

GlySpy and the Oligosaccharide Subtree Constraint Algorithm (OSCAR)

A Computational Approach to Sequencing Glycans

Anthony Lapadula
lapadula@cs.unh.edu

NH-BRIN Center for Structural Biology
at the University of New Hampshire
(UNH-CSB)

<http://glycome.unh.edu/>
<http://brin.unh.edu/>

Department of Chemistry
Parsons Hall

<http://www.unh.edu/chemistry/>

Department of Computer Science
Nesmith Hall

<http://www.cs.unh.edu/>

The University of New Hampshire
Durham, New Hampshire 03841

Table of Contents

1.	Abstract.....	4
2.	Introduction to Glycans.....	4
3.	Introduction to Mass Spectrometry.....	6
4.	MS ⁿ Fragment “Scars”.....	7
4.1.	Child Scars.....	7
4.2.	Parent Scars.....	7
5.	The Glycan Composition Finder.....	8
6.	Brief Overview of the Oligosaccharide Subtree Constraint Algorithm (OSCAR)....	9
6.1.	The Oligosaccharide Subtree Constraint Algorithm (OSCAR).....	10
7.	GlySpy Operational Overview.....	12
7.1.	Input.....	13
7.2.	Output.....	13
7.3.	Performance.....	14
8.	GlySpy Data Structures.....	14
8.1.	Fork.....	15
8.2.	Solution.....	15
8.3.	Mono.....	15
8.4.	Box.....	17
9.	GlySpy’s Core Algorithm: OSCAR.....	18
9.1.	Overview.....	18
9.2.	Boxes, Subtrees and Ions.....	18
9.3.	The Algorithm’s Main Phases.....	19
9.3.1.	Initial State.....	19
9.3.2.	Add MS Mass.....	19
9.3.3.	Add MS ⁿ Peak.....	19
9.3.3.1.	Composition Forking.....	19
9.3.3.2.	Selection Forking.....	20
9.3.3.3.	Draw Boxes.....	21
9.3.3.4.	Complementary Boxes.....	21
9.3.3.4.1.	Limitations.....	23
9.3.4.	Run Inference Rules.....	23

GlySpy and the Oligosaccharide Subtree Constraint Algorithm (OSCAR)

9.3.4.1.	Sample N-Linked Inference Rule	24
9.3.4.1.1.	Core Tree Restricts RootPossible	24
9.3.4.2.	Sample Box Inference Rules.....	24
9.3.4.2.1.	Single Box.....	24
9.3.4.2.2.	Complementary Boxes.....	25
9.3.4.3.	Sample Mono Inference Rules.....	25
9.3.4.3.1.	Apply Leaf	25
9.3.4.3.2.	Restrict NumChildrenPossible.....	26
9.3.5.	Calculate Scores.....	26
9.3.6.	Check Consistency.....	26
9.3.7.	Isomorph Pruning.....	27
9.3.8.	Summarize	28
9.3.8.1.	Generate Consistent Glycans	28
9.3.8.2.	Display Solution Statistics	29
10.	Future Work.....	29
10.1.	Possible Future Improvements.....	30
10.2.	The Goal: High-Throughput Glycan Sequencing	30
11.	Funding and Acknowledgements.....	31
12.	References.....	31

1. Abstract

We present the Oligosaccharide Subtree Constraint Algorithm (OSCAR), a novel algorithm for *de novo* sequencing of glycans using sequential mass spectrometry (MSⁿ) data. OSCAR accepts user-selected MSⁿ ion fragment paths and applies logical constraints to produce the full set of glycan structures that could yield the selected ions. We also describe GlySpy, the prototype tool that implements OSCAR, and provide examples of glycans that have been completely or partially sequenced by the tool.

2. Introduction to Glycans

Glycans are branching oligosaccharide (sugar) attachments found on certain proteins (glycoproteins) and fats (glycolipids). In *C. elegans*, the model organism studied at UNH-CSB, glycans are composed of a small number of monosaccharides:

- Glucose, galactose, and mannose. Collectively called hexoses, and abbreviated **H** in this paper.
- GlcNAc and GalNAc. Collectively called hexosamines, and abbreviated **N** when internal to the glycan, or **R** when located at the reducing end (the root of the glycan tree).
- Fucose, abbreviated **F**.

These monosaccharides are shown in Figure 1 are methylated versions of these four classes of monosaccharides. (In their natural, non-methylated state, each MeO group would be a simple OH group.)

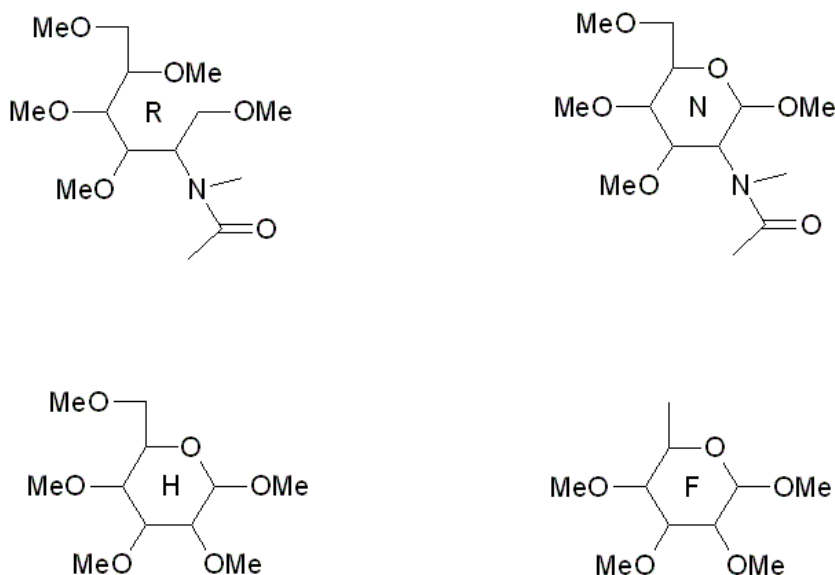


Figure 1: Methylated versions of the four classes of monosaccharides recognized by GlySpy.

Data presented in this paper were collected from glycans collected using two different procedures. In the first case, the monosaccharide class R is as shown in Figure 1, and we say that R has been *reduced*. In another case, the carbon ring of the R has not been

GlySpy and the Oligosaccharide Subtree Constraint Algorithm (OSCAR)

broken and R is identical to the N shown in this figure, and we say that R is *unreduced*. Even in this second case, we label the root monosaccharide as R and internal monosaccharides as N. GlySpy accepts a switch that differentiates between the two cases.

Each monosaccharide in the glycan may form a bond from its 1' carbon to the 2', 3', 4', or 6' position of its parent monosaccharide in the tree. See Figure 2 for the chemical diagram of one H 4-linked to another H. Figure 3 shows the same monosaccharide linkage, but in a shorthand notation that is used extensively throughout this paper. Here, each monosaccharide is labeled with both its class and a unique integer identifier, and possible linkage positions are indicated by numbers above the linkage arrow.



Figure 2: Example showing linkage of 1' carbon to 4' carbon. Here the left monosaccharide residue is considered the child of the parent monosaccharide on the right.

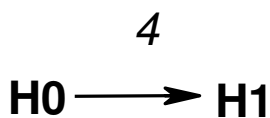


Figure 3: The same monosaccharide linkage as shown in Figure 2, but using a shorthand notation.

The N-linked subclass of glycans always contains a core tree with branching and linkage as shown in Figure 4. In N-linked glycans, these five monosaccharides are always present in exactly this configuration, and other monosaccharides are added to form more complicated N-linked glycans. Thus R is always the “root” of an N-linked glycan.

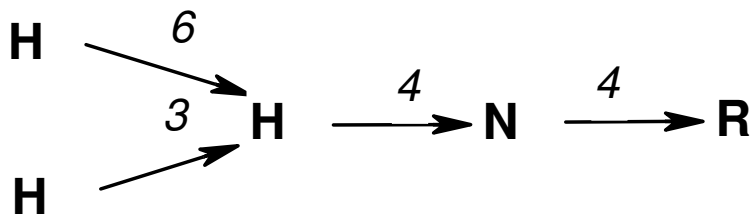


Figure 4: The monosaccharides and linkage always found in an N-linked glycan. Other monosaccharides may be added to this to form more complicated N-linked glycans.

3. Introduction to Mass Spectrometry

Mass Spectrometry, commonly called MS, is a procedure where a chemical sample is ionized and the mass to charge ratio (m/z) for each abundant molecule in the mixture is measured. The result is a spectrum indicating the relative abundance of the ions found.

Sequential Mass Spectrometry, or MS^n , can isolate a group of ions with a similar m/z (typically those ions that fall into a single peak on the MS spectrum), fragment those ions, and measure the m/z of the generated product ion fragments. The process can be repeated, with the product ions from one step being fragmented to reveal further internal detail. The first fragmentation of an MS ion yields an MS^2 spectrum; the fragmentation of an MS^2 product ion yields an MS^3 spectrum; and so on.

