

密级:



中国科学院大学
University of Chinese Academy of Sciences

硕士学位论文

N 糖结构数据库的构建算法研究及其应用

作者姓名: 张晓今

指导教师: 贺思敏 研究员

中国科学院计算技术研究所

学位类别: 工学硕士

学科专业: 计算机科学与技术

研究所: 中国科学院计算技术研究所

2017 年 05 月

Construction of N-glycan databases based on a linear
canonical representation of N-glycans

A Thesis Submitted to
The University of Chinese Academy of Sciences

in partial fulfillment of the requirement

for the degree of

Master of Science

in

Computer Science and Technology

by

Xiao-Jin Zhang

Thesis Supervisor: Professor Si-Min He

Institute of Computing Technology

Chinese Academy of Sciences

May, 2017

书脊

N 糖结构数据库的构建算法研究及其应用 张晓今 中国科学院大学

声 明

我声明本论文是我本人在导师指导下进行的研究工作及取得的研究成果。尽我所知，除了文中特别加以标注和致谢的地方外，本论文中不包含其他人已经发表或撰写过的研究成果。与我一同工作的同志对本研究所做的任何贡献均已在论文中作了明确的说明并表示了谢意。

作者签名: 日期:

论文版权使用授权书

本人授权中国科学院计算技术研究所可以保留并向国家有关部门或机构送交本论文的复印件和电子文档，允许本论文被查阅和借阅，可以将本论文的全部或部分内容编入有关数据库进行检索，可以采用影印、缩印或扫描等复制手段保存、汇编本论文。

(保密论文在解密后适用本授权书。)

作者签名: 导师签名: 日期:

摘要

N-糖基化作为一种普遍的蛋白质翻译后修饰，对生物功能有着重要的调控作用。随着质谱技术的发展，人们基于实验分析及文献报道构造了各种格式不一的糖库。然而，自然界中可能仍存在一些未经实验发现也未被记录在文献中的糖结构，即使对当前公开的所有糖结构库进行合并，得到的也是不全面的糖库。如果糖库不全，依赖于糖库的糖肽鉴定则无法搜索到正确的糖结构。考虑到当前糖库的局限性，本文旨在构造更加全面的理论 N-糖库，从而为 N-糖肽谱图的鉴定提供更好的保障。本文的创新点主要体现在如下三个方面：

第一，我们基于糖结构的邻接表存储格式枚举得到互不同构的 N-糖结构，并按照一定的规则去掉了一些不合理的糖结构，从而构建了一个更加全面合理且不包含同构 N-糖结构的枚举库。进一步，我们合并了 GlycomeDB 的 N-糖及人类血清中可能的高甘露、混合型及复杂型糖结构，得到了一个不包含同构 N-糖结构的合并库。通过分别基于合并库和枚举库进行糖肽谱图鉴定，我们发现枚举库有助于发现新的 N-糖结构，从而为我们鉴定到更多的糖肽提供了一种新的有效的途径。

第二，为了高效地对糖结构进行同构判定，我们设计了不同的糖结构哈希方法与编码方法，并尝试了不同的实现方法对糖结构库的去冗余过程进行优化。最终，我们在树结构的线性正则表示方法的基础上提出了一种新的编码方法，把分支的 N-糖结构映射为线性的字符串，不仅把 N-糖结构的同构判定问题转化为了字符串的冗余判定问题，而且提供了一种易于人工解码的 N-糖结构的线性正则表示方法，该表示方法极大地方便了 N-糖结构库之间的比较与合并。

第三，我们设计出了一种直接根据糖结构的线性正则字符串码构建理论 N-糖结构库的方法，该方法有效地提高了糖库构建的时空效率。同时，为了便于糖肽鉴定，我们构造了糖结构对应的理论谱图，即生成糖结构的 Y 离子。我们通过发现母结构的 Y 离子与子结构 Y 离子的组合之间的关系，设计了一种新的基于糖结构的线性正则表达式的子结构生成方法。与最初的基于糖结构的邻接表生成子结构的方法相比，该方法极大地提高了糖结构 Y 离子的生成效率。具体来说，生成 GlycomeDB 中的 N-糖与人类血清中三个最大通用结构的所有子结构合并得到的 7,884 个糖结构的 Y 离子，基于邻接表的子结构生成方法耗时 650 秒。我们通过算法方面的优化把这些糖结构的子结构生成时间缩短为 5 秒，进一步通过实现方面的优化把 5 秒加速到 2 秒。

关键词：质谱技术，N-糖结构线性正则表达式，N-糖结构库，糖肽鉴定

Abstract

Past years have witnessed protein glycosylation as one of the most abundant and critical post-translational modification. With the development of mass spectrometry, databases with various encoding schemes have been created based on experimental analyses and literature reports. However, it is highly probable that there exist some glycan structures which have been neither identified nor recorded. Thus, even the integration of all the publicly available glycan databases might not lead to a complete glycan database. Were the correct glycan structure not included in the database, it would be impossible for the glycopeptide identification dependent on glycan database to reach the correct glycan structure. Faced with this scenario, we aim at building a rational as well as comprehensive glycan database, thereby making it more beneficial for glycopeptide identification. The innovation points are embodied in the following three aspects:

Firstly, we enumerated all the theoretical N-glycans containing the specified number of monosaccharides, and the improbable glycan structures were further filtered out according to a series of research-based rules. Furthermore, the N-glycans from GlycomeDB and template-based N-glycans for human serum were merged into a nonredundant incorporated N-glycan database. We separately searched a set of glycopeptide spectra based on the theoretical N-glycan database and the incorporated N-glycan database, demonstrating the potential ability of the theoretical database for novel glycan discovery.

Secondly, we tried distinct hashing and coding methods with different implementation methods in order to identify the redundancy of the glycans efficiently. We proposed a novel linear canonical representation of the N-glycans based on that of the trees, the glycan structures are mapped into linearized strings, which not only succeed in simplifying the isomorphic detection for glycans as the problem of redundancy identification for strings, but also provide a linear storage format for the N-glycan structures which was easy to decode artificially and hence facilitating functions including comparison and incorporation among the glycan databases.

Thirdly, given the canonical representation of a glycan, it can be added with substrings to form new glycans with more nodes. In other words, we can enumerate glycans simply based on the canonical representation of the glycan structure, leading to a faster and more space-efficient enumeration approach. To facilitate the

glycopeptide identification, we generated the Y-ions of each glycan. We find the relationship between the Y-ions of each glycan and the composition of those of its subglycans, and developed a novel Y-ions generation method based on the canonical representation of the glycans. Compared with the original Y-ion generation method based on the adjacency-list of the glycans, this novel method significantly improved the generation efficiency. Specifically, given a total of 7,884 glycans, deriving from the integration of the Y-ions of the three largest plausible N-glycans in human serum and the N-glycans in GlycomeDB, the adjacency-list-based method took 650 seconds to generate the Y-ions of these glycans. The generation time was first reduced to 5 seconds through the algorithm optimization, and further reduced to 2 seconds through implementation optimization.

