toroidal face: 7

These points may be on the patch boundary for open surfaces but must include at least one interior point for closed surfaces.

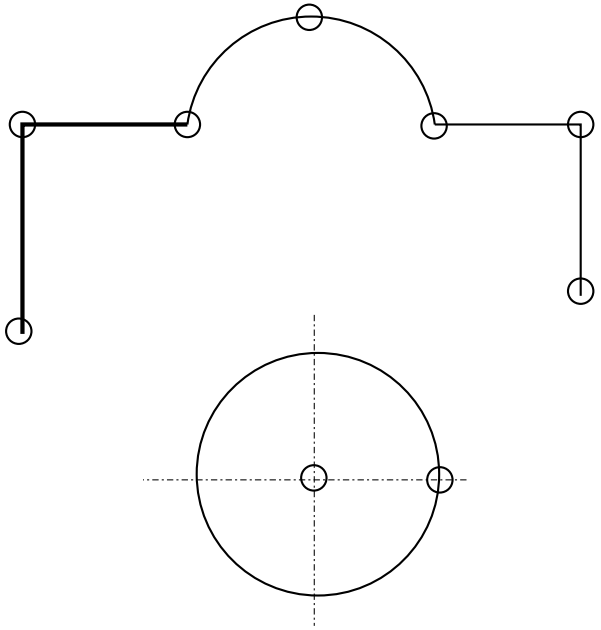
Congruence of Two Edges–1

Straight line segments:

- take both end points as the characteristic points

Circular arcs:

- take end points and the mid-point of the arc



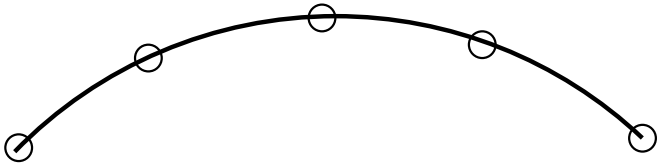
Congruence of Two Edges–2

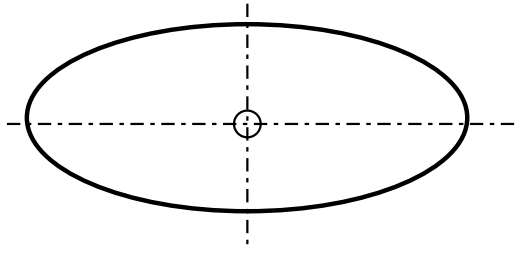
Elliptical arcs:

- take points $\frac{1}{4}$, $\frac{1}{2}$, $\frac{3}{4}$ of the way around the arc.

NURBS edges:

- use suitable control points (see algorithms due to Cohen, Hu and Ma et al.)
- may need degree elevation, reparametrisation etc.





Main Algorithm

The main algorithm

- looks for seed congruences and grows them

In detail, it

- calls `compatible` to quickly eliminate face pairs which can not be congruent
- calls `congruence` to check for approximate congruence between every compatible face pair in the model
- calls `expand` to extend each congruent face pair found, not already used, to include adjacent faces, then their neighbours, and so on.

The `compatible` Method–1

This method quickly decides whether a face pair can potentially be congruent.
Two faces can only be congruent if they satisfy all the following requirements:

- They must be of the same type (planar, cylindrical, etc.)
- Cylinders, cones, spheres and tori must have the same convexity or concavity
- Radii of cylinders, cones, spheres and tori must agree
- Semi-angles of cones must agree

The `compatible` Method–2

- Faces must have the same number of edge loops
- Corresponding loops must agree in type (external, internal or end loops)
- Corresponding loops must have the same number of edges
- Corresponding edges must agree in length
- Corresponding vertices must have the same number of edges around them
- Face types around corresponding vertices must be consistent

The congruence Method

This method decides if two faces, or two sets of faces, are congruent. The steps are as follows:

- collect the characteristic points from each face set
- compute a special tetrahedron T_1 for the first set (guaranteed to be large)
- perform a search for (maybe multiple) congruent tetrahedra $T_{21}, T_{22}, \ldots$ in the second set
- compute the mapping relating T_1 to each T_{2i}
- check if the remaining points in each set agree under this mapping, in each case
- return the result, and the mapping(s) if congruent

The expand Method

This method takes a seed pair of congruent faces, and expands it to two congruent features, as follows:

- put each face from the seed pair in a list
- find a new pair of faces which are neighbours of some face in each list respectively
- test if these two faces are congruent (pre-test using compatible)
- if congruent, add each face to the respective list and compute congruence of the two lists
- if lists congruent, keep new pair, else discard it
- continue until there are no neighbouring pairs left to consider.