Figure 5 shows a sample MS^n spectrum obtained by successive fragmentation starting with a glycan with m/z 1187.6, and passing through product ions with m/z 's of 894.4 and 676.3. (This ion pathway can be seen in the figure's upper-left corner.) The masses and relative abundance of the fragments generated from the 676.3 ion are shown, including major peaks representing abundant ions at 431.1, 458.1, and 519.2.

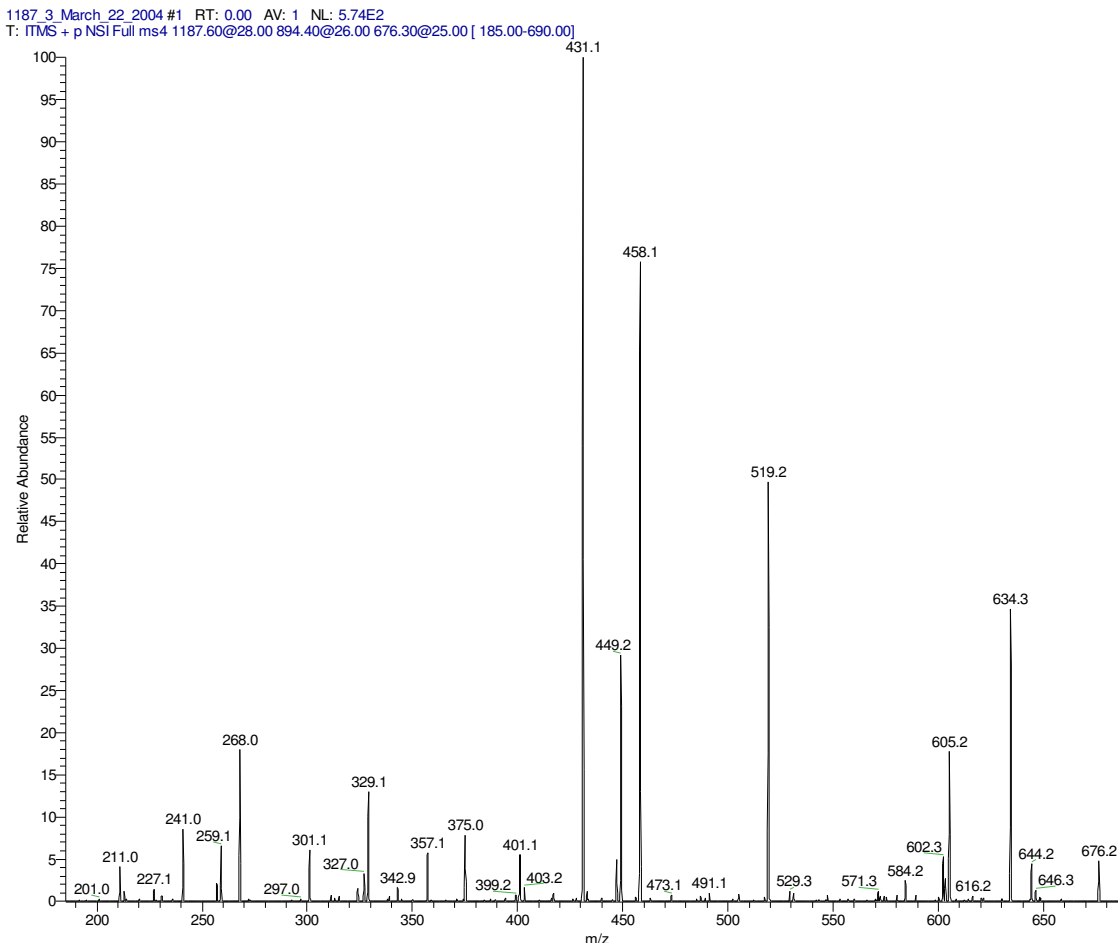


Figure 5: A sample MS^n spectrum.

4. MSⁿ Fragment “Scars”

Because the glycan samples examined by MSⁿ have been methylated, an MSⁿ fragment ion sometimes reveals how it was connected to its parent and child monosaccharides. We call these distinctive markings “scars.”

4.1. Child Scars

A monosaccharide fragment, when observed, will reveal the number of child monosaccharides which had been linked to it, but have been cleaved off. See Figure 6.

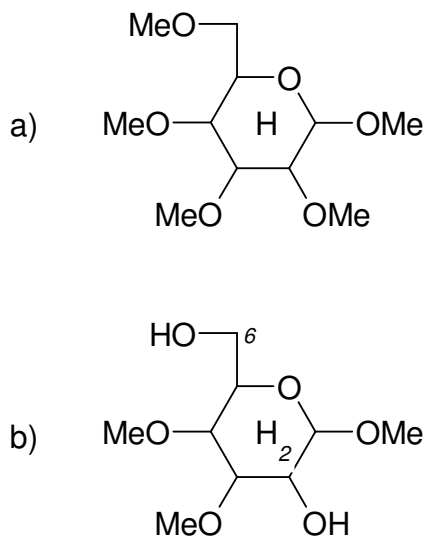


Figure 6: Child scars visible on a single H monosaccharide. The methylated H shown in (a) is shown with child scars at positions 2 and 6 in (b). The difference in mass between (a) and (b) reveals the number, but not position, of child scars.

However, these child scars provide no information as to linkage: A 2' child scar, for instance, will have the same mass differential as a 6' child scar. To infer linkage positions, we must find ions with indicative parent scars.

4.2. Parent Scars

Most parent scars provide no useful linkage information. However, linkage clues are provided by a subset of parent scars called “cross-ring cleavages.”

The less informative parent scars are:

- 1' –OH (abbreviated 1OH)
- 1' Double Bond (1DBL)

The crossing-ring cleavages are:

- 1-2' crossing-ring cleavage (abbreviated 1x2):

This fragment includes both the 2' and 3' carbons of the parent monosaccharide. However, experimentally, we do not observe crossing-ring cleavages for 3'-

connected sugars. (The reasons for this are currently unknown.) Therefore, a 1-2' crossing-ring cleavage is strongly indicative of a 2' linkage, and eliminates 3', 4', and 6' as possible linkages.

- 1-46' crossing-ring cleavage (1x46):

This cleavage includes the 4' and 6' carbons of the parent monosaccharide. For that reason, this cleavage cannot distinguish between 4'- and 6'-connected linkages, but does rule out 2' and 3' linkages.

- 1-6' crossing-ring cleavage (1x6):

This cleavage includes only the 6' carbon of the parent monosaccharide, and is therefore diagnostic of a 6' linkage.

Because parent scars are always located at the 1' carbon of the ion fragment, we sometimes refer to parent scars as 1' scars.

Note that this list of cross-ring cleavages is incomplete. Further MSⁿ experimental results appear to indicate the existence of other types of cross-ring ring cleavages, often found at much lower intensities than the common cleavages listed above. As these new cleavages become better understood and described, GlySpy will be extended to support them. As a practical matter, these missing cleavages are often revealed to the GlySpy user when an observed ion mass fails to map to a valid composition fragment. (This mapping is described in Section 5.) In this case, the user can often make progress by selecting other observed ions.

5. The Glycan Composition Finder

The Glycan Composition Finder is web-based tool developed by Hailong Zhang at the UNH-CSB. It provides the possible compositions of *C. elegans* N-linked glycans given a mass and an error tolerance for that mass. It also can limit its output to whole glycans (MS) or to glycan fragments (MSⁿ). (At the time of this writing, the Composition Finder is available at <http://brin.unh.edu/~hailong/new-mass>.)

For example, given a target MS mass of 1172 Da and an error tolerance of 1.0 Da, the Composition Finder returns the only possible unfragmented glycan composition with a mass between 1171 and 1173 Da: H₃NR, which has a monoisotopic mass of 1171.58.

The composition H₃NR means the precursor ion is exactly the N-linked core tree with no additional monosaccharides attached. When H₃NR is fragmented to yield an MS² spectrum, we see a major peak—that is, a peak corresponding to an abundant ion—with an m/z of 894.4. Given that mass and an error tolerance of 0.5, the Composition Finder would yield these five possible MSⁿ fragment compositions:

0 → [H₃/F₀/N₁/R₀] → 1DBL

1 → [H₃/F₀/N₀/R₁] → (none)

0 → [H₁/F₂/N₁/R₀] → 1x6

1 → [H₁/F₂/N₁/R₀] → 1x2

GlySpy and the Oligosaccharide Subtree Constraint Algorithm (OSCAR)



Our short-hand MSⁿ fragment notation has the format $C \rightarrow [H_n/F_n/N_n/R_n] \rightarrow S$ where:

C is the number (not type) of child scars,

H_n is the number of H residues in this fragment, etc., and

S is the parent scar type, represented as 1OH, 1DBL, 1x2, 1x46, 1x6 and (none). (Here, “(none)” means that the root node of the glycan is present in the fragment. Since the root node has no parent, it cannot have a parent scar.)