Keywords: mass spectrometry, linear canonical representation of N-glycan, N-glycan database, glycopeptide identification

目 录

摘要.....	I
Abstract.....	III
目录.....	V
图目录.....	VII
表目录.....	XI
第一章 引言	1
1.1 蛋白质的糖基化修饰	1
1.2 基于串联质谱技术的蛋白质鉴定	2
1.3 基于串联质谱技术的糖肽鉴定	3
1.4 公开的糖结构库及糖结构的表示方法	4
1.4.1 公开的糖结构库	5
1.4.2 糖结构的表示方法	7
1.5 本文的贡献	10
1.6 论文的组织	11
第二章 糖结构同构判定方法探究	13
2.1 基于哈希剪枝的糖同构判断方法	13
2.1.1 单糖的哈希	15
2.1.2 边的哈希	16
2.1.3 节点的哈希	16
2.1.4 自底向上哈希方法的效率测试	16
2.1.5 哈希方法总结	17
2.2 树同构判定方法调研	18
2.2.1 判断两棵树是否同构的 AHU74 算法	18
2.2.2 判断无根无序无标号树是否同构的 KS99 算法	20
2.2.3 基于树的序关系的字符串编码方法	22
2.2.4 广度优先字符串编码方法	23
2.2.5 深度优先字符串编码方法	25
2.2.6 字符串长度为 $2n$ 的树结构编码方法	26
2.3 糖结构同构判断的编码方法	27
2.3.1 双侧外包分层编码方法	27
2.3.2 单侧外包结合二层树编码方法	29
2.3.3 线性正则编码方法的总结	31
第三章 基于邻接表的理论 N-糖结构库构建	33
3.1 理论 N-糖结构库的枚举算法	33
3.2 败者树枚举更多节点数目的糖结构	35
3.3 合理糖结构筛选策略	36

3.4 糖结构库的扩展	38
3.5 基于邻接表生成糖结构的子结构的组成	40
3.6 糖结构的输出信息	42
第四章 基于糖结构线性正则表达式的 N-糖结构库构建	43
4.1 基于糖结构线性正则表达式的枚举方法	43
4.1.1 正则顺序的设计	44
4.1.2 枚举方法的设计	46
4.1.3 调整正则顺序方法的设计	52
4.1.4 计算糖结构中每个节点的分支数目的算法	54
4.1.5 相同兄弟节点子字符串的处理	56
4.1.6 基于糖结构的字符串码枚举糖结构示例	56
4.1.7 枚举算法总结	59
4.2 基于糖结构的字符串码生成糖的子结构的组成	60
4.2.1 糖结构的子结构的组成生成方法	61
4.2.2 生成糖结构的子结构加速策略探讨	62
4.3 枚举算法的效率评测	64
4.3.1 基于字符串码构建糖结构库的剪枝策略效率评测	64
4.3.2 基于字符串码生成糖结构的子结构的算法效率评测	65
4.3.3 基于字符串码构建糖结构库的算法效率评测	66
第五章 合并库的构建及枚举库的应用实验	67
5.1 糖结构合并库的构建	67
5.1.1 合并 GlycomeDB 的 N-糖库及人类血清通用结构库	67
5.1.2 合并库中糖结构的分布及来源	68
5.2 合并库与枚举库的包含关系分析	70
5.3 糖结构等价类的分析	71
5.4 枚举库的应用实验	73
第六章 基于线性正则表达式的搜索引擎设计	77
6.1 gMatch 的设计初衷	77
6.2 gMatch 的实现细节	79
6.3 gMatch 应用实验	82
第七章 结束语	85
7.1 全文总结	85
7.2 研究展望	87
参考文献	89
致 谢	i
作者简介	iii

图目录

图 1.1 不同生物体中的不同糖型	2
图 1.2 串联质谱流程示意图	3
图 1.3 糖肽鉴定流程	3
图 1.4 GlycomeDB 中 ID 值为 14282 的糖结构	8
图 1.5 GlycomeDB 中 ID 值为 14282 的糖结构的不同编码格式	8
图 1.6 图 1.4 所示糖结构的标号值	9
图 2.1 互为同构的两个糖结构	15
图 2.2 糖结构的哈希方法示意图	16
图 2.3 按照 AHU74 算法对两棵树进行同构判断的过程	19
图 2.4 树结构中每个节点的编号值及初始标号值	21
图 2.5 有根无序无标号树及其对应的字符串码	21
图 2.6 有根无序无标号树及其对应的字符串码	23
图 2.7 有根树及其对应的字符串码	24
图 2.8 A\$BB\$C\$DC#可能对应的两种有根有序树	24
图 2.9 有根树及其对应的字符串码	25
图 2.10 按照 AHU74 算法构造 DFCF 的过程	26
图 2.11 有根无序有标号树及其对应的字符串码	27
图 2.12 构造糖结构对应的字符串码的过程	28
图 2.13 给定根节点对应的字符串，解析出该字符串对应的糖结构的过程示例	29
图 2.14 包含连接位点信息的糖结构	29
图 2.15 糖结构包含的两层糖结构及其对应的标号值	30
图 2.16 单侧外包结合二层树的编码过程	31
图 3.1 由一个节点数目为 5 的糖结构枚举出节点数目为 6 的糖结构	34
图 3.2 GP Finder 的过滤规则	37
图 3.3 糖结构扩展示例	39
图 3.4 糖结构的部分子结构生成示例	41
图 4.1 正则顺序为字典序的糖结构枚举示例	44
图 4.2 希望设计一种正则顺序能很好地区分开叶子节点和非叶子节点	45
图 4.3 新的正则顺序下糖结构枚举示例	45
图 4.4 用来说明算法设计的糖结构	47
图 4.5 将图 4.4 中所示糖结构的节点 6 的左边插入单糖类型为 Hex 的节点得到的糖结构	47
图 4.6 兄弟节点的最右端插入新的节点	48
图 4.7 将四种单糖分别加入到图 4.6 所示糖结构的最左端叶子节点的下一层得到的糖结构	48
图 4.8 图 4.7 中所示的糖结构调整好正则顺序得到的糖结构	49
图 4.9 构造 4.8 所示糖结构的另一种方法	49
图 4.10 糖结构的枚举模板	50

图 4.11 在五糖核心的节点 1 上添加新的节点得到的糖结构	50
图 4.12 糖结构	51
图 4.13 添加不同类型的单糖得到的糖结构	51
图 4.14 扩展得到不符合正则顺序的糖结构	51
图 4.15 如果只添加符合正则条件的节点则无法生成该糖结构	52
图 4.16 糖结构的扩展示意图	52
图 4.17 正则顺序为字典序下的扩展示意图	53
图 4.18 糖结构没有五糖核心限定下的扩展示意图	53
图 4.19 正则顺序的调整示意图	54
图 4.20 糖结构及其对应的字符串和分支字符串	55
图 4.21 加入新节点后的糖结构及其对应的字符串和分支字符串	55
图 4.22 调整正则顺序之后的糖结构及其对应的字符串和分支字符串	55
图 4.23 相邻兄弟节点对应的字符串码相同的情形	56
图 4.24 以该糖结构为基础枚举新的糖结构	57
图 4.25 在 4.24 所示的糖结构的节点 1 的孩子节点左边添加新的节点得到的糖结构	57
图 4.26 在 4.24 所示的糖结构的节点 2 的孩子节点左边添加新的节点得到的糖结构	58
图 4.27 在图 4.24 所示的糖结构的节点 4 的最右端孩子节点的右边添加新的节点得到的糖结构	58
图 4.28 在图 4.24 所示的糖结构的节点 4 的最右端孩子节点的下一层添加新的节点得到的糖结构	59
图 4.29 用来说明垂直冗余的糖结构	59
图 4.30 糖结构扩展方法一	60
图 4.31 糖结构扩展方法二	60
图 4.32 基于糖结构的字符串码生成子结构的方法示例	61
图 4.33 节点 5 和节点 6 的子结构集合完全相同	63
图 4.34 糖结构示例。节点 3 与节点 4 对应的子结构集合不相同，而节点 4 和节点 5 对应的子结构集合完全相同	63
图 5.1 人类血清中三个最大可能的 N-糖	68
图 5.2 GlycomeDB 的 N-糖库与人类血清通用结构库的交并差	69
图 5.3 由五种单糖构成的合并库中糖结构来源的分布图	69
图 5.4 包含在合并库(6~10 个节点)却不在枚举库(6~10 个节点)中的糖结构	70
图 5.5 合并库中不符合 GP Finder 规则的糖结构的分布图	71
图 5.6 理论 Y 离子完全相同的糖结构示例	71
图 5.7 枚举库中节点数目为 11 的糖结构的等价类分布	72
图 5.8 合并库中 7,682 个糖结构的等价类分布图	73
图 5.9 糖结构的节点数目及糖库规模	74
图 5.10 pGlyco 2.0 分别基于合并库和枚举库鉴定到的谱图数目的韦恩图 ..	75
图 5.11 pGlyco 2.0 基于枚举库比基于合并库多鉴定到的一张谱图	76
图 5.12 枚举库鉴定到的新的糖链对应的可能的结构	76

