Gene Set Software Comparison

Commentary -- April 2

GeneSet, uniset, BaseSet
All
- Map identifiers
- Aware of GSEABase functionality; interoperation with org.*

unisets
- Reuse of S4Vectors
- Element / set
- Presentation quite confusing; terminology S4Vectors-specific
- Classed id and set types (GOSet, …) -- strongly typed (good, e.g., extra validity & content such as evidence and ontology) but additional complexity for user
    - Conversion between identifiers, like GSEABase::mapIdentifiers()
    - What about non-standard identifiers, e.g., arabidopsis?

GeneSet
- Explicit tidyverse grammar -- can be just as alienating for bioc-pros
- Tribble presentation
- gs_activate() %>% verb() versus %>% verb_element()
- `map <- mapIds(...); gs %>% map_genes(names(map), unname(map), “ENSEMBL”)`
    - ‘Curate id mapping yourself, then apply’

BaseSet
- Element / set
- Tidyverse approach
- Functionality: union(), intersection(), …
- Data-driven multi-map resolution

Additional features
- getURL() -- response to specific identifier type
- Coercion functions but not backward compatibility with GSEABase
- Track type and other (meta?) information about elements and sets
- Simplify gs_activate?

GSEABase

Status
- Mature
- https://bioconductor.org/packages/devel/GSEABase
- https://github.com/Bioconductor/GSEABase

Implementation
- S4

- Gene sets: GeneSet, GeneColorSet, GeneSetCollection
- Types to represent collections and identifiers, e.g., GOCollection, EntrezIdentifier

Functionality

- Set representation
- Identifier mapping -- `mapIdentifiers<-()`
- Provenance (e.g., creation date and source)
- I/O, e.g., gmt, slim, obo

Strengths

- Established
- Typed identifiers / collections

Limitations

- Inefficient -- one S4 object per gene set
- Non-'S4Vectors' paradigm
- Sets cannot be annotated with additional information

GeneSet

Status

- Development
- https://github.com/Kayla-Morrell/GeneSet

Implementation

- S4 wrapper coordinating S3 / dplyr tables

Functionality

- Gene set representation
- I/O

Strengths

- Familiar dplyr paradigm
- Scalable
- Flexible annotation of genes and sets

Limitations

- Untyped identifiers and sets

unisets

Status

- Maturing / Stable
- https://github.com/kevinrue/unisets
- Function reference: https://kevinrue.github.io/unisets/reference/index.html
- Example usage in a downstream package is demonstrated in “hancock” (https://github.com/kevinrue/hancock), along with the GeneSet package (above).

- Example usage in a data package is demonstrated in “xCellData” (https://github.com/kevinrue/xCellData)

Implementation

- S4 classes extending the “Hits” and “Vector” classes from the “S4Vectors” package
    - “Vector” subclasses are used to store elements and sets, and their metadata.
    - “Hits” subclasses are used to store relationships between elements (left node) and sets (right node), and their metadata.
- New types of relationship can be created by adding sub-classes of the “Hits” class, associated with additional constraints implemented as validity checks (see “FuzzyHits” and “GOHits”)
- New types of element and set identifiers can be created by adding sub-classes of the “IdVector” class.
    - The “IdVector” class is a new extension of the “Vector” class where “id” is a slot storing character identifiers, parallel to “elementMetadata”.

Functionality

- Gene set representation
    - Different validity checks distinguishing different types of relationships
- I/O
    - Support for GMT files (credits to Kayla for the initial code).
    - Integration with the “rtracklayer” package to automatically detect and dispatch on the file extension.
- Conversion between various classes of objects representing sets and membership
    - From and to ”list”
    - From and to “matrix” (i.e. incidence matrix)
- The package is intentionally restricted to a minimal set of essential functions:
    - Accessors: elements, sets, renaming, metadata
    - Dimensions: length, size of sets and elements
    - Subsetting: subset(), “[“
    - c(), duplicated(), unique(), union()
- Display in console
    - “DataFrame” format benefits of the “`showAsCell`” method to limit the amount of information previewed for large objects.

Strengths

- Typed relationships
    - “Hits”, “BaseSets”: untyped
        - Thought: I consider renaming “BaseSets” to “Sets”.
    - ”FuzzyHits”, ”FuzzySets”: requires “membership” metadata in the range [0,1]
    - ”GOHits”, “GOSets”: requires “evidence” and “ontology” metadata.
- Typed identifiers and sets

        - Future plan: this feature is intended to enable explicit or implicit conversion between types of identifiers using “org.*.db” packages. I intend to add methods to the “mapIds” generic for this functionality.
- Familiar “DataFrame” format for storing metadata.
- Scalable (> 3 million relationships example: https://kevinrue.github.io/unisets/articles/bioc-annotation.html#import-gene-ontology)

Limitations
- *De novo* construction/initialization of objects may be intimidating to novice users. This issue is currently addressed with extensive documentation of example usages, see https://kevinrue.github.io/unisets/articles/unisets.html#basesets-class for examples).
- **Only one type of element identifier and set identifier, respectively, can be used within a single object**. I would argue that this is not such a bad thing in itself. I think that different types of identifiers should live in different objects. If users wish to explicitly combine different types of identifiers, then one type of identifier should be converted to the other (with all the issues that come with mapping between identifiers). That said, I can see how users may wish to combine gene sets from different sources (e.g., GO, KEGG, MSigDB) and correct for multiple testing across the whole collection. But then again, that is a fairly advanced use case that should be handled by downstream developers rather than upstream data containers.
- R CMD check warnings likely due to roxygen documentation of mixed S4 and primitives (`Warning in formals(fun)`, `Warning in body(fun)` , see https://travis-ci.org/kevinrue/unisets)

BaseSet

Status
- Maturing
- https://github.com/llrs/BaseSet

Implementation
- A single S4 class (TidySet) with several S3 and S4 methods

Functionality
- Gene set representation
- Input from gmt
- Renaming of sets and elements (mapping identifiers)
- Conversion to several formats
- Using already listed elements to create new sets
- Add new sets to existing object
- Set operations: addition, union, intersection, subtraction, complement, creation, …

Strengths

- Familiar dplyr paradigm (select, filter, mutate, arrange, [activate])
- Scalable
- Size/cardinality of elements and sets
- Conversion between GSEABase classes and the new class
- Flexible annotation of genes and sets
- Allows using fuzzy sets (where the membership of a gene in a set is not fixed)

Limitations

- Support for data.frame derived object as internal tables (but not extensively tested on tibbles nor DataFrames)
- It doesn’t have any operations involving two objects, all implemented methods are for sets in the same object