6. Brief Overview of the Oligosaccharide Subtree Constraint Algorithm (OSCAR)

GlySpy is a prototype implementation of the Oligosaccharide Subtree Constraint Algorithm (OSCAR). When given a small set of MSⁿ ions selected by a human expert, OSCAR has proposed the same glycan structure as deduced by the human expert. (The glycans currently produced by the algorithm have some reduced specificity in monosaccharide identification and linkage, but that is consistent with what a human MSⁿ expert can deduce without access to non-MSⁿ methodologies.)

The algorithm encodes *no* biologically-based restrictions on valid glycan structures: that is, the algorithm does not apply constraints that arise from known glycan biosynthetic pathways. This contrasts with the computer programs reported in (1) and (4), which both use known biological constraints to restrict the set of proposed glycans. Because our approach does not include these restrictions, we expect that our algorithm will be able to detect novel structures. Further, we believe our approach will more naturally allow for future support for sequencing glycans extracted from other organisms, where different biological restrictions will be present. Also, the computer programs presented in (1) and (4) use only tandem MS (MS/MS) spectra and MS spectra, respectively, whereas the algorithm presented here uses sequential mass spectra (MSⁿ).

The Oligosaccharide Subtree Constraint Algorithm is *de novo*. It does not attempt to match an unknown glycan sample against a database of known glycans (2) (4) or against known structural motifs (3), but instead examines only MSⁿ ions in order to propose structures.

Figure 7 shows a hypothetical glycan that will be used to briefly illustrate OSCAR. (Because OSCAR does not currently compute anomeric configuration, all diagrams have been simplified for clarity.) This figure shows an abstract representation of the tree structure of the hypothetical glycan. This tree notation is implemented by GlySpy to map chemical compounds to an easily manipulated format. Of course, trees are well-known data structures that are extremely amenable to computer analysis, and the logical properties of trees are used by OSCAR to sequence glycans.

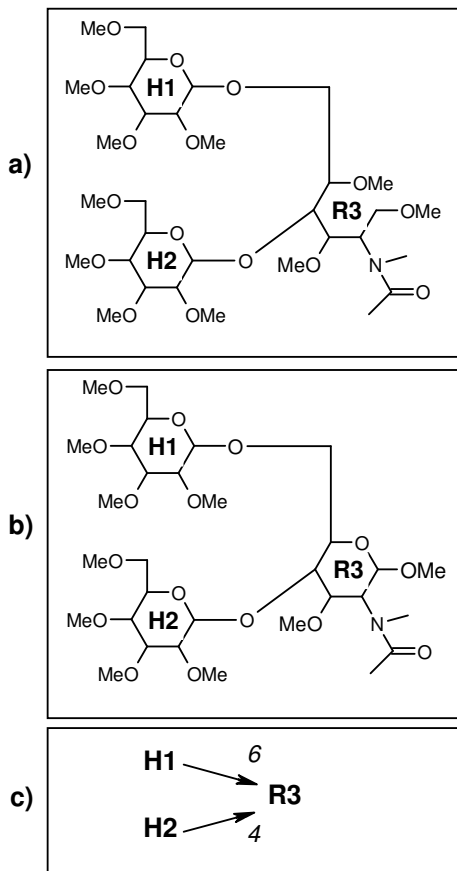


Figure 7: Simple glycan used to illustrate the Oligosaccharide Subtree Constraint Algorithm. Each monosaccharide residue has been labeled with its monomer class (H or R) and a number from 1 to 3 for unique identification. Glycans described in this paper have been prepared in two different ways, leading to difference in the reducing-end R monosaccharide residue. The reduced R is shown in (a) and the unreduced R is shown in (b); (a)'s m/z is approximately 16 Da greater than (b)'s. A compact notation for this glycan is shown in (c) and is used throughout this paper. Italicized numbers indicate linkage. Anomeric configuration is not shown.

6.1. The Oligosaccharide Subtree Constraint Algorithm (OSCAR)

The Oligosaccharide Subtree Constraint Algorithm takes as input a list of MS^n precursor/product ion pathways selected by the human operator. It constructs a representation of all possible glycans of the given mass and then, using the selected ion pathways, applies a series of inference rules to the proposed glycan set. These simple rules use the tree structure of glycans to enumerate possible structures by eliminating “impossible” structures—structures that would have to violate the logical properties of a tree in order to be consistent with the selected ion pathways. As roughly 50 inference rules are iteratively applied, the proposed glycan set is restricted to contain only those glycans that are consistent with the constraints imposed by the selected ions. The human operator can inspect the remaining proposed glycans to guide the selection of the next MS^n ion.

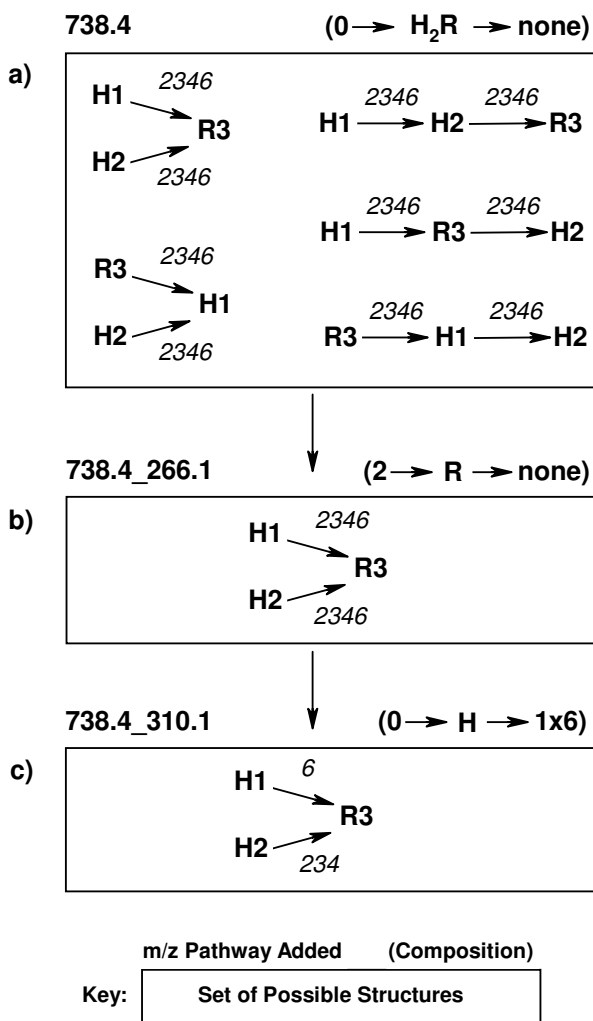


Figure 8: OSCAR’s processing steps for the hypothetical glycan shown in Figure 7. Step (a) shows initial m/z of 738.4 mapped to all possible structures for H₂R. Step (b) shows how the ion 2→R→none rules out most of the structures in step (a). (The composition notation “2→R→none” means that the R monosaccharide residue has two child scars and no parent scar.) Step (c) shows how a 1-6 cross-ring cleavage reveals linkage details. Residue H1 has been assigned the 6’ linkage, and so H2’s linkage has been narrowed to 2’, 3’, or 4’. Note that the compact visual representation used here can communicate partially-specified linkage details.

Figure 8(a) shows the initial state of the algorithm once the m/z of the unfragmented glycan is known to be 738.4. The Composition Finder database maps this m/z to the composition H₂R (with no child or parent scars), but at this point no branching or linkage details have been revealed. The possible structures at this point are shown in (a); all linkages are completely unspecified as indicated by the “2346” on each bond. (The user can optionally tell OSCAR that R must be the root of the glycan. That would eliminate many of the structures shown in (a), but is usually unnecessary, as the next selected ion will make clear.)

Figure 8(b) shows the effect of adding an MS² ion with m/z 266.1 (shown as the ion pathway 738.4_266.1). Composition Finder maps this to a single R with two child scars and no parent scars (2→R→none), indicating that this ion fragment has had two children

cleaved off. The corresponding box shows how most of the structures from step (a) have been ruled out. Now, only a single structure is possible, but linkage is still completely unspecified.

Figure 8(c) adds another MS² ion, this one with m/z 310.1. This ion maps to a composition of a single H with a 1-6 cross-ring cleavage (0→H→1x6). This means that one H must have been 6'-connected to R and, since the 6' position is now accounted for, the remaining H can only be 2'-, 3'-, or 4'-connected to R. Step (c)'s box indicates the final structure proposed by OSCAR given these input pathways.

