Partition Refinement for Classification

Sage Days 10 – Nancy, France

Robert L. Miller
University of Washington

October 11, 2008

Requirements

Suppose we have a collection of objects such that:

- Isomorphisms are finite permutations (of "points").
- There is a total ordering of the objects.
- A refinement function is implemented.
- A comparison function is implemented.
- An equivalence function is implemented.

Rewards

Then we obtain:

- Automorphism group computation
  (with base and strong generating set).
- Isomorphism calculation.
- Canonical labels (unique representatives for an iso-class).

Overview

The algorithms are based on B.D.McKay's partition method for graphs, which was generalized by J.S.Leon.

- Ordered partitions – the blocks are ordered, and within each block the elements are unordered.
- $(B_1, ..., B_k)$ is *finer* than $(B'_1, ..., B'_n)$ if $k \geq n$ and each B_i is a subset of some $B'_{i'}$.
- A *partition stack* is a sequence of ordered partitions, each one (strictly) finer than the previous.

Overview

- A partition stack whose finest partition consists of singletons (i.e. a discrete partition) defines an ordering of the points.
- A tree consisting of partition stacks is traversed, whose leaves correspond to different orderings of the point sets.
- Automorphisms of the object induce "automorphisms" of this tree.

The Refinement Function

- The hardest of the three functions to implement:
- INPUT: An ordered partition Π of the points of an object A.
- OUTPUT: An ordered partition Π' which is finer than Π, such that any automorphism of A that respects Π also respects Π'.
- Obviously one can define $\Pi' := \Pi$, but this is suggested only for the very patient.
- Examples can be found in `sage.groups.perm_gps.partn_ref.refinement_*`

The Comparison Function

- INPUT: Two objects A and B.
- OUTPUT: -1 if $A < B$, 0 if $A = B$, or 1 if $A > B$.
- Recall the requirement of a total ordering on objects, which is defined by this function: equality is needed to compute the automorphism group, order is needed for canonical labels.
- Examples can be found in `sage.groups.perm_gps.partn_ref.refinement_*`

The Equivalence Function

- INPUT: A ordered partition Π of the points of an object A.
- OUTPUT: True if the function can determine that all the discrete ordered partitions finer than Π are automorphic.
- In other words, each pair of such discrete partitions induces an automorphism of A.
- The function may return False even if the above holds.
- Examples can be found in `sage.groups.perm_gps.partn_ref.refinement_*`

Future Directions for Work

- It should be possible to require that isomorphisms are elements of a particular subgroup G of S_n.
- Requires Schreier-Sims algorithm (presently available via GAP and others), as well as several other BSGS algorithms.
- This will lead to several important permutation group computations, such as group intersection.
- Implement more types of objects!
- Substitution and functorial composition of species.

Canonical Augmentation

An algorithm for generating one representative from each isomorphism class. Also due to B.D.McKay. (Work in progress)

- Given any object A, there must be a sequence of *augmentations* leading to A:

  $A_0 \rightarrow A_1 \rightarrow \cdots \rightarrow A_n = A$.
- There must be a function o defined on objects, taking values in $\mathbb{Z}_{\geq 0}$, called the *order* , such that each augmentation increases the order by one. In the above sequence, $o(A_i) = i$.
- Call the set of chains as above the *search tree.*

Orbits on Children

- Define the set of *children* $C(X)$ of X to be the set of Y such that there is an augmentation $X \rightarrow Y$ and such that $o(X) + 1 = o(Y)$.
- Suppose we have just generated X. Then the automorphism group of X induces an action on $C(X)$. We want to select just one representative from each orbit under this action– call this transversal $C'(X)$.
- If we simply generate all the elements of $C'(X)$, we will eventually get repeats.

When is an augmentation canonical?

- Note that an isotype X will usually appear many times in the search tree, via different chains of augmentations.
- An *augmentation* is an ordered pair of labeled objects (X, Y), such that $Y \in C(X)$.
- An *isomorphism* of augmentations $(W, X) \cong (Y, Z)$ is a permutation γ such that $\gamma(W) = Y$ and $\gamma(X) = Z$.

When is an augmentation canonical?

- We want to define a canonical parent $M(X)$ for each object of positive order. Often this can be defined in terms of a canonical labeling map. For example, if we are augmenting graphs by adding one vertex at a time and edges connected to that vertex, we can define $M(X)$ as follows.
- If γ is the permutation taking X to its canonical label, simply delete $\gamma^{-1}(n)$ from X, where n is the highest vertex.
- In general, an augmentation (X, Y) is canonical if $(X, Y) \cong (M(Y), Y)$.
- Instead of the generated object being canonical, the object was generated *in a canonical way.*

Canonical Augmentation I

If we traverse only those nodes which are generated from minimal objects by a chain of canonical augmentations, then note if $X \cong Y$ are both generated, then we have

$$(P(X), X) \cong (M(X), X) \cong (M(Y), Y) \cong (P(Y), Y),$$

which in particular implies that $P(X) \cong P(Y)$. If isomorphs have already been rejected on the parents, we can conclude that $p(X) = p(Y) =: Z$. Since $(Z, X) \cong (Z, Y)$, there is a $\gamma \in \mathrm{Aut}(Z)$ such that $\gamma \cdot X = Y$. But we have already eliminated this possibility by computing $C'(Z)$.

Canonical Augmentation II

Algorithm 4

```
def traverse(node X):
    report X
    CX ← C(X)
    for each orbit of CX under Aut(X):
        select one representative Z
        if (Z,p(Z)) ≅ (Z,m(Z)):
            traverse(Z)
```

Already implemented in Sage

```
sage: for g in graphs(4):
...       print g.characteristic_polynomial()
x^4
x^4 + x^2
x^4
x^4 + x^2
x^4 + x^2
x^4 + 1
x^4 + x^2 + 1
x^4 + 1
x^4
x^4 + x^2
x^4 + 1
```

References

Technical:

- B. D. McKay, Practical graph isomorphism, *Congr. Numer.* 30 (1981), 45–87.
- J. S. Leon, Permutation Group Algorithms Based on Partitions, I: Theory and Algorithms, *J. Symbolic Computation* 12 (1991), 533–583.
- B. D. McKay, Isomorph-free exhaustive generation, *J. Algorithms* 26 (1998), 306–324.

Overview:

- P. Kaski, P. R. J. Östergård, *Classification Algorithms for Codes and Designs,* Springer-Verlag, Berlin, 2006.

Fin.