图 6.1 谱图 1 的匹配情况	78
图 6.2 谱图 2 的匹配情况	79
图 6.3 判断一张谱图是否为糖肽谱图的特征峰	80
图 6.4 五糖核心对应的 Y 离子	81
图 6.5 糖结构理论谱峰表	82
图 6.6 gMatch 召回的谱图 M-Brain-Z-T-1612-01.5945.5945.2.1.dta 对应的鉴定结果	83
图 6.7 gMatch 召回的谱图 M-Brain-Z-T-1612-01.4441.4441.3.0.dta 对应的鉴定结果	83
图 6.8 gMatch 召回的谱图 M-Brain-Z-T-1612-01.6627.6627.2.0.dta 对应的鉴定结果	84

表目录

表 1.1 糖库及其网址	5
表 2.1 单糖的哈希值	15
表 2.2 自底向上哈希与遍历方法的效率对比	17
表 2.3 自底向上哈希与质量哈希方法的效率对比	17
表 3.1 单糖的分支数目限制	37
表 3.2 不合理的糖结构剪枝前的枚举结果	38
表 3.3 不合理的糖结构剪枝后的枚举结果	38
表 4.1 测试环境	64
表 4.2 剪枝策略的时间效率测试	65
表 4.3 剪枝策略的空间效率测试	65
表 4.4 基于字符串码构建糖结构库的算法效率评测	66
表 6.1 实验数据的搜索参数	78
表 6.2 pGlyco 2.0 软件基于合并库对鼠数据的鉴定结果	78
表 6.3 pGlyco 给出的鉴定结果及相应指标值	84

第一章 引言

糖基化在蛋白质折叠、循环系统的稳定性等方面扮演着重要的角色[1, 2]。不少证据表明很多疾病如癌症与糖基化异常非常相关[3-6]。因此，糖肽的鉴定有着极其重要的意义。下面我们分别对糖基化修饰、基于质谱技术的蛋白质与糖蛋白鉴定、当前主要的糖结构库以及糖结构在数据库中的存储格式这些背景知识进行介绍，指出糖结构库及糖结构存储格式存在的问题，并介绍了本文的主要贡献以及行文框架。

1.1 蛋白质的糖基化修饰

糖基化是蛋白质的一种非常普遍的翻译后修饰，据估计，大约有 50% 的哺乳动物的蛋白质发生了糖基化[7]。O-糖基化和 N-糖基化为生物体内两种最常见的糖基化修饰。O-糖修饰能且只能发生在丝氨酸 (serines) 或苏氨酸 (theonines) 上，并与这两种残基的羟基团相连。[8]指出哺乳动物的蛋白糖基化中大约有 90% 为 N-糖基化。N-糖结构的内侧通常会包含一个五糖核心（如图 1.1 所示），与蛋白质中的天冬酰胺序列中的酰胺基相连，通常的模式为 Asn-X-Ser/Thr，其中，Asn 表示天冬酰胺，X 表示除了脯氨酸的任一氨基酸 Ser 表示丝氨酸，Thr 表示苏氨酸[9-11]。还有少量糖基化修饰发生在 Asn-X-Cys 上，其中，Cys 表示半胱氨酸。N-糖结构可以分为高甘露糖型、混合型和复杂型糖结构。不同的糖结构可以连在同一个位点上（微观不均一性），而同一个蛋白质上也可有不同的位点连接糖结构（显著不均一性）[12]。糖肽位点分析表明 N-X-T 比 N-X-S 更容易成为 N-糖基化位点[13, 14]。图 1.1 中展示了构成 N-糖的单糖类型，其中 Xyl 通常只在植物体中存在，Mannose 和 Galactose 一般不能被质谱直接区分开来，统称为 Hex。因此，在不考虑植物糖蛋白鉴定的情况下，N-糖可以看做是由 Hex、HexNAc、NeuAc、NeuGc、dHex 这五种单糖构成，通常用一个五维向量表示这五种单糖的数目，称之为糖组成。比如，五糖核心的糖组成为 [3, 2, 0, 0, 0]。从图 1.1 中还可以看出人类的 N-糖中不包含 NeuGc。[15]指出我们可以从三个方面来区分 N-糖和 O-糖。（1）核心结构的类型和形状；（2）任一链的性质和长度；（3）任一终端抗原 (terminal epitopes)，比如，唾液酸修饰、A 抗原或 B 抗原。

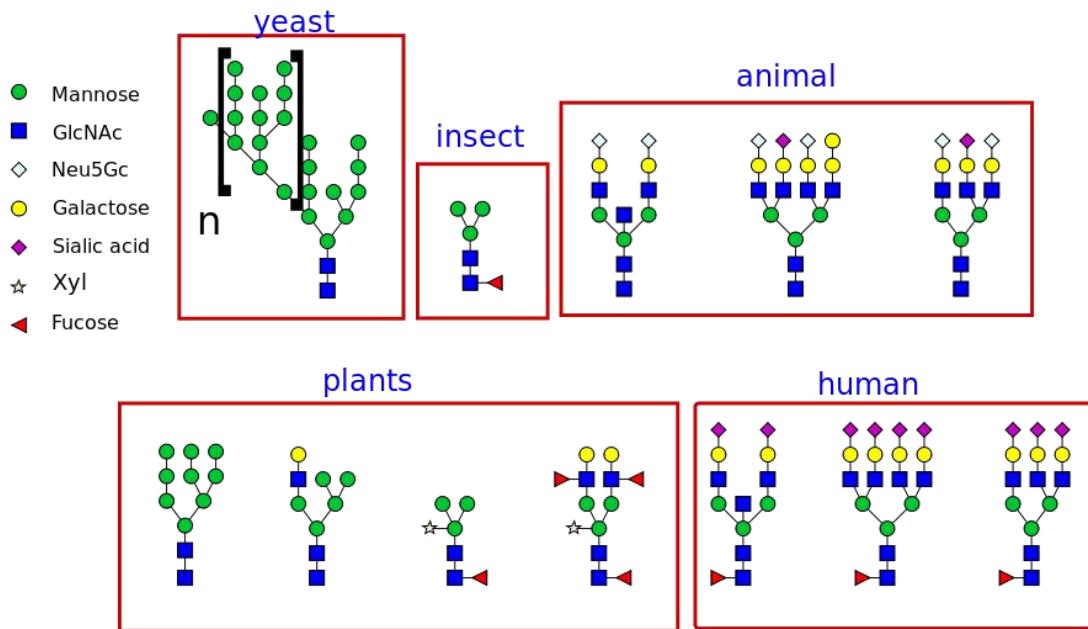


图 1.1 不同生物体中的不同糖型 (参考自维基百科)

1.2 基于串联质谱技术的蛋白质鉴定

利用串联质谱技术用来鉴定蛋白质的方法通常分为三类，分别为数据库搜索、从头测序和肽序列标签查询。其中，基于数据库搜索的串联质谱技术是当前最常用的蛋白质鉴定方法。串联质谱技术将蛋白质混合样品在一定的蛋白酶作用下酶解成更短的肽段序列，用液相色谱或者其它分离方法对肽段进行分离，分析检测肽段离子，并将肽段进行二次碎裂生成一系列碎片离子，碎片离子的质荷比(m/z 值)及强度值记录在 MS/MS 谱图中，得到实验串联质谱[16]，如图 1.2 所示。依赖于数据库搜索的方法将数据库中的蛋白质按照一定的酶切方式模拟酶解成多个肽段序列，对于给定的实验肽段，比较该实验肽段的质量与数据库中模拟生成的理论肽段，如果实验肽段与理论肽段的质量差在设置的误差范围内，则将这些满足质量误差的理论肽段进行进一步的碎裂，生成理论串联质谱，并与实验串联质谱进行匹配，从而实现对肽段序列的鉴定。