This is a hypothetical example. The situation when analyzing real glycans is much more computationally involved. For example, each selected ion m/z often maps to many different possible compositions, and, further, the monosaccharides in a given fragment might map to any number of subsets of monosaccharides taken from the precursor ion. (In the given example, the H in the cross-ring ion of Figure 8(c) could have been assigned to either H1 or H2, but H1 was selected for clarity.) In these more complicated cases, OSCAR keeps track of *all combinations of compositions and monosaccharide mappings*, eliminating “impossible” combinations that lead to detectable logical contradictions. In effect, OSCAR runs many possible solutions in parallel and discards those that generate no valid structures.

GlySpy, which implements the Oligosaccharide Subtree Constraint Algorithm prototype, also includes the ability to narrow the search to N-linked glycans only, reducing the number of structures proposed.

7. GlySpy Operational Overview

GlySpy is currently a text-based application that runs on Microsoft Windows. It consists of nearly 20,000 lines of C++ and is being developed using Microsoft Visual Studio.

We briefly report the results of applying this algorithm to a novel N-linked glycan (with a composition of H₅F₂RN) reported by the UNH-CSB. Figure 9 shows the structure of this glycan as well as the structure reported by OSCAR.

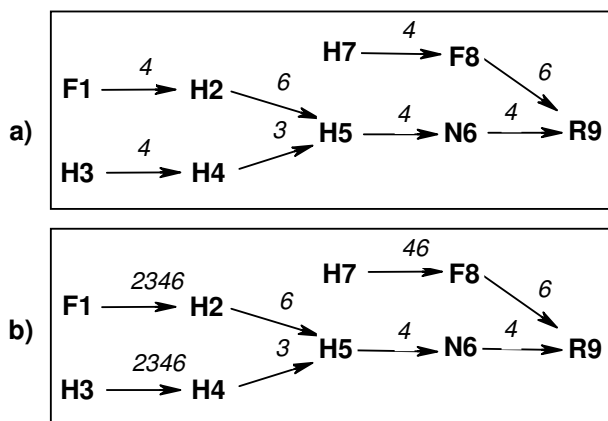


Figure 9: Structure of the N-linked glycan H₅F₂RN (m/z 1928) as (a) resolved by a human expert and (b) as proposed by OSCAR.

7.1. Input

GlySpy accepts a list of MS and MSⁿ masses of methylated *C. elegans* glycan fragments, plus a number of command-line switches. For example, “-nlinked” restricts GlySpy to glycans that contain the core N-linked subtree. For convenience, these switches can also be embedded in the input command stream.

Here is a sample input file. (The semicolon character introduces a comment, and GlySpy ignores the remainder of the line. These comments have been included to indicate what each ion maps to in the completed structure.)

```
-unreducedr ; Use unreduced mass for R
-nlinked ; Assume N-Linked
;
; Peaks Composition
; -----
AddPeak 1928.0 ; (H5 F2 N R)
AddPeak 1928.0_1272.6 ; 0→(H4 F N )→1
AddPeak 1928.0_1272.6_1027.6 ; 0→(H4 F )→1
AddPeak 1928.0_1272.6_850.5 ; 1→(H2 F )→1
AddPeak 1928.0_1272.6_850.5_210.9 ; 0→( F )→1
AddPeak 1928.0_1272.6_850.5_414.9 ; 0→(H F )→1
AddPeak 1928.0_678.1 ; 1→(H F R)→none
AddPeak 1928.0_678.1_432.9_329.0 ; 0→(H )→1x46
AddPeak 1928.0_678.1_474.9 ; 0→(H F )→1x6
AddPeak 1928.0_678.1_474.9_257.2 ; 1→( F )→1x6
AddPeak 1928.0_1272.6_850.5_474.9 ; 0→(H F )→1x6
Summarize
```

The AddPeak command accepts an argument of the format MS_MS2_MS3, where MS is the mass of the MS (unfragmented) ion, MS2 is the mass of a product ion fragmented from the MS ion, MS3 is the mass of a product ion fragmented from the MS2 ion, and so on. In this way, GlySpy knows the precursor/product relationship of all input ions.

7.2. Output

The Summarize command causes GlySpy to output the set of glycans that are consistent with the given input masses. In this case, the only glycan output is shown in Figure 9(b), which compares well to the expert-sequenced structure shown in Figure 9(a).

Notice that GlySpy was unable to fully resolve some parent/child linkages. For example, H7 is shown to connect to parent F8, but the linkage could only be restricted to 4' or 6'. This is because the single crossing-ring cleavage in the input data for this section of the tree was a 1-46' cleavage:

```
AddPeak 1928.0_678.1_432.9_329.0 ; 0→(H )→1x46
```

A human operator, utilizing other commands provided by GlySpy, could examine the raw MSⁿ data, if available, to attempt to resolve this linkage further. In this case, the operator would attempt to find an ion representing a 1-6' crossing-ring cleavage. Failing to find such an ion, the operator might assume that this was actually a 4' linkage, and resort to non-MS methods for confirmation.

Other output describes how each input ion is mapped to an oligosaccharide composition (including child and parent scars), and the location in the final structure where each

oligosaccharide fragment is located. The output for a few selected ions is given in Table 1 and illustrates how the 6' linkage from F8 to R9 was determined.

Ion pathway	Composition	Location in Glycan
1928.0_678.1	1→(H F R)→none	H7 F8 R8
1928.0_678.1_474.9	0→(H F)→1x6	H7 F8
1928.0_678.1_474.9_257.2	1→(F)→1x6	F8

Table 1: The final mapping for selected ions into the N-linked glycan of Figure 9. Each selected ion was mapped to many different compositions and many different proposed locations within the final glycan structure before arriving at these mappings.

7.3. Performance

For this glycan, the GlySpy executed in 0.4 seconds. (All reported execution times are from a 1.7 GHz Pentium 4 laptop PC with 512 MB of memory.)

8. GlySpy Data Structures

The algorithm's main data structures are Fork, Solution, Mono, and Box, which can be understood as follows:

- A Fork is *one interpretation* of the input MS^n ions, and can produce a set of glycans that are consistent with that interpretation. (Recall from Section 5 that each MS^n ion can map to multiple compositions and/or specific monosaccharides. A separate Fork is created for every combination of these mappings¹.)
- A Solution contains a set of Forks that covers *all possible interpretations* of the input MS^n ions. The union of the glycans produced by all contained Forks represents all possible glycan structures given the selected ions.
- A Mono is a single monosaccharide.
- A Box encloses a set of Monos that are known to belong to a single MS^n ion.

Roughly speaking, a Fork represents one “slice” of the entire problem, which is itself represented by the Solution. Each Fork contains a set of Monos, along with a series of Boxes that encloses subsets of those Monos. Each Box groups together Monos that are assumed, in this Fork, to form a connected subtree that is embedded in the final glycan structure.

Graphically, Figure 10 is a single Fork containing one Box (labeled 0 in the upper left corner) and nine Monos (labeled H0-H4, F5-F6, N7, and R8). This Fork represents the

¹ The term “Fork” comes from a quote attributed to Yogi Berra: “When you come to a fork in the road, take it.” This is exactly the search strategy used by the Logical Inference Algorithm: when multiple interpretations are possible, all of them are explored, one per Fork.

same H₅F₂RN glycan discussed in Section 6, although the individual monosaccharides have been renumbered here to allow a smoother presentation of the algorithm. A Solution would contain one or more of these Forks.

0	H0	H1	H2	H3	H4	F5	F6	N7	R8
---	----	----	----	----	----	----	----	----	----

Figure 10: Sample Box containing nine Monos representing the glycan H₅F₂RN.

8.1. Fork

A Fork is one interpretation of the input MSⁿ ions, and can produce a set of glycans that are consistent with that interpretation. As more ions are selected and added to the Fork, the Fork becomes more specific, capable of generating fewer and fewer consistent glycans. When a Fork is found to generate no consistent glycans, it is marked as “dead” and removed from the Solution.

A Fork contains:

- A set of Monos
- A list of Boxes that describe how the various MSⁿ inputs have been mapped to those Monos
- A set of Monos that represent the possible roots of the glycan (called `MSRootPossible`)
- A score that indicates how constrained the Fork is, with a *lower* score representing a more constrained and therefore more specific Fork.

8.2. Solution

A Solution is simply a set of Forks. The Solution as a whole can generate the entire set of consistent glycans. (Consistent glycans are those that are not refuted by any of the input MSⁿ data. For example, if the input MSⁿ data specify that the only F in a glycan is definitely connected to R, then the Solution will only generate glycans where F is connected to R.)

As new MSⁿ ion fragments are added to GlySpy, the Solution can grow or shrink. Forks are added when multiple search paths must be explored, and are removed when they are discovered to be internally inconsistent or redundant with another Fork in the Solution.