Algorithm—Remarks

As stated, this algorithm would be very inefficient:

- each face pair would potentially be re-tested for congruence many times

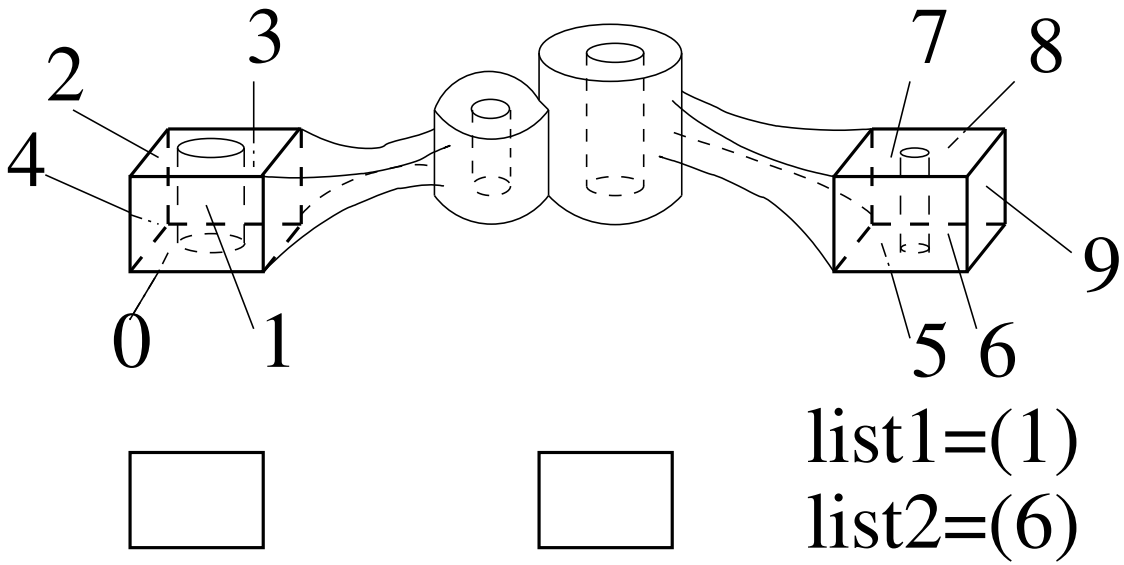
So, we cache the result of testing congruence of face pairs.

We merge multiple congruences at the end of the algorithm e.g.

- $\{A, B\}, \{B, C\} \rightarrow \{A, B, C\}$

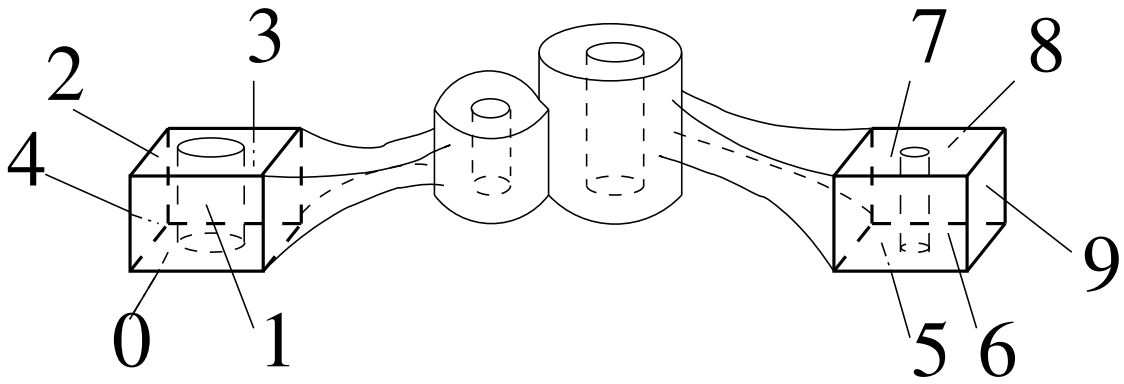
Demonstrative Example–1

Faces 1 and 6 are detected as a seed pair of congruent faces

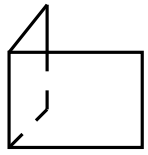


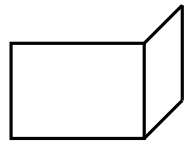
Demonstrative Example–2

Faces 4 and 9 are adjacent to 1 and 6, are congruent, and extend the seed congruence



list1=(1)
list2=(6)