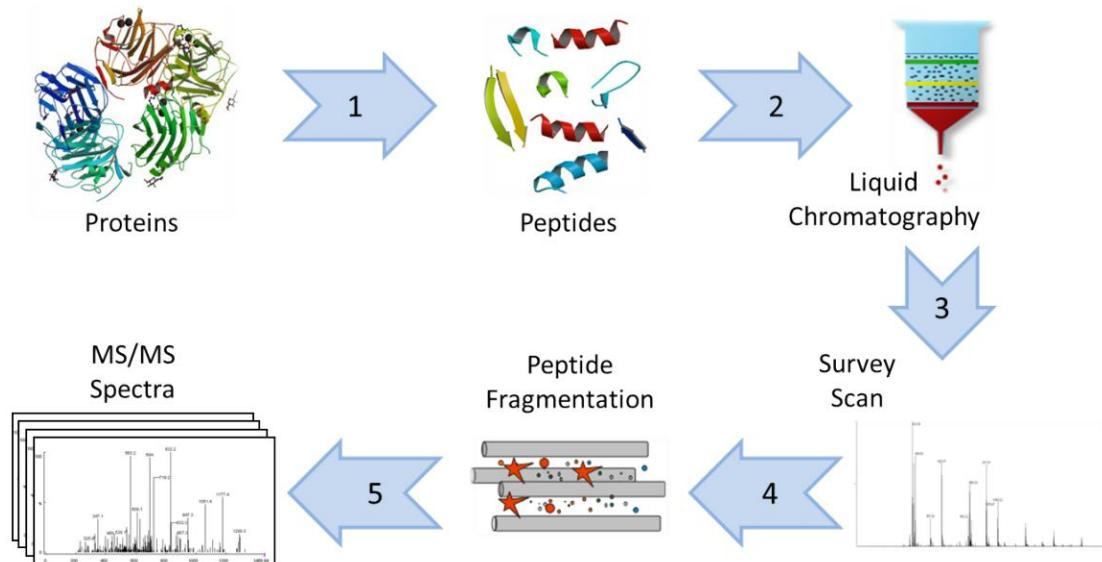


图 1.2 串联质谱流程示意图[16]

1) 蛋白质酶解; 2) 肽段分离; 3) 扫描得到一级质谱; 4) 肽段碎裂; 5) 得到串联质谱。

1.3 基于串联质谱技术的糖肽鉴定

基于质谱技术的糖肽鉴定方法主要可以分为对完整糖肽的鉴定和对糖和肽段依次鉴定两种方法, 如图 1.3 所示。下面分别介绍这两种方法。

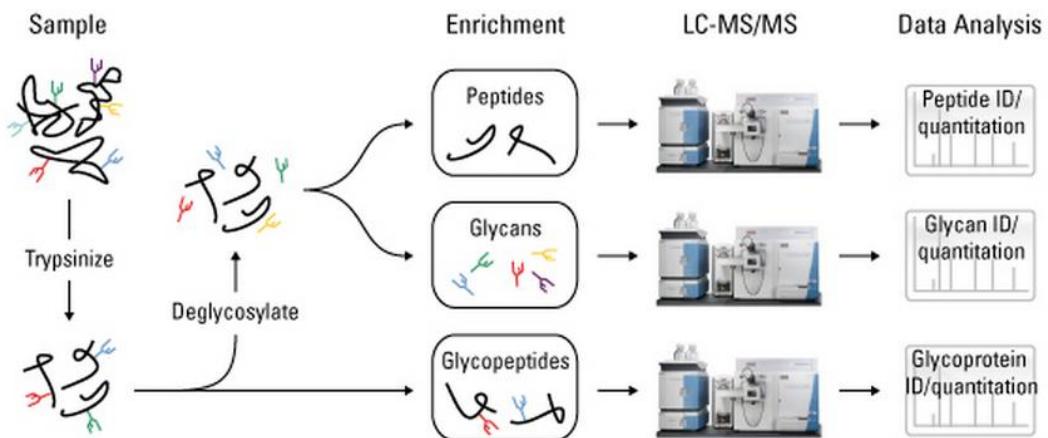


图 1.3 糖肽鉴定流程

糖蛋白被酶解成糖肽后, 或者直接经过液相色谱分离继而碎裂得到串联质谱 (LC-MS/MS), 或者先切掉糖链并分别对糖和肽段进行富集之后再进行 LC-MS/MS (图片引用自 ThermoFisher 官网)

一种方法将糖肽中的糖链与肽段进行分离，分别进行质谱鉴定后组合鉴定结果。人们发现用 N-糖酰胺酶对 N-糖进行酶解之后，糖基化位点 N 会变成天冬氨酸 D，相当于糖基化位点 N 上发生了修饰，称之为脱酰胺修饰。基于这一发现，我们可以对糖肽中切掉糖链的肽段进行固定位点的可变修饰鉴定，其中，固定位点为糖基化位点 N，可变修饰为脱酰胺修饰，从而实现对肽段部分的鉴定。而糖链部分可以直接根据完整糖肽的谱图进行鉴定。然而，切糖链方法中仅仅根据糖链的质量可能无法完全区分出不同的糖链，根据肽段质量来推断肽段序列也存在类似的问题，且去糖基化的糖肽谱图与非糖肽谱图之间没有明显的区分性，这些都可能导致鉴定的错误率增加[17]。

另一种方法直接对完整糖肽样品进行质谱鉴定，该方法根据母离子生成的二级谱图数目可以细分为两类方法。如果对母离子只进行一次碎裂，得到一张糖肽二级谱图，则称之为一母一谱的 N-糖肽鉴定方法。比如，GlycoPep Detector[18]枚举出了所有可能的糖肽组合，并模拟碎裂生成了糖肽对应的理论碎片离子，从而对母离子碎裂得到的 ETD 谱图进行解析。然而，糖肽的一张二级谱图提供的碎片离子的信息相当有限，给鉴定软件带来了很大的挑战。因此，人们继而用质谱仪对母离子进行多次碎裂，得到多张糖肽二级谱图，称之为一母多谱的 N-糖肽鉴定方法。比如，GlycoMaster DB[19] 将母离子分别碎裂成 HCD 谱图和 ETD 谱图。通过 HCD 谱图筛选出糖肽谱图并解析出糖组成，通过 ETD 谱图解析出肽段序列。与切糖链方法中可能出现的质量无法区分以及谱图容易混淆的问题相比，对完整糖肽样品的质谱鉴定保留了糖结构的连接位点信息，鉴定结果更加可靠 [17]。

1.4 公开的糖结构库及糖结构的表示方法

依赖于数据库搜索的串联质谱技术是当前糖肽鉴定的主流方法，因此，蛋白序列库和糖结构数据库的构建至关重要。本节中，我们对当前主要的公开糖结构库以及糖结构的表示方法进行了介绍，并指出了当前糖结构库用于糖肽鉴定存在的问题以及糖结构表示方法的一些不足。

1.4.1 公开的糖结构库

随着互联网技术的发展，人们相继构造了各种数据库如 CarbBank[20], GlycoSciences.de[21], KEGG[22-24], CFG[25], BCSDB[26], JCGGDB[27], EUROCARBDB[28], UniCarbDB[29], GlycoSuiteDB[30, 31], GlycomeDB[32-34], GlyTouCan[35]来存储糖链信息。表 1.1 给出了这些糖库的网址，下面分别对这些糖库进行介绍。

表 1.1 糖库及其网址

糖库	糖库网址
CarbBank	http://glycomics.ccrc.uga.edu/GlycomicsPortal/showEntry.action?id=46
GLYCOSCIENCES.de	http://glycosciences.de/
KEGG	http://www.genome.jp/kegg/glycan/
CFG	http://www.functionalglycomics.org/
BCSDB	http://csdb.glycoscience.ru/bacterial/
JCGGDB	http://jcggdb.jp/
EUROCARBDB	http://relax.organ.su.se/eurocarb/home.action
UniCarbKB	http://unicarbkb.org/
GlycoSuiteDB	http://web.expasy.org/glycosuitedb.html
GlycomeDB	http://www.glycome-db.org/
GlyTouCan	https://glytoucan.org/