8.3. Mono

A Mono represents a monosaccharide. If the initial MS input maps to H₅F₂RN (as in this running example), then every Fork will contain nine Monos as shown in Figure 10.

A Mono contains:

- The type of Mono (H, F, N, or R)
- The index of the Mono (in this example, 0 to 8)

- A set representing the Monos that might *possibly* be this Mono's parent in the glycan (called `ParentPossible`)
- A set that represents the Mono that is *definitely* this Mono's parent (`ParentDefinite`)
- A set representing the Monos that might *possibly* be this Mono's children in the glycan (`ChildrenPossible`)
- A set that represents the Monos that are *definitely* this Mono's children (`ChildrenDefinite`)
- A set containing the number of possible children this Mono might have (`NumChildrenPossible`)
- A set containing the possible linkage positions (2, 3, 4, or 6) between this Mono and its parent (`Linkage`)

When created, the Mono's fields are initialized as follows:

- `ParentPossible`: All Monos except self
- `ParentDefinite`: Empty
- `ChildrenPossible`: All Monos except self
- `ChildrenDefinite`: Empty
- `NumChildrenPossible`: { 0, 1, 2, 3, 4 }
- `Linkage`: { 2, 3, 4, 6 }

Over time, OSCAR attempts to restrict `ParentPossible`, `ChildrenPossible`, `NumChildrenPossible`, and `Linkage` to fewer and fewer possible values. When the identity of a Mono's parent or child is learned with certainty, the `ParentDefinite` and `ChildrenDefinite` fields are set. For example, if a Mono has `NumChildrenPossible = { 1 }` (meaning that the given Mono must have exactly one child) and `ChildrenPossible = { H0 }`, then `ChildrenDefinite` can be set to { H0 }.

Implementation digression: Some inference rules, discussed later, can draw useful inferences knowing only the *possible* parents and children, whereas other inference rules apply only when the parent and children are *definitely* known. This is why both sets of information are computed. It could also be argued that having both parent and child information available is redundant, since one can be computed from the other. In our experience, some inference rules are much easier to write, understand, and debug given parent information, whereas others are more naturally expressed in terms of children. The minimal additional storage space pays for itself in ease of program maintenance and improved execution time.

8.4. Box

A Box represents a single ion and maps that ion to a set of Monos plus a number of child scars and a type of parent scar. The Monos within the Box must form a connected subtree embedded within the glycan.

Each Box contains:

- A set of Monos contained by this Box
- The number of child scars on the ion
- The type of the ion's 1' scar.
 - So if one MSⁿ input maps to $0 \rightarrow [H_3/F_0/N_0/R_0] \rightarrow 1x6$, there will be one corresponding Box that has zero child scars, contains three H Monos, and has a 6' crossing-ring cleavage.
- A unique ordinal number, with the first Box in a Fork always numbered zero (called `Index`)
- A link to a complementary Box, if any (more on complementary Boxes later)
- A set that represents the Monos that might *possibly* be the root of this Box (`RootPossible`)
- A set that represents the Mono that is *definitely* the root of this Box (`RootDefinite`)
- A set that represents the Mono that this Box's root might *possibly* connect to (`RootParentPossible`)
- A set that represents the Mono that this Box's root *definitely* connects to (`RootParentDefinite`)
- A set containing the possible linkage positions (2, 3, 4, or 6) between this Box and its parent (`Linkage`)

As with Mono, Box maintains `Possible` and `Definite` variations of both `Root` and `RootParent`. Again, this is because some inference rules can be applied knowing only which roots and root parents are possible, but other inference rules are valid only when the root and root parent are known with certainty.

When created, the Box's fields are initialized as follows:

- `RootPossible`: All Monos contained by this Box
- `RootDefinite`: Empty
- `RootParentPossible`: Any Mono not contained by this Box
- `RootParentDefinite`: Empty
- `Linkage`: { 2, 3, 4, 6 }

Over time, OSCAR attempts to restrict `RootPossible`, `RootParentPossible`, and `Linkage` to fewer and fewer possible values. When the identity of a `Box`'s root or root parent is learned with certainty, the `RootDefinite` and `RootParentDefinite` fields are set.

9. GlySpy's Core Algorithm: OSCAR

The Oligosaccharide Subtree Constraint Algorithm is the computational heart of GlySpy. It accepts MS^n input peaks, adds/removes Forks to/from the Solution, and applies the logical constraints that are used to determine which Forks in the Solution are still capable of producing consistent glycans.

9.1. Overview

The algorithm maintains a single Solution that contains a set of Forks, each of which can generate a number of consistent glycans. Each MS^n input peak selected by the user is mapped to a number of entries in the Composition Finder database; each of these compositions is then applied to the existing Forks. Existing Forks may need to be copied ("forked") before this application can be done.

Forks are tentative hypotheses that are discarded if discovered to contain internal contradictions. They are small data structures that are easy to copy and discard, and are that are created whenever multiple interpretations of an input ion must be examined.

Logical inference rules are then applied iteratively to each Fork. Forks that can generate no consistent glycans are marked as dead and removed from the Solution. Also, isomorphic (redundant) Forks are removed from the Solution, to ensure that the Solution does not grow to include an exponential number of isomorphic Forks.

It is important to note that OSCAR is *de novo* and utilizes no knowledge of the various classes of glycans found in *C. elegans*. Almost all of the constraints applied are straightforward corollaries that arise from the tree structure of glycans; a few remaining constraints are command-line options available to the operator (such as whether the glycan search should be constrained to N-linked glycans only).

9.2. Boxes, Subtrees and Ions

A glycan is a well-formed tree in the classic Computer Science sense: Every node (Mono) in the tree (glycan) has exactly one parent, except for the root of the tree, which has no parent, and cycles are not allowed. This means that the initial Box, which contains the entire glycan, also, by definition, represents a well-formed tree.

A product ion is also a well-formed subtree. You can visualize a product ion as a subtree that was embedded in the original glycan. The MS^n process used to fragment parent ions into product ions severs individual parent/child bonds in the glycan, but never forms new bonds.

Since each Box in a Fork represents either the initial glycan or a product ion of the glycan, it follows that **every Box in the Fork represents a well-formed subtree**. This is the key to how OSCAR both makes progress and discards inconsistent Forks.

For example, since a Box is a subtree and every subtree has exactly one root, we know that exactly one of the Monos in the Box must be the root of that subtree. If OSCAR at some point deduces that two different Monos must be the root of the same Box, or that no Monos can possibly be the root of the Box, then the algorithm has detected an inconsistency. The Fork containing that Box will be marked as inconsistent and removed from the Solution. Consistency checks like these are applied to every Box in every Fork, and in practice prove very efficient at pruning inconsistent Forks.

9.3. The Algorithm's Main Phases

9.3.1. Initial State

The Solution is set to empty (containing no Forks).

9.3.2. Add MS Mass

The first AddPeak command encountered (e.g., “AddPeak 1928.0” in the sample input shown above) defines the monoisotopic mass of the glycan of interest. GlySpy maps this mass, taking into account a user-defined error tolerance, to a number of entries in the Composition Finder database. One Fork is created for each of these MS compositions.

Typically, for *C. elegans* and the default MS error tolerance of 1.0 Da, only one composition is found and so the Solution contains a single Fork. In this case, the Composition Finder maps 1928.0 to a single valid MS composition: [H₅/F₂/N₁/R₁]. After this step, the Solution would contain a single Fork:

⁰	H0	H1	H2	H3	H4	F5	F6	N7	R8
--------------	----	----	----	----	----	----	----	----	----

Any product ion X generated from the precursor ion 1928.0 must select its Monos from those in 1928.0, any product ion Y generated in turn from product ion X must select its Monos from X, and so on. This will be represented graphically by drawing a new Box for each generated ion as this example continues.

9.3.3. Add MSⁿ Peak

Each additional time the AddPeak command is issued (e.g., “AddPeak 1928.0_1272.6_1027.6”), GlySpy parses the argument into a sequence of masses. The first mass always being the initial MS mass (here, 1928.0); subsequent masses represent product ions (precursor is 1928.0, product is 1272.6, and product of product is 1027.6).

Each mass is then added to every Fork, in a way that maintains the precursor/product relationships of all masses.