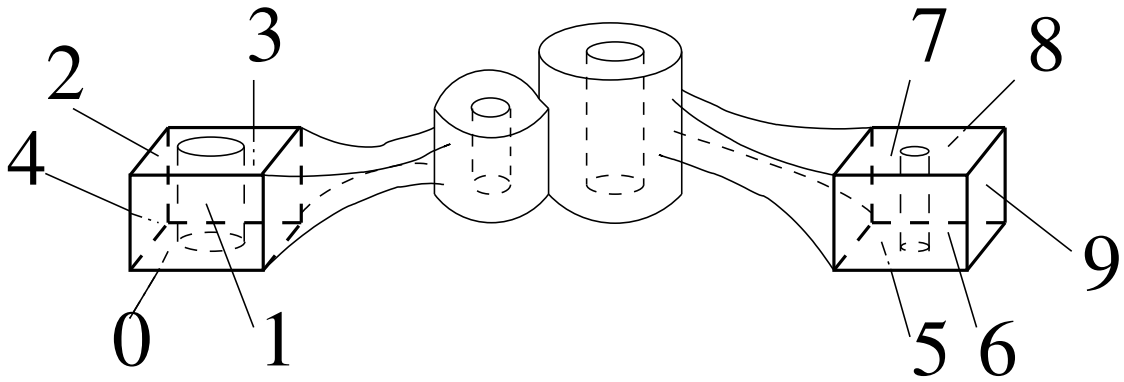


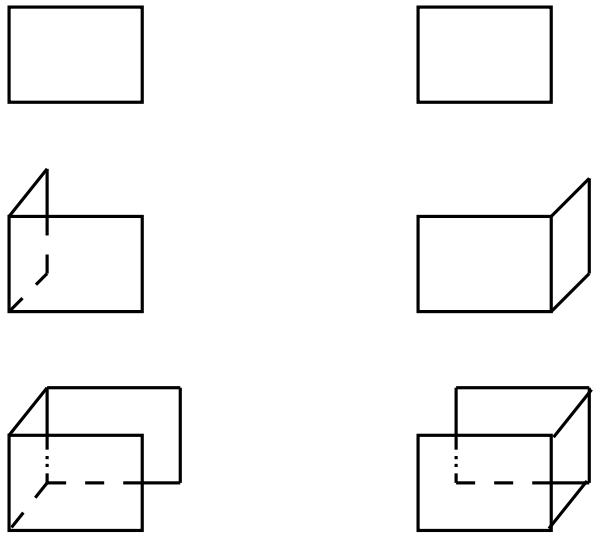


list1=(1 4)
list2=(6 9)

Demonstrative Example–3

Faces 2 and 7 are not congruent. Faces 3 and 8 are congruent, and extend the congruence





list1=(1)
list2=(6)

list1=(1 4)
list2=(6 9)

list1=(1 3 4)
list2=(6 8 9)

Algorithm Analysis–1

First consider the `congruence` checker component:

- Suppose `congruence` is called on two sets of p points.
- p is bounded for each type of surface.
- No more than $O(p^{1.5})$ matching tetrahedra are possible (see references)
- Testing each takes time $O(p^2)$.

So, overall, the `congruence` method takes time $O(p^{3.5})$.

Algorithm Analysis–2

Next, consider the whole algorithm.

Instead of a worst case analysis, we make some assumptions:

- we are dealing with "real engineering objects", so:
- each face has a fixed maximum number of neighbours $\leq m$, a constant
- each vertex has a fixed maximum number of edges $\leq m$, a constant

Most objects to be reverse engineered will be of this type.