CarbBank

1992 年，第一个公开的碳水化合物结构库 CarbBank 建成，90 年代中期由于资金不足而停止维护[36]。该数据库中包含从文献中提取的大概 50,000 个糖结构[37]，不少独立的糖结构库也随之建成，其中不少糖库都是由单个的研究组构建的[38-40]。

GLYCOSCIENCES.de

GLYCOSCIENCES.de 是最早的糖组学网站之一，是由德国癌症研究中心 (German Cancer Research Center) 在 CarbBank 的基础上开发的，现在专注于糖的三维结构。

KEGG

21 世纪初期，KEGG GLYAN 作为一个新的糖结构库加入到了 KEGG 资源

库中。KEGG 糖库包含经实验证的糖结构, CarbBank 和文献中的糖结构信息, 以及从其它 KEGG 资源库中搜集到的包含基因和路径信息的糖信息。KEGG 分为 17 个库, 涵盖系统、基因、化学、健康这四个类别的信息, 用于理解生物系统的高级功能, 截至 2016 年 12 月 16 日, KEGG 已经收录了 11,015 个糖结构。

CFG

NIH 资助的功能糖组学协会(consortium for functional glycomics) 构建了一个用来支持聚糖阵列(glycan array)、质谱分析等的数据库网站, 称之为 CFG[37]。该网站中包含来自从 CFG 中导入的哺乳动物的聚糖阵列(glycan array)以及从其它数据库如 CarbBank、GlycoMinds Ltd 中搜集到的糖结构[36]。

BCSDB

俄罗斯构建的 BCSDB 包含从文献中搜集到的细菌、植物和真菌糖结构和核磁共振数据。具体来说, BCSDB 几乎收录了 1941 年到 2015 年发表的文献中所有的细菌糖结构, 包含来自 6,433 个微生物的 11,605 个糖结构。

JCGGDB

JCGGDB 是一个整合了日本所有糖资源的网站, 由 JCGG 构建。GlycoEpitope 作为 JCGGDB 中主要的数据库之一, 包含了从文献中搜集的糖抗原和抗体。此外, JCGGDB 也发布了一个糖肽数据库 GlycoProtDB, 该数据库中包含通过实验验证的糖基化位点的信息。

EUROCarbDB

2005 年, European Union 构建了 EUROCarbDB[36]。EuroCarbDB 提供了一系列糖信息学工具, 如 GlycoPeakFinder、AutoGU、Gasper、Glycoworkbench、MonosaccharideDB、GLYCOSCIENCE.de、GlycomicsPortal 等。此外, EuroCarbDB 存储了一系列糖结构相关的实验数据, 包括串联质谱信息, HPLC 和 NMR 等[41]。

UniCarbKB

UniCarbKB 在 GlycoSuiteDB 的基础上, 按照 EUROCarbDB 的搜索标准整合了多种资源的糖蛋白数据库[42]。该数据库提供糖结构、糖蛋白、糖基化位点以及相关的参考文献等。

GlycoSuiteDB

GlycoSuiteDB, 澳大利亚开发的一款商业性的数据库, 包含发表文章中的糖蛋白对应的糖结构以及糖蛋白的信息, 以及鉴定糖结构的生物背景信息, 相关数据信息均已经过实验证。GlycoSuiteDB 最近成为 ExPASy 中的一部分, 并且已被整合到 UniCarbKB 中。

GlycomeDB

GlycomeDB 集成了七个公开的数据库[43], 分别为 BCSDB、CarbBank、CFG、

GlycoBase(Dublin)、GlycoBase(Lille)、GLYCOSCIENCES.de 和 KEGG。GlycomeDB 还提供了 JAVA 实现的软件 GlycoUpdateDB，该软件可以下载这七个公开数据库，并将这些数据库中的糖结构按照 GlycoCT 格式和一种类似于 Glyde 的格式 Glyde II 存储，最终将糖结构及其在原数据库中的 ID 信息存储在 GlycomeDB 中。

GlyTouCan

GlyTouCan 中包含很多公开糖库中的糖结构，比如 GlycomeDB、BCSDB、GlycoEpitope 等，提供了 GlycoCT 表示、Linear Code 表示及 KCF 表示。GlyTouCan 中给出了相应糖结构库的链接，且正在尝试链接到 UniCarbKB。有些数据库中可能包含重复的糖结构。对于这些糖结构，GlyToucan 只会展示其中一个糖结构，并在 Linked DB 区域列出原始数据库中所有的糖结构。GlyTouCan 的一大特色就是提供了注册糖结构的功能，注册的用户和时间信息会包含在糖结构相关信息中，且每个糖结构会有一个唯一的编录号便于引用。有一些糖结构由于不能用 GlycoCT 格式表示而不能在 GlyTouCan 中展示。为了解决这个问题，GlyTouCan 2.0 打算将 WURCS (Web 3.0 Unique Representation of Carbohydrate Structures) 作为基本表示方法。

这些糖结构库是人们基于实验分析或者文献报道构造的。然而，人们对糖基转移酶[8, 44]的研究尚处于探索阶段，糖链在生物体内的合成机制仍没有模板，因此，自然界中可能仍存在一些未经实验发现也未被记录在文献中的糖结构，即使对当前公开的所有糖结构库进行合并，得到的也是不全面的糖库。数据库的构建在糖肽鉴定过程中发挥着关键的作用，不完整的库可能导致鉴定输在起跑线上。近年来，不少文献如[36, 39, 45]多次强调了当前糖库的不完整性。如果糖库不全，正确的糖结构没有包含在糖库中，则依赖于糖库搜索的糖肽鉴定软件将无法搜索到正确的糖结构，从而导致鉴定错误。本文旨在构建更加全面的理论糖结构库，从而有效地克服鉴定过程中糖结构库不全这一局限。

1.4.2 糖结构的表示方法

肽段的线性结构使其可以直接在计算机中用线性字符串表示，而糖的分支结构大大增加了其在计算机中存储的难度。于是，数据库构建者纷纷提出了不同的糖结构的存储格式，比如，GlycoSciences.de 采用 LINUCS 格式[46]，KEGG 采用 KCF 格式[47]，EUROCarbDB 采用 GlycoCT 格式[48]，却导致了很多互不兼容的存储格式的产生[36, 39]，且大部分存储格式只用于了一个糖库，图 1.4 为 GlycomeDB 中 ID 值为 14282 的糖结构。图 1.5 展示了该糖结构按照不同的存储方法得到的编码格式。[41]把糖结构的存储格式分为三类。第一类是把糖结构表

示成几行文本，如 CarbBank 数据库用的格式[20, 49]；第二类是把糖结构线性化，把分支作为插入部分，比如 LINUCS 格式[46]、BCSDB 数据库用的格式[50]、Linear Code 格式[51]，这些格式都对分支进行排序以保证每个糖结构的线性表示唯一；第三种是连接表的方式，比如，KCF 格式[47]、GlycoCT 格式[48]等。下面具体介绍 KCF 存储格式和 Linear Code 存储格式。