9.3.3.1. Composition Forking

When a given MSⁿ mass, in this case 1272.6, is added to a given Fork, the mass is first converted to a list of possible compositions. Using the default MSⁿ error tolerance of 0.5 Da, these six possible compositions are returned by the Composition Finder:

- C1) $0 \rightarrow [H_4/F_1/N_1/R_0] \rightarrow 1\text{DBL}$
- C2) $1 \rightarrow [H_4/F_1/N_0/R_1] \rightarrow (\text{none})$
- C3) $0 \rightarrow [H_2/F_3/N_1/R_0] \rightarrow 1 \times 6$
- C4) $1 \rightarrow [H_2/F_3/N_1/R_0] \rightarrow 1 \times 2$
- C5) $2 \rightarrow [H_2/F_3/N_1/R_0] \rightarrow 1 \times 46$
- C6) $2 \rightarrow [H_0/F_0/N_5/R_0] \rightarrow 1 \times 2$

GlySpy compares the possible product compositions to the composition of the precursor ion, $[H_5/F_2/N_1/R_1]$. Compositions C3, C4 and C5 all have more F Monos (3) than the precursor did (2), so they are not legal product compositions and are discarded. (The product's Monos must be a subset of the precursor's.) Likewise, composition C6 has more N Monos (5) than the precursor did (1), so C6 is also discarded.

The two remaining compositions, C1 and C2, are considered viable and will cause the existing parent Fork to be cloned ("forked") into at least two Forks, one Fork for C1 and one for C2. (If no composition was determined to be viable, the Fork would be marked as inconsistent and removed from the Solution.) Creating new forks to account for multiple possible compositions is called *composition forking*.

However, there is a second type of forking that must first be accounted for: selection forking.

9.3.3.2. Selection Forking

Selection forking is used when the Monos in a product ion might have come from different combinations of ions in the precursor ion. If, for example, a product ion maps to $0 \rightarrow [H_1/F_0/N_0/R_0] \rightarrow 1\text{DBL}$ and the precursor is $0 \rightarrow [H_4/F_0/N_0/R_0] \rightarrow 1\text{DBL}$, there are four ways the product ion can be subsetted out of the precursor. Since the subsetting is done at the time the product ion is added, GlySpy cannot know which of the four precursor Hs became that particular product H. To solve this problem, GlySpy merely forks the precursor Fork four ways, and uses each Fork to represent one of the four different possibilities: each Fork would draw its product Box around a different H.

In general, there will be one Fork created for each combination of legal product subset compositions taken out of the precursor composition. This is *selection forking*.

As GlySpy collects more information about the compound, it will become clear that some of these Forks are inconsistent. In this example, the product H must be a leaf (because the initial zero in " $0 \rightarrow [H_1/F_0/N_0/R_0] \rightarrow 1\text{DBL}$ " specifies that no child scars are present), but in one of the four Forks, that H may also be required to be the parent of some other Mono. This is a logical inconsistency, so the Fork will be marked as dead and removed from the Solution.

Returning to the running example of the previous section, where we have compositions C1 and C2, we see that those compositions each have multiple ways of being subsetted out of the precursor composition. In both cases, we need to select four H product Monos out of five precursor H Monos, and one F out of two. For each composition, then, we

need all combinations of five Hs taken four at a time, and two Fs taken one at a time. The formula to compute the number of combinations of n items taken r at a time is:

$${}_nC_r = n! / (r! (n-r)!)$$

Substituting n=5 and r=4, we see that each Fork must be copied ${}_5C_4 = 5$ times for the H selection forking. For the F selection forking, we have n=2 and r=1, yielding ${}_2C_1 = 2$ forks. So composition C1 and C2 each require a total of $5 * 2 = 10$ selection forks, and the Solution will contain a total of 20 Forks (10 selection forks * 2 composition forks). Although it seems that the number of Forks would grow extremely quickly, in practice most of these Forks will be discarded almost immediately as being either inconsistent (see Section 9.3.6) or isomorphic (Section 9.3.7).

9.3.3.3. Draw Boxes

For a given Fork, the new product ion causes one Box to be created. Given this precursor composition:

0	H0	H1	H2	H3	H4	F5	F6	N7	R8
---	----	----	----	----	----	----	----	----	----

Assume this Fork is one of the selection forks for composition C1 ($0 \rightarrow [H_4/F_1/N_1/R_0] \rightarrow 1DBL$), and that the specific Monos subsetted out of the precursor were H1, H2, H3, H4, F5, and N7. Rearranging the order of the Monos in the precursor is useful to group the subsetted Monos together:

0	H1	H2	H3	H4	F5	N7	H0	F6	R8
---	----	----	----	----	----	----	----	----	----

Now we can draw a new overlapping Box, labeled 1, to represent the specific Monos that make up the product ion:

	1								
0	H1	H2	H3	H4	F5	N7	H0	F6	R8

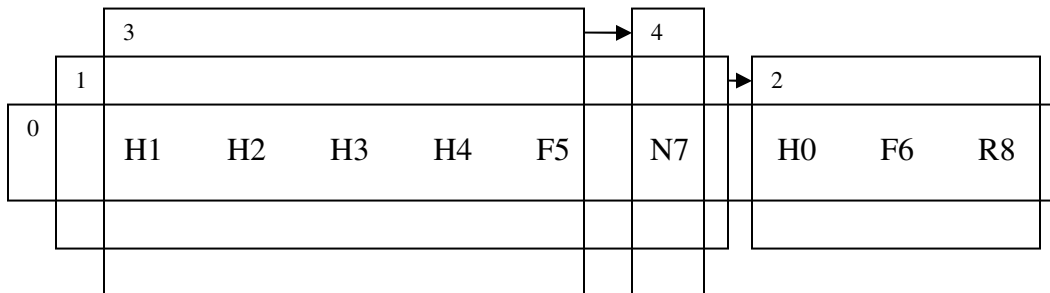
This helps visualize the fact that we have identified two separate subtrees in this glycan: the glycan in its entirety (represented by Box 0), and the product ion's embedded subtree (Box 1). However, there is more information to be gleaned from this particular product ion. We will be able to draw a Box that is complementary to Box 1.

9.3.3.4. Complementary Boxes

In Mass Spectrometry, we see that a precursor ion is fragmented into one or more product MS^n ions. In some cases, GlySpy is able to determine that a precursor ion was fragmented into exactly two product ions, the one that was observed in the data, and a second complementary ion that can be inferred.

Assume that the product ion represented by Box 1 was itself fragmented, yielding an observed “grand-product” ion that mapped to the composition $0 \rightarrow [H_0/F_0/N_1/R_0] \rightarrow 1\text{DBL}$. In this case we would be able to compute the complementary inferred “grand-product” ion as having the composition $[H_4/F_1/N_0/R_0]$ with 0 child scars and 1 parent scar. We would also know that the observed ion was the major ion and the inferred ion was minor.

Putting this all together, we can now draw Boxes for the two newly-created product ions, creating Boxes 3 and 4, subsetting their precursor Box (Box 1) in the process:



(Note that this particular ion does not represent the sample glycan shown in Figure 9, but is instead used only to illustrate the drawing of a nested Boxes for a multiple-generation product ions.)

The algorithm has begun to create the nested, overlapping Boxes that will be constrained by a series of inference rules. These rules will attempt to deduce facts about each Box and Mono in such a way as to infer the entire structure of the original glycan. This process is described in detail in Section 9.3.4.

9.3.3.4.1.

Limitations

There are many cases where a product ion cannot be proven to have a single inferred complement. (One example: When the observed product ion gains both a parent scar and a child scar relative to the precursor ion, implying that the product had a fragment cleaved from both above and below.) In these cases, a Box is drawn for the observed product, but no complementary Box is drawn.

There are also cases where the product ion can be shown to have a complementary ion, but it cannot be determined which ion is major and which is minor. (One example: A precursor ion has one parent scar and one child scar, and produces an observed product ion that also has one parent scar and one child scar, but that has one Mono fewer than the precursor. It is impossible to tell if the Mono that was lost was removed from above or below the observed ion.) In these cases, a complementary Box is drawn, and linked to the Box for the observed product ion, but both Boxes are marked as indeterminate instead of major or minor.

9.3.4.

Run Inference Rules

After all the Forks in the Solution have been updated to include the user-selected MS^n peak, a number of inference rules are run on each Fork. (Actually, the inference rules are run repeatedly on each Fork until the Fork’s score stabilizes, meaning that no further progress is being made. Scoring will be discussed later.)

GlySpy and the Oligosaccharide Subtree Constraint Algorithm (OSCAR)

The inference rules modify the various fields of each Mono and Box in a Fork, attempting to infer the set of glycans that are consistent with constraints imposed by each Mono and Box. Each inference rule is written in such a way that it can be applied at any time, in any order; this eliminates the problem of ordering the execution of the inference rules and opens the possibility of a future parallel or distributed implementation.

In all cases, inference rules attempt to use the subtrees defined by a Fork's Boxes to:

- Shrink the Possible sets (`ParentPossible` and `ChildrenPossible` in Mono; `RootPossible` and `RootParentPossible` in Box; `MSRootPossible` in Fork)
- Grow the Definite sets (`ParentDefinite` and `ChildrenDefinite` in Mono; `RootDefinite` and `RootParentDefinite` in Box)
- Shrink the `NumChildrenPossible` set in Mono
- Shrink the `Linkage` set in both Mono and Box

There are several different types of inference rules, and examples of each type are described in detail below. Currently, GlySpy implements over 50 inference rules.