Algorithm Analysis–3

Now consider three particular cases:

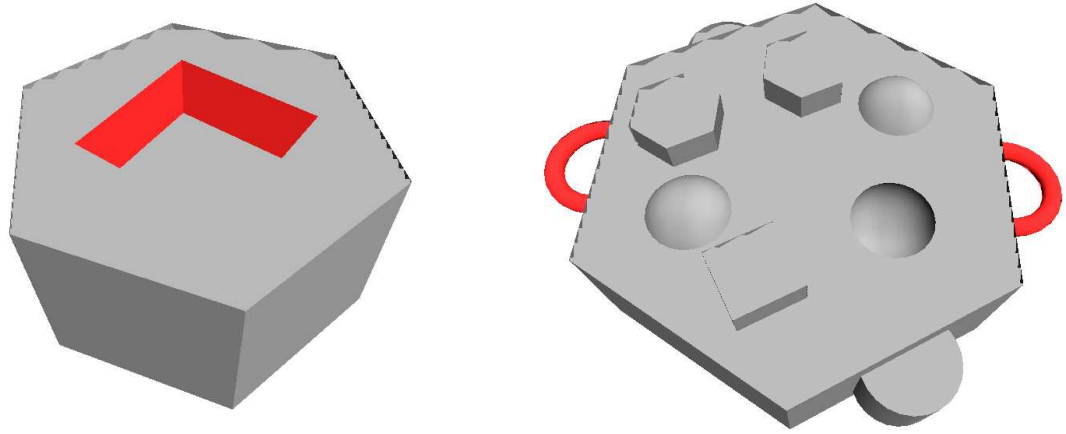
1. the object has no congruent features.
   $O(n^2)$ pairs of faces are checked for congruence, each in time $O(m^{3.5})$. Overall time is $O(n^2)$.
2. there are $n/2$ separate congruences, each comprising a single congruent face pair.
   Similarly, overall time is $O(n^2)$.
3. a single congruence relates the two halves of the object.
   This takes time $m \sum_{i=1}^{n/2-1} O((mi)^{3.5}) = O(n^{4.5})$

Expected complexity lies between $O(n^2)$ and $O(n^{4.5})$.

