Tag-based Modularity in Tree-based Genetic Programming

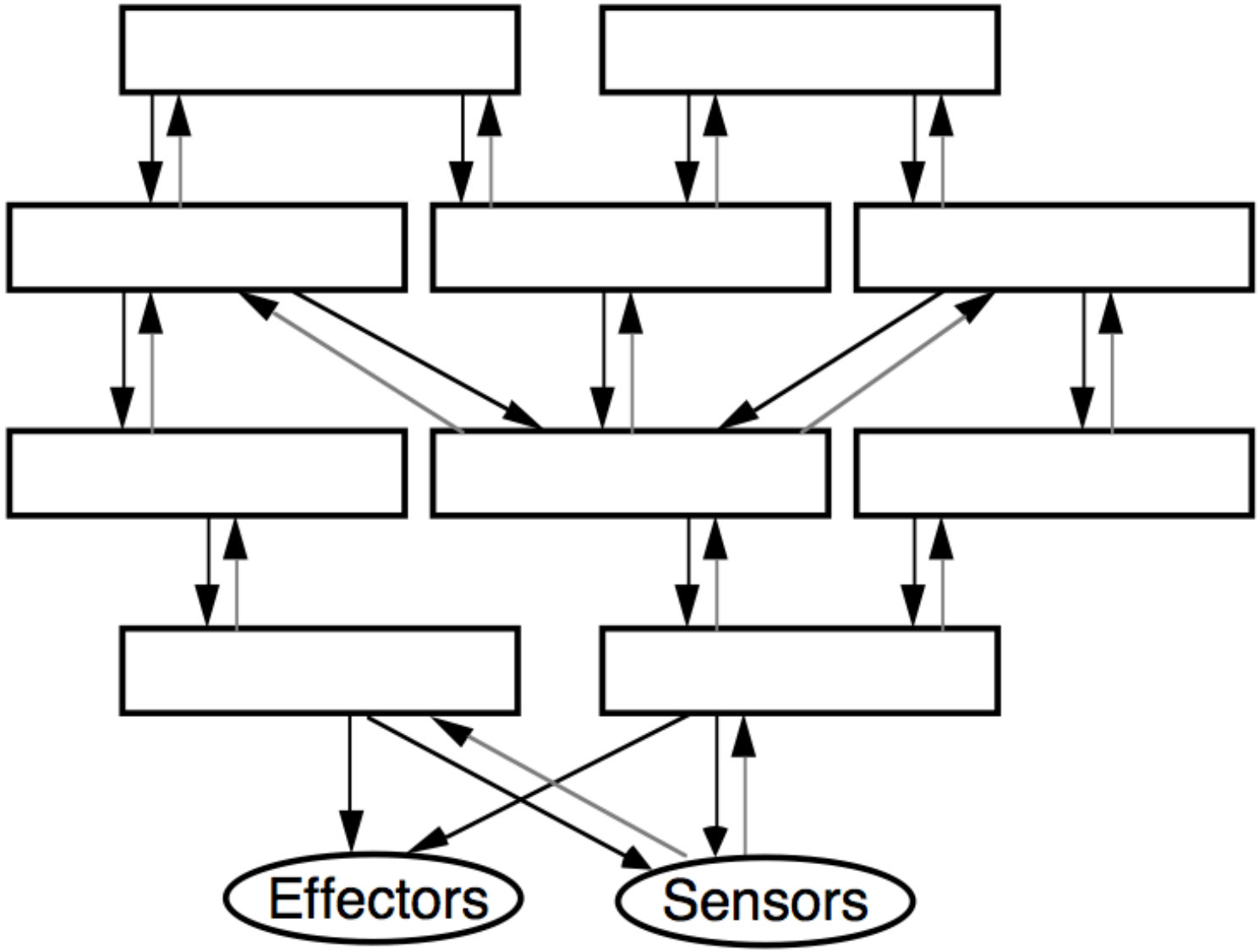
Lee Spector, Kyle Harrington & Thomas Helmuth

Cognitive Science, Hampshire College
Computer Science, Brandeis University
Computer Science, University of Massachusetts, Amherst

Outline

- Evolving modular programs
- Tags
- Evolving tag-based modules in PushGP
- Can we do the same thing in tree-based GP?
- Experiments and Conclusions
- Prospects

Modularity is Everywhere



Modules in GP

- Automatically-defined functions (Koza), macros (Spector)
- Architecture-altering operations (Koza)
- Module acquisition/encapsulation systems (Kinnear, Roberts, many others)
- Modules in GE (Swafford et al., others)
- In Push: code-manipulation instructions that build/execute modules as programs run

ADFs

- All programs in the population have the same, pre-specified architecture
- Genetic operators respect that architecture
-
```
(progn (defn adf0 (arg0 arg1) ...)
       (defn adf1 (arg0 arg1 arg2) ...)
  (.... (adf1 ...) (adf0 ...) ...))
```
- Complicated, brittle, limited...
- Architecture-altering operations: more so