9.3.4.1. Sample N-Linked Inference Rule

Some inference rules only apply when the user has given the “-nlinked” command-line switch, specifying that GlySpy should limit its search to valid N-linked glycans.

When -nlinked is specified, GlySpy, during the initialization of any new Fork, arbitrarily selects three H Monos, one N Mono, and one R Mono to represent the core tree, and then sets the `ParentDefinite` and `Linkage` fields of those Monos to specify the core tree.

9.3.4.1.1. Core Tree Restricts RootPossible

One N-linked inference rule uses the structure of the core tree to limit the `RootPossible` field of Boxes. For example, if a given Box contains the N of the core tree, then it is known that the three core H Monos cannot be the root of that Box, and those H Monos are excluded from `RootPossible`. Similar actions are taken if the Box contains the core R (removing N and all three H Monos from `RootPossible`) or the “parent” core H (removing its two H children).

9.3.4.2. Sample Box Inference Rules

A large number of inference rules operate primarily on one Box or a pair complementary Boxes. Here are examples of each.

9.3.4.2.1. Single Box

If a Box is found that has no parent scar and contains only a single Mono, then that Mono must be the root of the entire glycan. This Mono is added to the Fork's `MSRootPossible` set. Once the `MSRootPossible` set contains a single entry, other inference rules will propagate that information to the Boxes and Monos in the Fork. For example, if the glycan root is known to be Mono X, then other inference rules will:

- Remove X from the ChildrenPossible set of every Mono
- Add X to the RootDefinite set of every Box that contains X

This illustrates how the information gained from one simple inference rule can propagate throughout the Monos and Boxes in a Fork, and how the updated Monos and Boxes themselves will allow other inference rules to draw further conclusions, and so on.

Another powerful inference rule is one that detects leaves (Monos with no children). If a Box is found that contains a single Mono and has no child scars, then that Mono is known to be a leaf. The NumChildrenPossible set for that Mono is therefore restricted to { 0 }. (See Section 9.3.4.3.1 for examples of how other inference rules then propagate that information.)

9.3.4.2.2. *Complementary Boxes*

In our running example, we know that Box 1 and Box 2 are complements, that Box 1 is minor and Box 2 is major, and that both Boxes have Box 0 as their precursor. Here are some of the actions taken by inference rules when a major/minor complementary Box pair like this is examined:

- The RootPossible of the major Box is restricted to the RootPossible of the precursor Box (because the major Box by definition must contain the precursor's root)
- Conversely, the RootPossible of the precursor Box is restricted to the RootPossible of the major Box
- The RootParentPossible of the major Box excludes all Monos found in the minor Box (because the major Box is not rooted in the minor Box)
- Conversely, the RootParentPossible of the minor Box is restricted to the Monos in the major Box (because the root of the minor Box must connect to the major Box)

9.3.4.3. *Sample Mono Inference Rules*

Other inference rules examine Monos instead of Boxes. Here are some examples.

9.3.4.3.1. *Apply Leaf*

Once a Mono M is known to be a leaf (indicated by the Mono's NumChildrenPossible set containing only 0, with 1, 2, 3, and 4 having been previously eliminated), a number of restrictions can be applied:

- M's ChildrenPossible set is cleared to empty (because M cannot have children)
- For all Monos AM in the Fork, remove M from AM's ParentPossible set (because no Mono can have M as a parent)
- For all Boxes AB in the Fork, remove M as AB's RootParentPossible (because M cannot be the parent of any Box's root)

GlySpy and the Oligosaccharide Subtree Constraint Algorithm (OSCAR)

9.3.4.3.2.

Restrict NumChildrenPossible

If a Mono M has, for example, 2 Monos in its `ChildrenPossible` set, then we can exclude 3 and 4 from M 's `NumChildrenPossible`.

9.3.5. Calculate Scores

The full set of inference rules is applied iteratively to a given Fork until that Fork fails to make any progress. GlySpy uses the notion of scores to represent progress. A decreasing score implies that the Fork has been made more specific, and a score that does not change between applications of the set of inference rules means that the Fork has stalled and will not benefit from further applications of the inference rules.

Each Fork computes its score as follows (where $|X|$ represents the number of elements in set X):

- Add the score of each Mono in the Fork
- Add the score of each Box in the Fork
- Add `|MSRootPossible|`

The methods `Mono` and `Box` use to compute their scores are very similar to each other. Recall that the `Possible` sets shrink over time and the `Definite` sets grow, and that a decreasing score implies progress is being made. GlySpy therefore adds `|Possible|` and subtracts `|Definite|` when computing the score for a Mono or Box. Similar reasoning is applied to the other sets contained by Mono and Box: the shrinking sets `NumChildrenPossible` and `Linkage` are added.

Specifically, `Mono` computes its score as follows:

- Add `|ParentPossible|`, `|ChildrenPossible|`, `|NumChildrenPossible|`, `|Linkage|`
- Subtract `|ParentDefinite|`, `|ChildrenDefinite|`

`Box` computes its score as follows:

- Add `|RootPossible|`, `|RootParentPossible|`, `|Linkage|`
- Subtract `|RootDefinite|`, `|RootParentDefinite|`

Interestingly, a Fork's score does not seem to correlate closely with the number of consistent glycans it produces. The score is used for only two purposes: to terminate the iterative application of the inference rules, and to prune isomorphic Forks, described in Section 9.3.7.

9.3.6. Check Consistency

After the inference rules have been applied to each Fork, GlySpy examines each Fork for internal logical inconsistencies. If any inconsistencies are found, the Fork is marked as inconsistent and removed from the Solution.

Here are some of the conditions examined to ensure that a Fork is consistent:

- Each Mono must have at least one entry in its `NumChildrenPossible` set

GlySpy and the Oligosaccharide Subtree Constraint Algorithm (OSCAR)

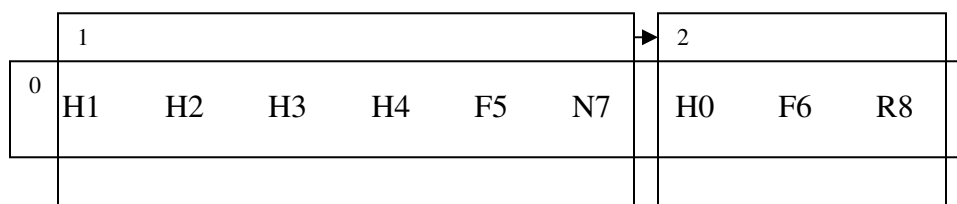
- Each Mono must have at least one Mono in its `ParentPossible` set, unless the Mono is possibly the MS root (in which case it would have no parent)
- Each Box must have at least one Mono in `RootPossible` (because every Box must have a Root)

Other consistency checks are performed as well, all of which verify that the constraints of a logical tree are not being violated by any Mono or Box in the Fork.

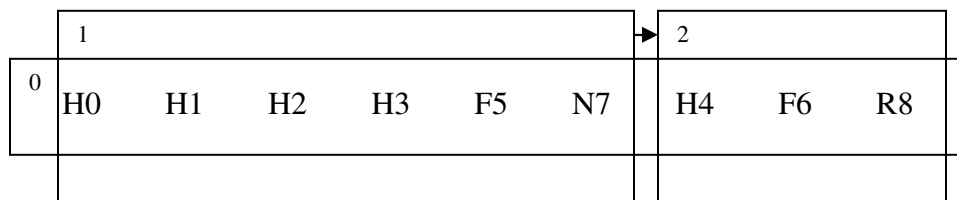
9.3.7. Isomorph Pruning

Selection forking can introduce Forks that are *guaranteed* to generate identical sets of consistent glycans. These redundant Forks are called isomorphs. Two Forks are isomorphic if (1) a one-to-one mapping exists from one Fork's Monos to the other Fork's Monos, and (2) a one-to-one mapping exists from one Fork's Boxes to the other Fork's Boxes. In other words, two Forks are isomorphic if one Fork's Monos and Boxes can simply be renumbered to yield an identical copy of the other Fork.

Returning to our running example, here is the Fork we presented as one of the outputs of the selection forking phase:



But another Fork created by the selection forking phase would have selected H0..H3 (instead of H1..H4) for Box 1, yielding:



After these two Forks are created, it becomes clear that they are isomorphs of each other. They are guaranteed to generate identical sets of consistent glycans, with only the numbering of the various H Monos being different.

A naïve implementation of isomorph pruning could compare every Fork against every other Fork, but that would have a run-time complexity of $O(n^2)$, a poor choice as n (the number of Forks in the Solution) becomes large.

Instead, GlySpy searches for isomorphic Forks after the inference rules have been applied to all Forks, which, as a side-effect, assigns a score to every Fork. Because the inference rules never base their actions on the index of the Monos or Boxes, isomorphic Forks generate the same score. GlySpy sorts the Forks by score, puts Forks with identical scores into buckets, and then only searches for isomorphic pairs within each bucket.