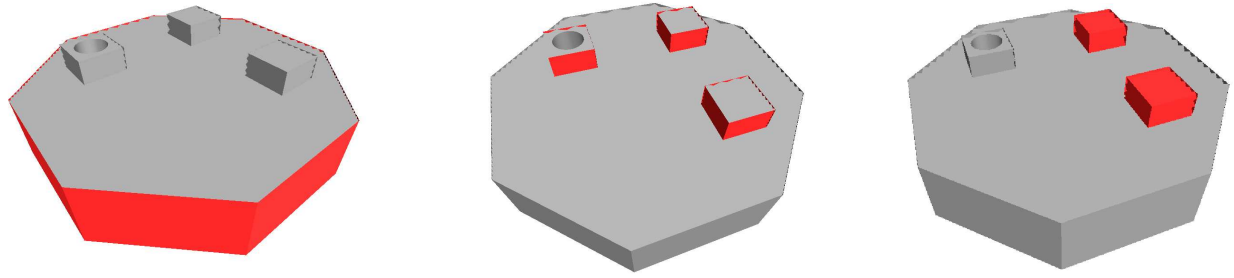
Testing Congruence Detection–1

(Some) congruent features detected shown in red

Test objects 1 and 2

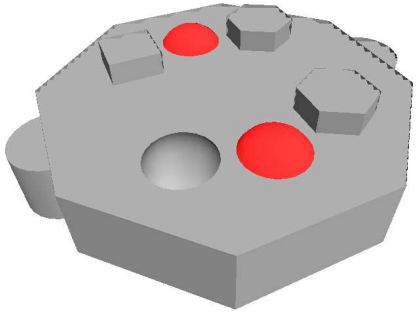


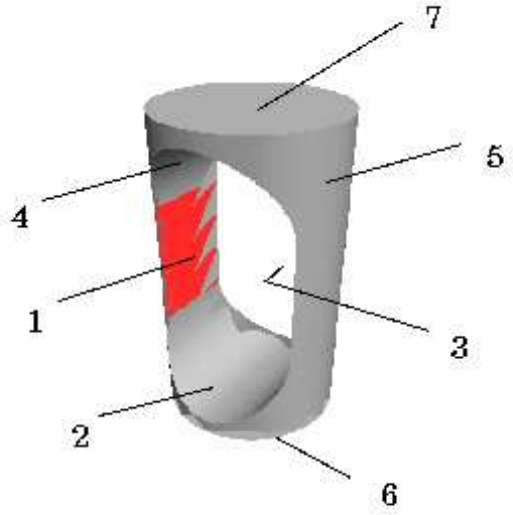
Test object 3

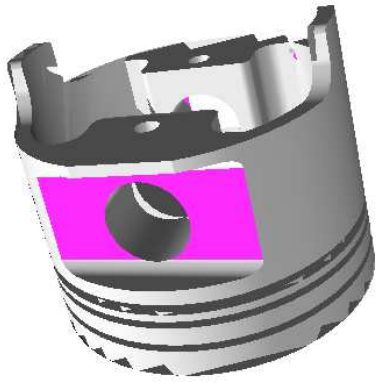


Testing Congruence Detection–2

Test objects 4, 5, 6





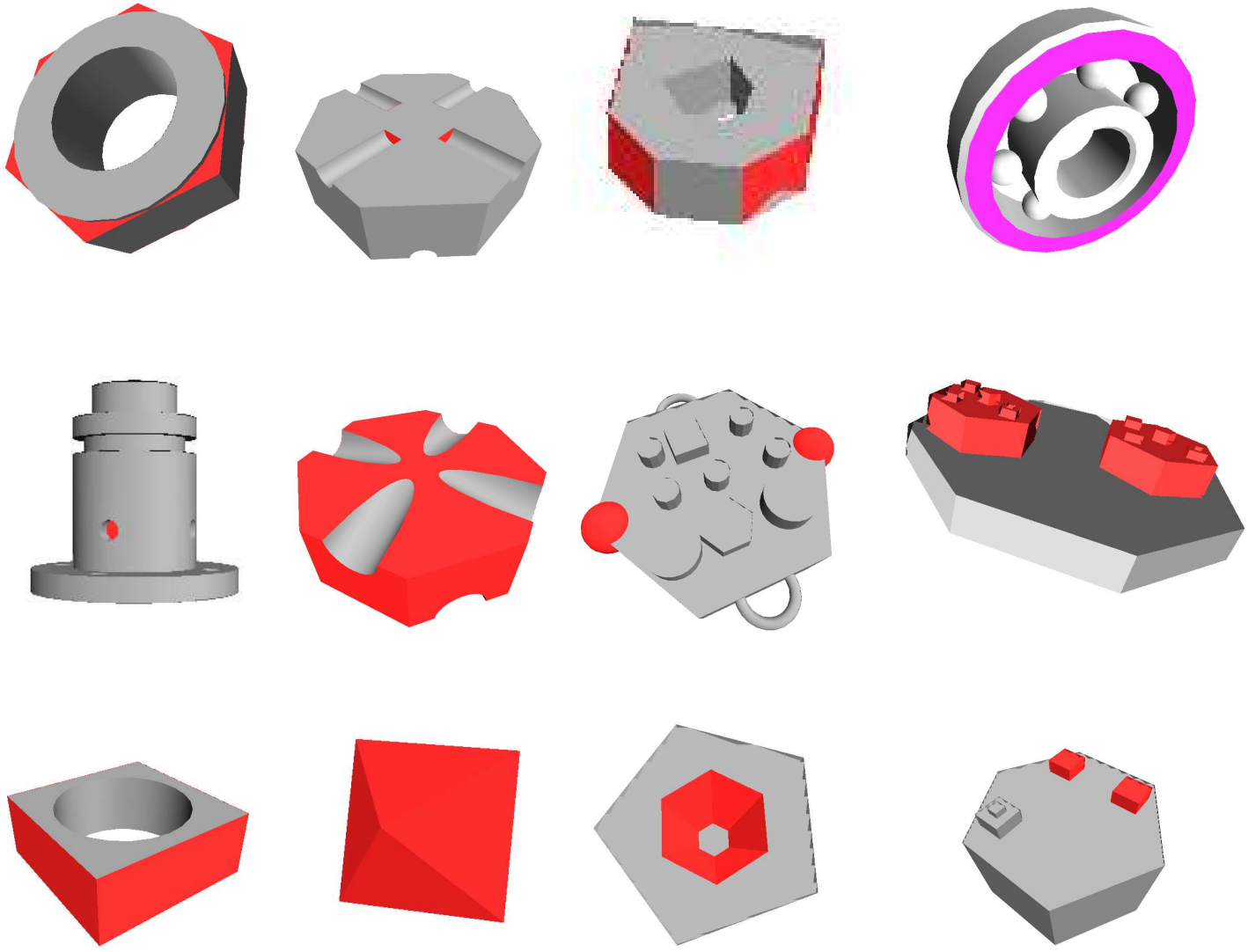


Congruences Found and Times

Object	Faces	Congruences	Time (s)
1	13	2	2.20
2	38	5	9.88
3	33	5	10.40
4	27	3	5.45
5	7	3	1.43
6	70	20	21.19