Tags

- Roots in John Holland's work on principles of complex adaptive systems
- Applied in models of the evolution of altruism, with agents having tags and tag-difference thresholds for donation
- A tag is *an initially meaningless identifier that can come to have meaning through the matches in which it participates*
- Matches may be inexact

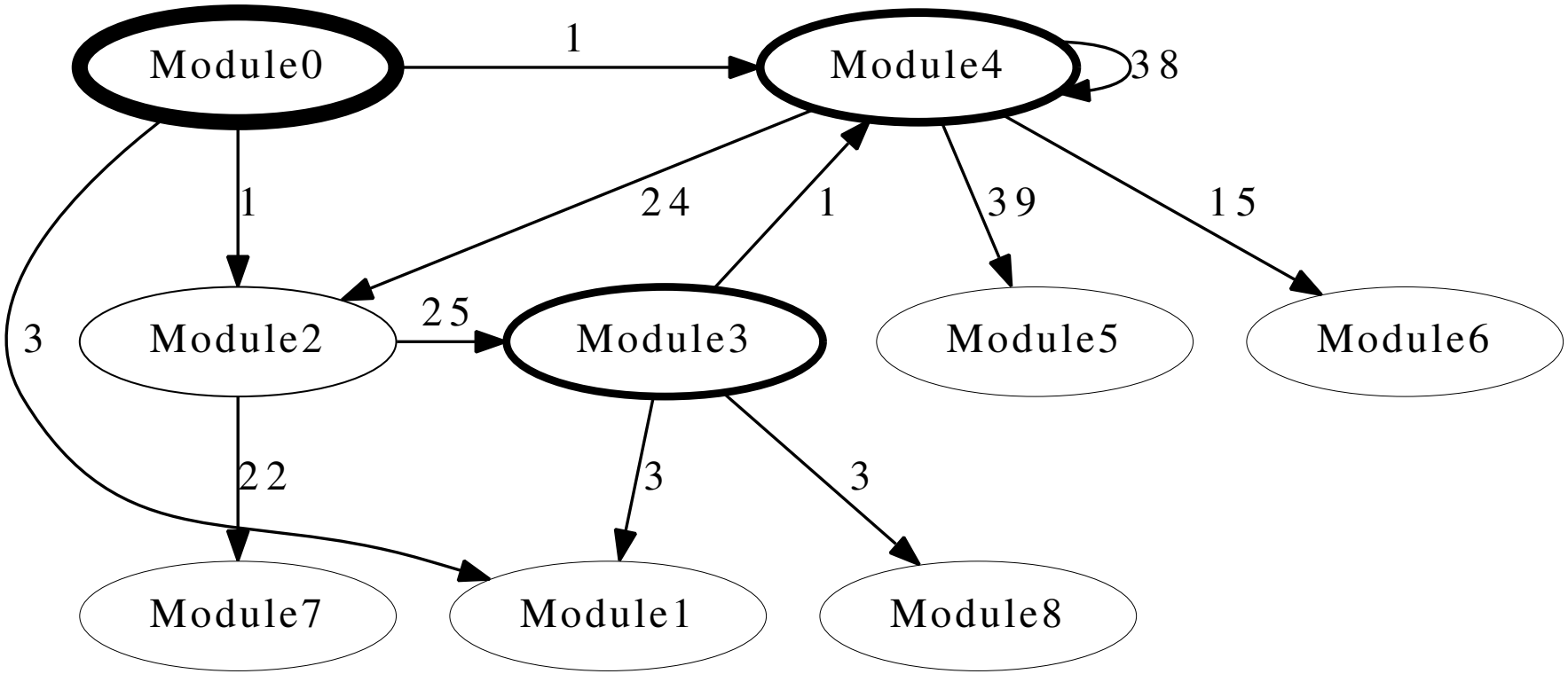
Tag-based Modules in GP

- Add mechanisms for tagging code
- Add mechanisms for retrieving/branching to code with closest matching tag
- As long as any code has been tagged, all branches go somewhere
- Number of tagged modules can grow incrementally over evolutionary time
- We use integer tags and unidirectional difference with wraparound for inexact matching

Push

- Stack-based postfix language with one stack per type
- Types include: integer, float, Boolean, name, code, exec, vector, matrix, quantum gate, [add more as needed]
- Missing argument? NOOP
- Trivial syntax:
  program → instruction | literal | (program*)

Evolved DSOAR Architecture (in one environment)



Prior Conclusions

- Execution stack manipulation supports the evolution of modular programs in many situations
- Tag-based modules are more effective in complex, non-uniform problem environments
- Tag-based modules may help to evolve complex software and solutions to unsolved problems in the future

Tags in S-Expressions

- A simple form:
  `(progn (tag-123 (+ a b)) tagged-034)`
- Challenges:
  - Endless recursion
  - Return values (of tagging and of tag references prior to tagging)
  - Arguments, particularly of multiple types
  - Multiple return values from modules