This implementation of isomorph pruning is critical in achieving GlySpy's fast execution times.

9.3.8. Summarize

When the "Summarize" command is issued, GlySpy generates and displays a set of consistent glycans for each remaining Fork. GlySpy then displays some Solution statistics.

9.3.8.1. Generate Consistent Glycans

This phase generates the output that the human operator is most interested in: the complete set of consistent glycans as restricted by the selected MSⁿ peaks.

To do this, GlySpy iterates over all remaining Forks in the Solution. For each Fork:

- If the MS root of the glycan is still unknown, the Fork is skipped because the number of generated glycans will likely be large, overwhelming the operator. (The operator is notified that the generated glycan list is incomplete and is told which Forks were skipped.)
- Otherwise, the `ParentPossible` set for each Mono is examined. A glycan is assembled with every Mono attached one of its possible parents. Then the glycan is subjected to further consistency checks (more below) and, if found to be valid, displayed. This process iterates until every possible Mono parent-child combination, and therefore every possible glycan, is examined.

Each proposed glycan is subjected to these consistency checks (and more) before it is considered to be consistent:

- Every Mono in the glycan should be reachable from the root of the proposed glycan
- The number of children any Mono has must be consistent with that Mono's `NumChildrenPossible`
- If the Mono has any definite (instead of merely possible) children, then that parent-child relationship must be present in the proposed glycan
- For every Box in the Fork:
 - The Monos in that Box must form a connected subtree embedded in the proposed glycan
 - The embedded subtree must have the same number of child scars (children of the glycan that fall outside the Box) as the Box
 - The root Mono's `Linkage` must agree with the Box's `Linkage`

For the input example shown in Section 7.1, the output is that of the sole consistent glycan shown in Figure 9(b), repeated here in Figure 11 for ease of reference.

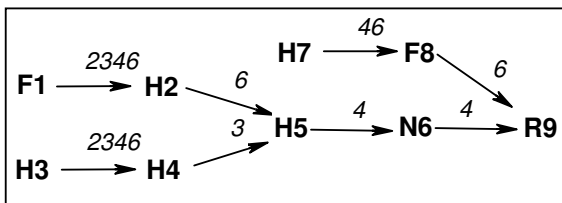


Figure 11: The same glycan as in **Figure 9(b)**, repeated for ease of reference. This is the sole glycan specified by GlySpy for the input shown in Section 7.1.

GlySpy's text output of glycan structures follows the time-honored Computer Science approach of indenting to indicate a child relationship. Additionally, the parenthesized numbers following each node represent the possible linkage of that node to its parent.

```

Generated 1 unique glycan (from 1 consistent, 1 total):
===== Begin Unique Glycans =====
R ( )
  F (6)
    H (46)
      N (4)
        H (4)
          H (3)
            H (2346)
              H (6)
                F (2346)
===== End Unique Glycans =====

```

9.3.8.2. Display Solution Statistics

After the set of consistent glycans is displayed, GlySpy also displays some statistics. For the input example shown in Section 7.1, some of the statistics displayed are:

```

Total forks: 158   Live: 1   Dead: 157 (Inconsistent: 154 Isomorph: 3)
From live forks --> Best: 28   Worst: 28   Average: 28.00
Histogram of live forks with score >= 0 (score*qty):
28
RunInferencesOnce called 272 times

```

This shows that over the course of the program's execution, a total of 158 Forks were created, but 157 of them were removed from the Solution, leaving a lone consistent Fork. (Of the 157 removed Forks, 154 were removed because they were inconsistent and 3 were removed because they were isomorphs. We might expect a larger number of isomorphs, but in this example, most of the isomorphic Forks are marked as inconsistent before they are discovered to be isomorphs. Recall that GlySpy runs the inference rules on Forks before searching for isomorphs, so Forks that are both isomorphic and inconsistent will be marked as inconsistent.)

We also see that the set of inference rules were run a total of 272 times. The `-time` command-line switch causes this additional output:

```
GlySpy: Total elapsed time: 0.39 secs
```

10. Future Work

GlySpy and OSCAR currently demonstrate a novel approach to semiautomatic glycan sequencing using MSⁿ data, but much more work can be done.

10.1. Possible Future Improvements

- A graphical user interface for both input (navigating MSⁿ spectra) and output (displaying sets of consistent glycans)
- Provide guidance to human operator for next MSⁿ peak to select (if the data already exist) or which MSⁿ data to generate (if the data are being collected) in order to minimize the amount of operator time required at the mass spectrometer to resolve a particular glycan
- Isotopic peak elimination
- Addition of new cross-ring cleavages as they are identified experimentally
- Monosaccharide identification using relative intensities fragments generated from dimers and trimers
- Automatic calibration of raw MSⁿ data (if the raw data suggest a peak at a mass of, say, 1171.4 but the nearest possible compound has a mass of 1171.58, then the raw data can be adjusted accordingly)
- Extend to support other monosaccharides, which could be the first step in sequencing glycans of species other than *C. elegans*
- Support O-linked glycans, first for *C. elegans*, then for other species. (Note: GlySpy makes no assumptions about which glycan structures are possible, so it can likely already solve O-linked glycans. However, without having had O-linked glycan data to test this, we cannot yet make this claim. Also, we expect that O-linked glycans may reveal the need for more inference rules, glycan consistency checks, etc.)
- Flag MSⁿ fragments that cannot be explained by the generated glycans. These fragments will likely hint at isobars that the human operator can then explore by selecting different MSⁿ peaks.
- Extensions to support other glycan derivatives (that is, glycans derivatized by some method other than permethylation)
- Continue to validate GlySpy at the UNH-CSB before distributing to other research labs

10.2. The Goal: High-Throughput Glycan Sequencing

The end goal of GlySpy is high-throughput, fully automated glycan sequencing, including support for isomeric mixtures.

We are investigating removing the need for an expert human operator by improving OSCAR to the point where it will be able to elucidate glycan structures given little or no human input. We are currently reviewing multiple strategies for this.

One of the most promising techniques identified so far involves the use of so-called Genetic Algorithms, an algorithmic approach to searching through a vast sea of possible structures. In this approach, GlySpy would essentially choose many sets of MSⁿ ions

GlySpy and the Oligosaccharide Subtree Constraint Algorithm (OSCAR)

more or less at random. Each of these sets would be evaluated for the quantity and specificity of the glycan structures produced. The ion sets that produced the best glycan structures would then be “mated,” with each parent set contributing half of its ions to each of its two offspring sets. This would give rise to a second generation of ion sets that would, on average, produce higher-quality glycan structures than the first generation. This process would continue until the proposed glycan structures could account for all of the major observed MSⁿ ions, and then the proposed glycans, along with their supporting MSⁿ ion selections, would be displayed. (Since different sets of ions might eventually come to represent different isomers in the glycan sample, we believe this technique may even be applied to isomeric mixtures, but at the current time, this is merely a conjecture.) GlySpy contains an initial prototype implementation of this approach, but more experimentation is required before we understand how fruitful this approach may become.

However, in the best-case scenario, GlySpy would elucidate glycan structure with no human guidance whatsoever. If successful, this would represent a strong contribution toward transforming glycan structural identification into a high-throughput technology.

11. Funding and Acknowledgements

Funding for this work was provided by NIH-BRIN grant RR16459.

Special thanks go to these members of the UNH-CSB: Hailong Zhang for the Composition Finder and his tireless explanation of the underlying biology and chemistry of glycan sequencing; Andy Hanneman and Suddham Singh for the explanations of Mass Spectrometry, access to their MSⁿ glycan data, and discussions of how they sequence glycans manually; and Dr. Vernon Reinhold for his help and support.

Also, thanks to these members of the University of New Hampshire Computer Science faculty for their ideas and feedback: Dr. Philip Hatcher, Dr. R. Daniel Bergeron and Dr. Ted Sparr. Dr. Hatcher’s work on a different *de novo* glycan sequencing algorithm was especially helpful.

12. References

- (1) Ethier, M.; Saba, J. A.; Ens, W.; Standing, K. G.; Perreault, H. *Rapid Commun. in Mass Spectrom.* **2002**, *16*, 1743-1754.
- (2) Cooper, C.A.; Joshi, H. J.; Harrison, M. J.; Wilkins, M. R.; Packer, N. H. *Nucleic Acids Research* **2003**, *Vol. 31, No. 1*, 511-513.
- (3) Tseng, K.; Hedrick, J. L.; Lebrilla, C. B. *Anal. Chem.* **1999**, *71*, 3747-3754.
- (4) Cooper, C. A.; Gasteiger, E.; Packer, N. H. *Proteomics* **2001**, *1*, 340-349.