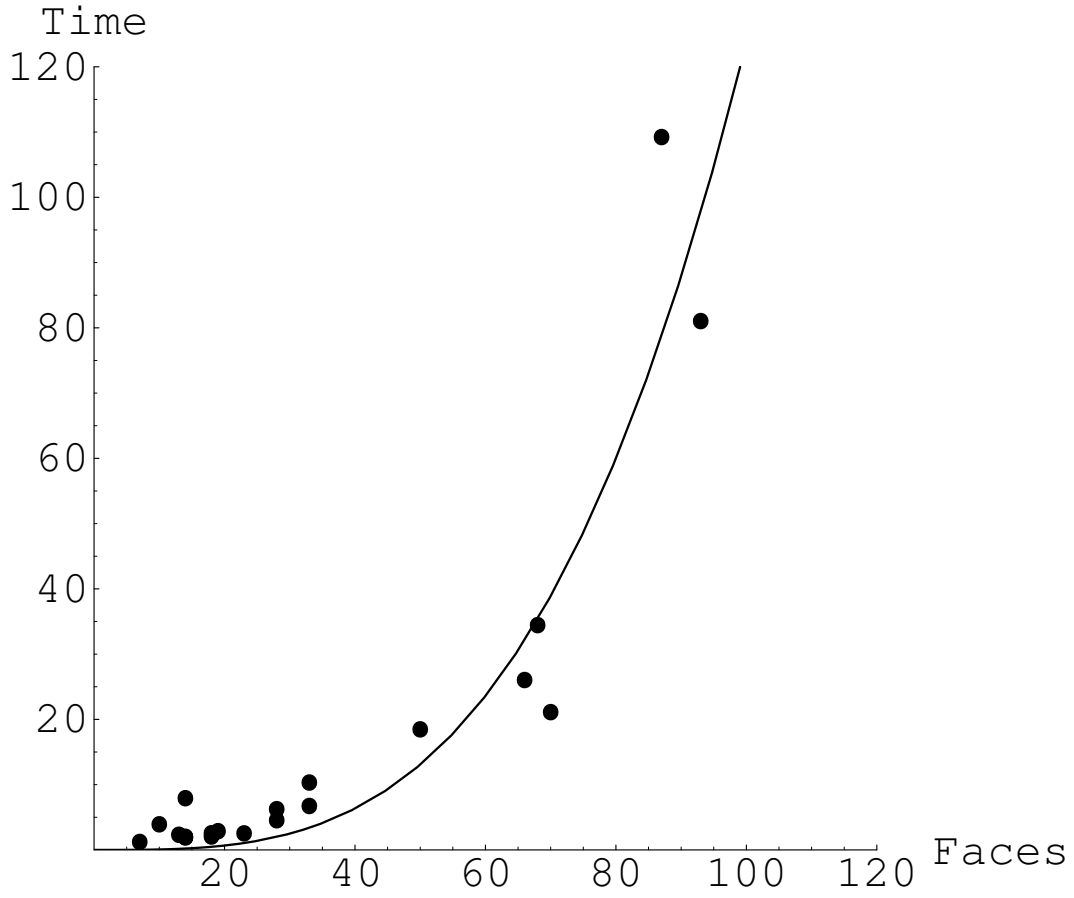
Further Testing—Congruences

Further test models



Algorithm Performance Test

We plotted time taken (seconds) versus number of faces for all test cases:



The best fit to the timing data is $O(n^{3.24})$.

Conclusions

- An algorithm for approximate congruence detection has been developed,
- with reverse engineering in mind.
- It can handle simple non-planar faces.
- It finds all expected congruences.
- It runs in an acceptable length of time for moderately complex models.