Endless Recursion

- Here we enforce an execution step limit and (generally) penalize programs that exceed it
- In Push results may be available even when execution is terminated for hitting the limit

Return Values

- Of tagging: we consider:
  - "silent": return default constant value without executing tagged code
  - "non-silent": evaluate the tagged code and return its value
- Of tag references prior to tagging: here we return a default constant value
- In Push it is trivial to provide *no* return value in all of these cases

Arguments

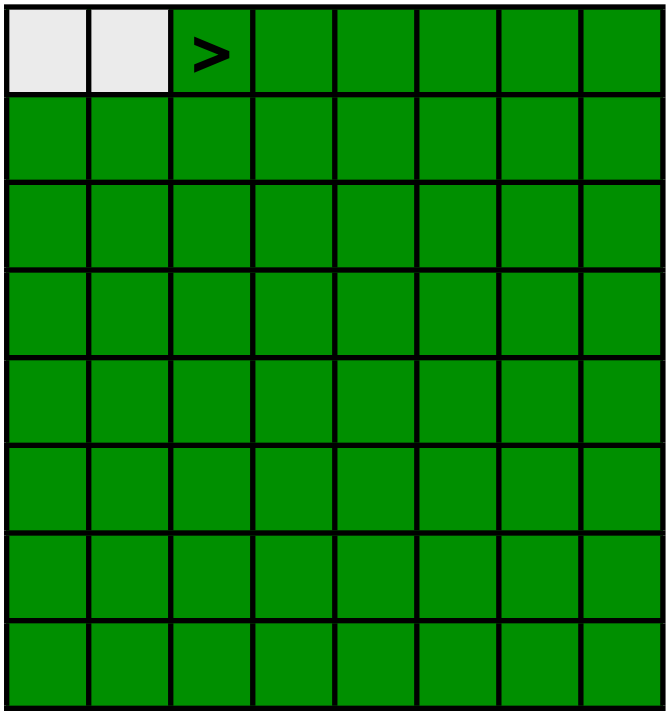
- Here we allow only 0-argument modules
- There's a tricky way in which the tag space itself can *conceivably* be used to pass arguments anyway (see paper)
- In Push any number of arguments may be provided without doing anything special

Multiple Return Values

- Basically we ignore this here, although again the tag space could *conceivably* be used to do something similar
- In Push it is trivial to return any number of values

Lawnmower Problem

- Used by Koza to demonstrate utility of ADFs for scaling GP up to larger problems



Lawnmower Results

Table 2: Results of genetic programming in several conditions related to tagging on the lawnmower problem.

Tags	Silent	No-op	Successes	MBF	Effort
No	—	No	63	0.45	282,000
No	—	Yes	53	0.62	357,000
Yes	No	—	97	0.13	30,000
Yes	Yes	—	65	0.57	144,000

Even 4-Parity Results

Table 4: Results of genetic programming in several conditions related to tagging on the even-4-parity problem.

Tags	Silent	No-op	Succ	MBF	Effort
No	—	No	58	0.56	234,000
No	—	Yes	36	1.02	495,000
Yes	No	—	22	1.44	950,000
Yes	Yes	—	19	1.61	1,044,000

First Conclusion

- The very simple form of "tags in trees" considered here is not always good!
- The remainder of our paper considers less simple approaches that may be better

Unbounded Recursion

- Penalties for hitting the execution limit may make tags too "dangerous"
- Note: we did not use penalties for the Lawnmower problem, since it works by side effects on the lawn state
- Various approaches are possible for *eliminating* rather than penalizing unbounded recursion

Arguments

- Pseudo-argument symbols like **`arg0`**, **`arg1`**, etc., which get replaced by values passed in as arguments to argument-taking tag-reference functions like **`tagged-1-arg`**, **`tagged-2-arg`**, etc.
  - Defaults for args in wrong contexts
  - Pre-specify maximum number of args
- Tag references with embedded argument reference tags

Program Size and Depth

- In the simple scheme presented here, tagging and tag references increase tree size and depth
- Depth and size limits therefore punish tag usage
- Tagging and tag-reference calls can be omitted from counts/limits in various ways
- Alternative syntax may involve less impact on size and depth

More Conclusions

- Trees are constraining!
- Tags may nonetheless be useful in tree-based GP
- Everyone should use Push ;-)
- Tag-based modules may also be a good fit to other forms of GP, e.g. Cartesian GP and Grammatical Evolution

Thanks

This material is based upon work supported by the National Science Foundation under Grant No. 1017817. Any opinions, findings, and conclusions or recommendations expressed in this publication are those of the authors and do not necessarily reflect the views of the National Science Foundation. Thanks also to Omri Bernstein, Emma Tosch, Kwaku Yeboah Antwi, and Rebecca S. Neimark for discussions related to this work, to Josiah Erikson for systems support, and to Hampshire College for support for the Hampshire College Institute for Computational Intelligence.