Extracting dimensions from games

It might be argued that ... having the program select terms for the evaluation polynomial from a supplied list is much too simple and that the program should generate the terms for itself. Unfortunately, no satisfactory scheme for doing this has yet been devised. With a man-generated list one might at least ask that the terms be members of an orthogonal set, assuming that this has some meaning as applied to the evaluation of a checker position. Apparently, no one knows enough about checkers to define such a set.

Arthur Samuel

Introduction

Intuitively, we think many games have important *dimensions of play*. In chess for instance:

- Strength of opening book
- Controlling the center
- Coordinating the rooks

But what are dimensions?

- Dimensions refers to a decomposition of the game. Can we do that? Are these just metaphors?
- Maybe a dimension is a function, mapping players to some number indicating their strength. Finding such functions automatically is often NP-complete, though;

Pareto coevolution to the rescue

- Pareto coevolution suggests a different concept of dimension: sets of tests. The idea is that, to see where a candidate lies, test it against all tests in a dimension-set and place it accordingly.
- This is like geometry —to place a point on a dimension, you "test" it against the coordinate's values until you find where it fits.
  - [here's a picture]

An Algorithm

- Remarkably, there is an $O(n^4)$ algorithm which can extract these dimensions;
- But how? Many decompositions are NP-complete;
- Because, *by using sets of tests instead of functions, we have weakened the representational power of a dimension.*

Features

- Not too expensive. Uses $O(n^2)$ plays and $O(n^4)$ computations on the outcome matrix —so, can run online (Pareto coev and hBOA are ~$O(n^3)$);
- Guaranteed to work under certain conditions;
- Data-neutral; can run on any outcome matrix, even from humans (e.g., chess tourny results, spelling bee results).

Pseudocode

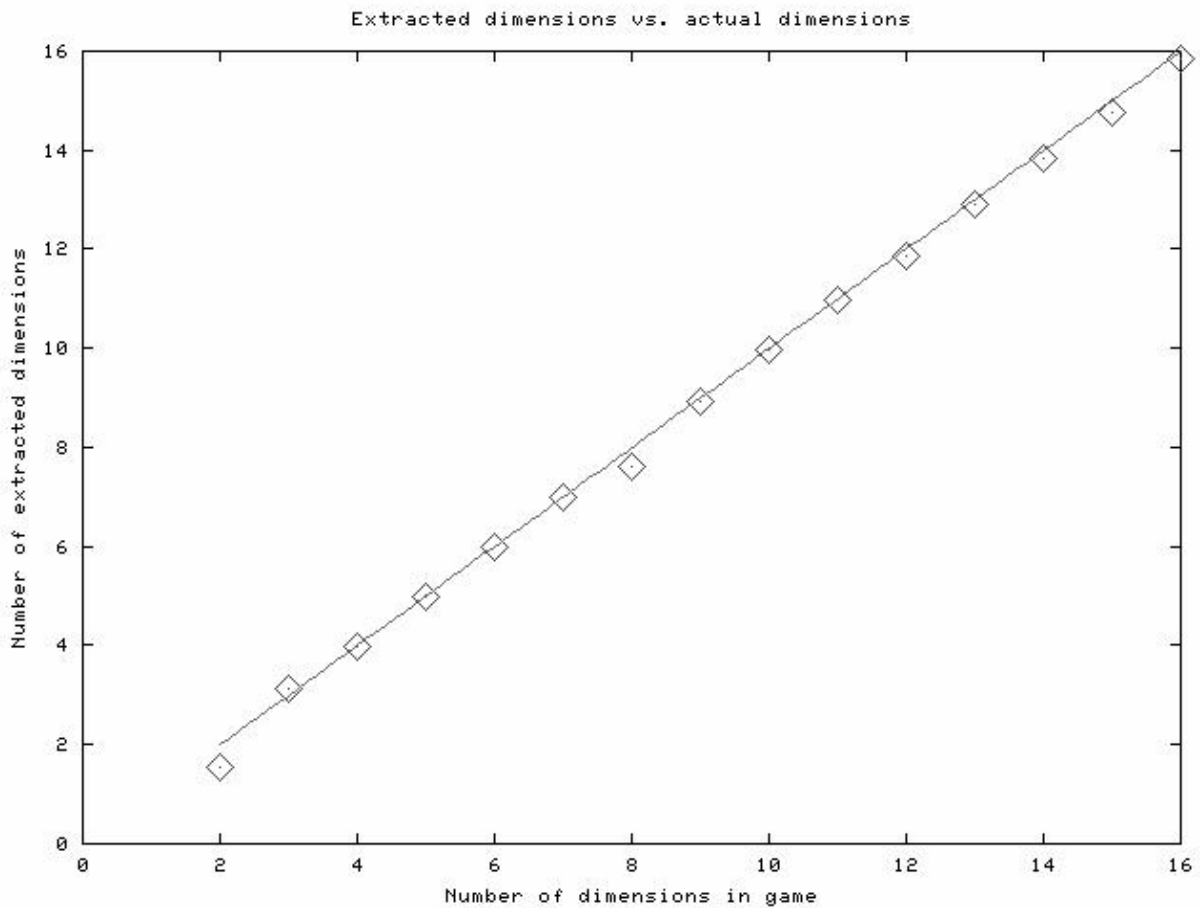
```
input:
List Candidates,Tests
boolean play(cand,test)
boolean ConsistentWith(test1,test2)
Test and(test1,test2)
output:
Tree dimensions

sort tests by number of fails (bad/lose outcomes)

/* eliminate redundant tests */
for each test1,test2,test3 in Tests (with all three distinct)
     if test3 == and(test1,test2)
          eliminate test3

/* add test in appropriate spot */
for each test in Tests
     for each leaf in leaves(dimensions)
          if ConsistentWith(test, leaf)
               add test as child to leaf
if test was not added to a leaf
     add test as child to root(dimensions)
```

Results


Extracted dimensions vs. actual dimensions
Number of extracted dimensions
Number of dimensions in game
0
2
4
6
8
10
12
14
16

Potential uses

- Online diagnostic. If an algorithm starts losing dimensions, it is focusing;
- Estimating how "hard" a problem is, resources needed. Or even *online* resource reallocation;
- A new way of thinking about games —we can do this to any game matrix;
- Knowledge extraction —generalize over dimensions.

Fun Questions

- After we have extracted dimensions, can we see what they have in common?
- Could the extracted dimensions correspond to what we as humans feel are important dimensions of play?
  - Maybe, because coevolution is like learning by self-play