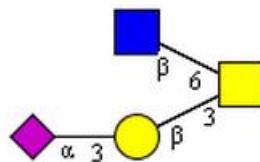


图 1.4 GlycomeDB 中 ID 值为 14282 的糖结构

A	<chem>[] [a-D-GalpNAc]{[(3+1)][b-D-Galp]{[(3+2)][a-D-Neup5Ac]{}{(6+1)}[b-D-GlcNAc]{}}}</chem>																																							
B	<chem>Ac(1-5)aXNeup(1-3)bDGAlp(1-3)[Ac(1-2)bDGlcN,Ac(1-2)]bDGAlpN</chem>																																							
C	$ \begin{array}{c} \text{b-D-GlcNAc- (1-6)} + \\ \\ \text{a-D-GalpNAc} \\ \\ \text{a-D-Neup5Ac- (2-3) -b-D-Galp- (1-3) +} \end{array} $																																							
D	<p>TRP type : α-D-GalNAc Sugars: [incr. 3,3] α-D-NeuAcp (2-3) (acidic) [incr. 3] β-D-Galp (1-3) (neutral) [incr. 6] β-D-GlcNAcp (1-6) (neutral)</p>																																							
E	<p>RES 1b:a-dgal-HEX-1:5 2s:n-acetyl 3b:b-dgal-HEX-1:5 4b:a-dgro-dgal-NON-2:6 1:a 2:keto 3:d 5s:n-acetyl 6b:b-dglc-HEX-1:5 7s:n-acetyl LIN 1:1d(2+1)2n 2:1o(3+1)3d 3:3o(3+2)4d 4:4d(5+1)5n 5:1o(6+1)6d 6:6d(2+1)7n</p>																																							
F	<table border="1"> <thead> <tr> <th>ENTRY</th> <th>G00208</th> <th>Glycan</th> </tr> </thead> <tbody> <tr> <td>NODE</td> <td>5</td> <td></td> </tr> <tr> <td>1</td> <td>Ser/Thr</td> <td>13 0</td> </tr> <tr> <td>2</td> <td>GalNAc</td> <td>3 0</td> </tr> <tr> <td>3</td> <td>Gal</td> <td>-5 -4</td> </tr> <tr> <td>4</td> <td>GlcNAc</td> <td>-6 4</td> </tr> <tr> <td>5</td> <td>Neu5Ac</td> <td>-14 -4</td> </tr> <tr> <td>EDGE</td> <td>4</td> <td></td> </tr> <tr> <td>1</td> <td>2:a1</td> <td>1</td> </tr> <tr> <td>2</td> <td>3:b1</td> <td>2:3</td> </tr> <tr> <td>3</td> <td>4:b1</td> <td>2:6</td> </tr> <tr> <td>4</td> <td>5:a2</td> <td>3:3</td> </tr> <tr> <td></td> <td>///</td> <td></td> </tr> </tbody> </table>	ENTRY	G00208	Glycan	NODE	5		1	Ser/Thr	13 0	2	GalNAc	3 0	3	Gal	-5 -4	4	GlcNAc	-6 4	5	Neu5Ac	-14 -4	EDGE	4		1	2:a1	1	2	3:b1	2:3	3	4:b1	2:6	4	5:a2	3:3		///	
ENTRY	G00208	Glycan																																						
NODE	5																																							
1	Ser/Thr	13 0																																						
2	GalNAc	3 0																																						
3	Gal	-5 -4																																						
4	GlcNAc	-6 4																																						
5	Neu5Ac	-14 -4																																						
EDGE	4																																							
1	2:a1	1																																						
2	3:b1	2:3																																						
3	4:b1	2:6																																						
4	5:a2	3:3																																						
	///																																							
G	<chem>NNa3Ab3(GNb6)AN</chem>																																							

图 1.5 GlycomeDB 中 ID 值为 14282 的糖结构的不同编码格式[41]

A) LINUCS, B) BCSDB 编码, C) CarbBank 的格式, D) GlycoBase(Lille)的格式, E) GlycoCT 格式, F) KCF 格式, G) Linear Code 格式。

1.4.2.1 KCF 格式

KEGG 用 KCF (KEGG Chemical Function) 格式(图 1.5F)来表示糖结构的化学结构。KCF 格式使用的是图表示法，其中节点表示单糖，边表示连接位点。KCF 糖结构表示格式可以分为三个部分，分别为 ENTRY, NODE 和 EDGE。ENTRY 行对应的是糖结构的名字，不一定所有糖结构的 KCF 表示都包含糖结构的名字，即这一行不是必填项。NODE 行对应的数字表示该糖结构是由多少个糖单元构成的。接下来几行的格式为“ n name x y ”，其中 n 表示单糖的标号，name 为单糖的名字， x 和 y 分别是在二维空间中画这个糖结构时单糖的 x 和 y 坐标。

EDGE 行对应的数字表示边的数目。接下来几行的格式为“ e $n_1:h c_1, n_2:c_2$ ”，其中 e 为第 e 个连接位点， n_1 和 n_2 对应于相邻两个单糖的标号， h 可以是 a 或者 b ， c_1 和 c_2 分别表示标号值为 n_1 和 n_2 的相邻单糖的碳数目。最后，三个斜杠表示 entry 结束[52]。图 1.5 (F) 展示了图 1.4 所示糖结构的表示方法，从 NODE 行和 EDGE 行可以看出该糖结构包含五个节点和四条边，其中，五个节点的标号如图 1.6 所示。

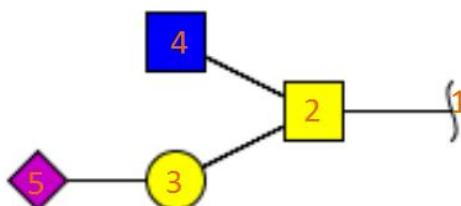


图 1.6 图 1.4 所示糖结构的标号值

1.4.2.2 Linear Code 格式

Linear Code 格式是由 GlycoMinds Ltd 定义的一种糖格式(图 1.5G)，它可以用来处理不确定或者未知的信息。Linear Code 采用单糖的单字符术语以简化糖结构的表示，并且用一种紧凑的格式表示糖苷键。Linear Code 格式中，用 a 和 b 分别表示 α 和 β ， a 和 b 的后面为残基连接端包含的碳的数目，表示相邻单糖之间的连接位置[52]。比如，糖结构 β -D-Galp(2P)-(1-3)- β -D-Glcp 的 Linear Code 格式

为 A[2P]b3Gb, 其中, D-GlcP (全称为 D-Glucose) 的单字符表示为 G, D-GalP(全称为 D-Galactose)的单字符表示为 A。对于图 1.4 所示的糖结构, 其 Linear Code 为 NNa3Ab3(GNb6)AN, 从右往左读, 分支放在括号内。其中, D-GalpNAc (全称为 N-Acetylgalactosamine) 的 Linear Code 为 AN, D-Galp (全称为 D-Galactose) 的 Linear Code 为 A, D-Neup5Ac(全称为 N-Acetylneurameric acid) 的 Linear Code 为 NN, D-GlcPNAc (全称为 N-Acetylglactosamine) 的 Linear Code 为 GN。

糖结构的分支结构使得其在计算机种的存储格式不如线性的肽段那么直观。于是, 数据库构建者纷纷提出了不同的糖结构的存储格式, 却导致了很多互不兼容的存储格式的产生[36, 39], 给不同糖结构库之间的相互关联带来了很大的不便[46]。大部分表示方法既不能同时满足易于人工解读及计算机存储这两个人性化需求, 也不适用于糖结构的同构判定。比如, KCF 格式采用的是图表法, 这种存储方法的空间效率不高, 且不能方便地区分出同构的糖结构。Linear Code 格式是线性的存储格式, 虽然便于计算机存储, 但是这种表示方法将糖结构的每一层都由相同的括号外包, 加大了糖结构的解码即由 Linear Code 格式推出糖结构的拓扑关系的难度, 给人工解读带来了不便, 尤其是对于较大的糖结构。LINUCS 格式也是一种线性正则的糖结构表示方法, 但是这种方法是从根节点往叶节点遍历糖结构, 这就要求增加额外的比较规则来确定遍历兄弟节点的顺序。并且, LINUCS 格式的排序规则中使用了连接位点的信息, 因此这一表示方法不能直接用于糖结构的同构判定。

1.5 本文的贡献

本文主要围绕 N-糖结构库的构建展开研究, 对糖结构的同构判定、大规模糖结构库的枚举、糖结构库的构建速度等方面进行了优化, 同时提出了一种新的 N-糖结构的线性正则表达式, 使得分支的糖结构也能像肽段序列一样通过线性字符串码来存储, 从而极大地方便了糖结构的查询以及不同糖结构库之间的比较与合并。下面对本文的主要贡献进行详细阐述, 主要分为如下三个方面:

第一, 我们提出了一种新的 N-糖结构的线性正则表达式, 一方面, 通过把非线性的糖结构编码为线性正则的字符串码, 我们为糖结构提供了一种线性的存储格式。该表示方法不仅便于计算机存储、查找及解析, 还具备易于人工解读的性质, 即给定糖结构的字符串表示, 人们可以很轻松地推出糖结构的糖组成及不同单糖之间的连接方式。另一方面, 该编码方法保证糖结构同构当且仅当糖结构对应的线性正则表达式相同, 从而为糖结构的同构判定提供了一种应用高效且理

论层面优美的方法。

第二，我们基于糖结构的线性正则表达式，设计了一种新的包含互不同构糖结构的理论 N-糖结构库的构建方法，不仅节省了糖结构的存储空间，而且还有有效地提高了糖结构库构建的效率。具体实现了根据糖结构的字符串码往糖结构中加入新的节点、调整糖结构的正则顺序从而构造新的正则糖结构、计算糖结构中每个节点的分支数目、判断糖结构的合理性以及生成糖结构的 Y 离子（含根子结构）的组成等算法。与基于邻接表的方法构建不包含同构糖结构的理论 N-糖结构库的方法相比，我们直接基于糖结构的线性字符串码枚举糖结构，可以避免由糖结构的邻接表存储格式到糖结构的线性字符串码之间的转化操作。我们设计了一些有效的剪枝策略，从而大大减少了冗余糖结构的生成，进一步提高了枚举的效率。值得一提的是，通过探究糖结构 Y 离子构造的本质，我们发现母结构的 Y 离子与子结构 Y 离子的组合之间的关系，从而极大地提高了糖结构 Y 离子的生成效率。

第三，我们发现糖结构库中存在一些 Y 离子组成完全相同的糖结构，这类糖结构的存在可能会降低糖肽鉴定软件如 pGlyco 2.0 (pGlyco[53]的改进版本) 的性能，且糖肽鉴定软件无法根据糖结构的理论碎裂区分出这些糖结构。我们通过将糖结构库中 Y 离子组成相同的糖结构归并为同一类，使得归并后的糖结构库更加适用于基于质谱技术的糖肽鉴定软件。进一步，我们合并了 GlycomeDB 中的 N-糖结构以及人类血清中三个最大的通用结构的子结构，得到一个包含常用糖结构的合并库，通过对基于归并后的枚举库及基于归并后的合并库的鉴定结果，我们发现枚举库有助于发现潜在的新的 N-糖结构，从而为新的糖结构的发现提供了一种有效的途径。

1.6 论文的组织

本文的各章内容组织如下：

第一章介绍了蛋白质的糖基化修饰、基于串联质谱技术的蛋白质鉴定和糖肽鉴定、糖结构的表示方法等一些基本知识，并对当前公开的糖结构库及糖结构的存储格式进行了调研。

第二章介绍了糖结构同构判断的方法，首先采用哈希剪枝提高糖同构判断的效率，然后基于一些树同构的判断方法，提出了两种新的糖结构编码方法，使得糖结构的同构判定算法不仅在应用层面高效，而且从理论的角度来看也比较优美。

第三章介绍了基于糖结构的邻接表存储格式的理论 N-糖结构库构建。我们枚举了给定节点数目范围的理论糖结构，并去掉了这些糖结构中的同构冗余。为

了避免糖库规模太大引入过多的假阳，我们提供了两种缩减理论 N-糖结构库的策略，从而使枚举库更加适用于糖肽鉴定。基于糖结构的邻接表，我们还可以生成糖结构的 Y 离子，使得糖肽鉴定更加方便。

第四章中我们基于新提出的糖结构的线性正则编码方法，实现了根据糖结构的字符串码生成新的糖结构、调整糖结构的正则顺序、计算糖结构中每个节点的分支数目以及生成糖结构的子结构等算法。我们分别用基于糖结构的字符串码和基于糖结构的邻接表构造理论糖结构的方法枚举出 6~10 个节点的糖结构，并对齐了这两种不同枚举算法的枚举结果。

第五章中我们将 GlycomeDB 中的 N-糖与人类血清中三个最大通用结构的子结构进行合并得到合并库，通过对枚举库及合并库中 Y 离子组成相同的糖结构进行统计分析，发现枚举库中大规模等价类的存在可能降低 pGlyco 2.0 的鉴定性能。我们把枚举库的糖结构按照 Y 离子组成进行归并，使得 pGlyco 2.0 能够基于枚举库实现有效的糖肽鉴定，且验证了基于枚举库可以有助于发现潜在的新的 N-糖结构。

第六章中我们基于糖结构的线性正则编码方法设计了搜索引擎 gMatch，该搜索引擎可以有效地提高糖肽谱图鉴定结果的召回率。

第二章 糖结构同构判定方法探究

由于当前常用的质谱技术产生的碎片离子信息还不足以体现出糖结构的连接位点信息[19]，一些依赖于数据库搜索的糖肽鉴定软件 GlycoMaster DB[19], GRIP[54], ArMone2.0[55], GlycoPeptideSearch[56], GPQuest[57], I-GPA[58], pGlyco[53] 只考虑了单糖之间的拓扑结构，而忽略了单糖之间的连接位点信息。不考虑连接位点信息的 N-糖结构可以看成是有根树，由于同构的糖结构对应的拓扑结构完全相同，我们称同构的糖结构互为冗余。为了保证糖库中糖结构的唯一性，我们需要去掉糖结构中的冗余结构，即对于同构的糖结构，只保留其中一个。首先，我们采用对 n 个糖结构进行两两遍历的方法来判断糖结构之间的同构关系，并采用哈希剪枝的方法对糖结构的冗余判定进行加速。虽然哈希剪枝的加速效果不错，但是，哈希值与同构的糖结构之间并不是一一映射。我们希望糖结构的同构判定算法不仅应用起来高效，而且从理论的层面来看是优美的。由于糖结构可以看成是树结构，我们进一步对树结构的线性正则编码方法进行了调研，并提出了一种新的易于人工解码的编码方法。

2.1 基于哈希剪枝的糖同构判断方法

给定 n 个糖结构，为了找到这 n 个糖结构中的同构冗余，一种最直观的方法是对这些糖结构进行两两比较。为了判断两个糖结构是否互为同构，我们先比较这两个糖结构的根节点对应的单糖类型是否相同，如果不相同，则这两个糖结构一定不互为同构；如果相同，则继续比较以根节点的孩子节点为根的子结构是否互为同构，如果均互为同构结构，则这两个糖结构互为同构，否则，这两个糖结构互为同构。我们用 $t[v]$ 表示节点 v 对应的单糖类型，具体判断同构的方法如**算法 2.1** 所述。

算法 2.1 基于单糖类型的糖结构同构判定算法

输入: 糖结构 A,记为 Ga; 糖结构 B,记为 Gb; A 的节点 NodeA; B 的节点 NodeB

输出: 糖结构 A 与糖结构 B 是否互为同构结构

Procedure Is_Redundant(Ga,Gb,NodeA,NodeB)

```

if Ga.t[NodeA] != Gb.t[NodeB] or Ga.degree[NodeA] != Gb.degree[NodeB]
    return false
for u ∈ Ga.Adj[NodeA]           #对于糖结构 A 中与 NodeA 相邻的每个节点 u
    find=false
    for v ∈ Gb.Adj[NodeB]         #对于糖结构 B 中与 NodeB 相邻的每个节点 v
        if Is_Redundant(Ga,Gb,u,v)
            find=true      #如果糖结构 A、B 中分别以 u、v 为根的子结构同构
            Ga.Adj[NodeA]=Ga.Adj[NodeA]-{u}#NodeA 的邻接节点集合中去掉 u
            Gb.Adj[NodeB]=Gb.Adj[NodeB]-{v}#NodeB 的邻接节点集合中去掉 v
            break
        if find==false#以 NodeB 的邻接节点为根的结构均不与以 u 为根的结构同构
            return false          #则 A 与 B 不同构, 算法停止
    return true
end procedure

```

下面根据**算法 2.1** 来判断图 2.1 所示的糖结构 A 和 B 是否互为同构。为了描述的方便, 记以 A 的节点 n 为根的糖结构为 A_n , 以 B 的节点 n 为根的糖结构为 B_n , 判断过程如下:

- 1)比较 A_1 与 B_1 对应的单糖类型, 均为 HexNAc;
- 2)判断 A_2 与 B_2 是否互为同构: 比较 A_2 与 B_2 对应的单糖类型, 均为 HexNAc;
- 3)判断 A_3 与 B_3 是否互为同构: 比较 A_3 与 B_3 对应的单糖类型, 均为 Hex;
- 4)判断 A_4 、 A_5 与 B_4 、 B_5 是否分别互为同构: A_4 与 B_5 互为同构结构, A_5 与 B_4 互为同构结构, 据此可知 A 与 B 互为同构结构。

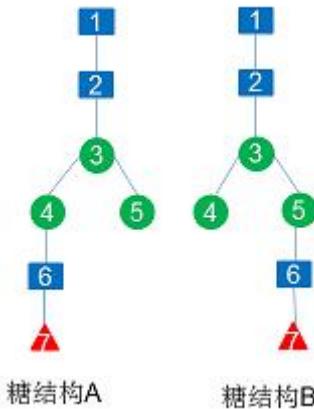


图 2.1 互为同构的两个糖结构

其中，单糖类型及其表示符号分别为：HexNAc（蓝色方块），Hex（绿色圆形），NeuAc（紫色菱形），dHex（红色三角形）。

按照**算法 2.1** 判断两个糖结构是否互为同构，需要分别对这两个糖结构进行遍历。如果两个糖结构同构，则至少需要对两个糖结构的每个节点分别遍历一次；如果这两个糖结构不同构，且只是叶节点对应的单糖类型不同，则算法递归至叶节点才能停止。假设这 n 个糖结构互不同构，每个糖结构都需要与另外 $n-1$ 个糖结构进行比较，而每次比较都需要对糖结构的节点进行访问及比较直至发现两个糖结构不同构。最坏情况下，每个糖结构都需要进行 $n-1$ 次从根节点到叶节点的遍历。如果我们先把糖结构遍历一遍，并且记录相应的哈希值，则可以在一定程度上减少每个糖结构的遍历次数。基于这一思想，我们依次设计了一系列哈希方法对遍历过程进行剪枝，下面具体介绍自底向上的哈希方法。自底向上哈希方法将糖结构的单糖种类、分支信息及层次信息从叶子节点往上汇总到根节点。我们将糖结构的单糖、边和节点分别按照不同的哈希函数进行哈希。为了方便描述，我们称糖结构的根节点对应的哈希值为糖结构的哈希值。

2.1.1 单糖的哈希

考虑到质数是数学这门语言的基石，我们分别用不同的质数作为单糖的哈希值，如表 2.1 所示。

表 2.1 单糖的哈希值

单糖类型	Hex	HexNAc	NeuAc	dHex
哈希值	3	5	7	11

2.1.2 边的哈希

由于树是有层次结构的，因此，对边的哈希需要体现出层次差异，即对一条边的两个端点设置不同的哈希值。记边的靠近根的节点对应的单糖的哈希值为 ValueU，边的远离根的节点对应的单糖的哈希值为 ValueD，则边的哈希值为 $(ValueU+13)*ValueD$ 。

2.1.3 节点的哈希

如果为叶子节点，则该节点的哈希值为其对应的单糖的哈希值；如果该节点不是叶子节点，则该节点的哈希值为：该节点对应的单糖的哈希值加上边的哈希值再加上与该节点相连的节点的哈希值的乘积，将这个值乘以 10 并乘以该节点度数，再加上与该节点的邻接节点的哈希值之和。比如，对于图 2.2 所示的糖结构，假设根节点对应的层数为 1，则第 3 层的节点对应的哈希值为： $(3+48+48+3*3)*10*2+3+3=2166$ 。

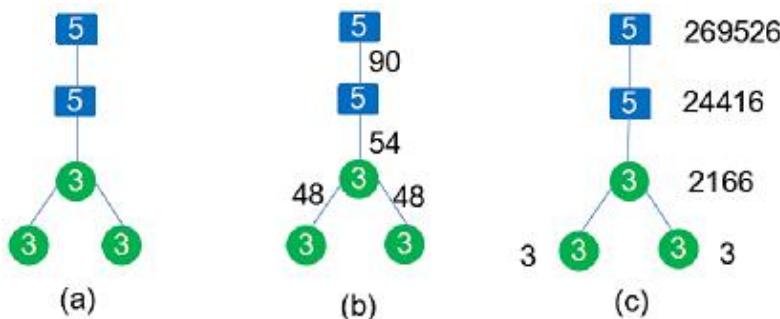


图 2.2 糖结构的哈希方法示意图

(a)单糖的哈希值 (b)边的哈希值 (c)节点的哈希值

2.1.4 自底向上哈希方法的效率测试

如果糖结构的根节点哈希值不同，则可以直接判断出这些糖结构不互为冗余，从而有效地对递归算法进行剪枝；如果糖结构的根节点哈希值相同，则需要对这些糖结构进行两两判断是否同构。哈希方法越强，哈希值冲突的次数就越少，从而能有效地减少糖结构之间的比较次数。当哈希方法达到一一映射的效果，即两个糖结构是否互为同构等价于判断这两个糖结构根节点的哈希值是否相等时，对于哈希值相等的糖结构只需保留其中一个，而不需按照递归算法两两比较。

我们分别用基于自底向上的哈希方法及直接对糖结构两两遍历的方法去冗

余，以五糖核心为基础枚举出 6~10 个节点数目的糖集合分别耗时 23.376s 与 1173.57s，哈希方法的加速比达到 50.2。

为了进一步分析哈希方法对于不同糖结构集合的加速效果，我们构造了两个测试集。测试集 1 中包含互不为同构的 31429 个节点数目为 6~10 的糖结构，测试集 2 中包含 77268 个节点数目为 6~10 的糖结构，该测试集中每个糖结构都有其对应的同构结构，去冗余之后有 20993 个糖结构。

按照自底向上的哈希方法，可以把测试集 2 中的所有不为同构的糖结构哈希到不同的值，即哈希值的冲突为 0。考虑到互为同构的糖结构的质量一定相等，糖结构的质量值可以作为一种哈希方法。进一步，我们不仅对比了自底向上的哈希方法与简单的遍历方法的去冗余性能，而且将之与另一种按照质量进行哈希的去冗余性能进行了对比。在 Intel(R)Core(TM) i5-4570 处理器，16GB 内存，windows 7 64 位操作系统下，两组对比结果分别如表 2.2、2.3 所示。

表 2.2 自底向上哈希与遍历方法的效率对比

去冗余所需时间	自底向上哈希(s)	遍历方法(s)	加速比
测试集 1	0.326	296.31	908.92
测试集 2	0.942	581.749	617.56

表 2.3 自底向上哈希与质量哈希方法的效率对比

去冗余所需时间	自底向上哈希(s)	质量哈希(s)	加速比
测试集 1	0.319	15.378	48.207
测试集 2	0.939	36.329	38.675

2.1.5 哈希方法总结

以上哈希方法均是通过单糖的哈希值和边的哈希值组合得到节点的哈希值，而同构糖结构的单糖类型及边一一对应，故同构糖结构一定对应着相同的哈希值，即哈希值满足如下性质：

性质 1 互为同构的糖结构的哈希值一定相等，但哈希值相等的糖结构不一定互为同构。

算法 2.1 介绍的去冗余方法在深搜的过程中比较的是节点对应的单糖类型，而由性质 1 可知，互为同构的糖结构对应的相同的哈希值，哈希值不同的糖结构一定不为同构，故可以将算法 2.1 改进为比较糖结构中节点的哈希值而不是标号值。另一方面，我们计算出糖结构的哈希值之后，可以先将糖结构按照其对应的哈希值进行分组，不同组的糖结构一定不互为同构，只需对同一组的糖结构进行同构判定，从而有效地省去糖结构之间一些不必要的比较操作。

2.2 树同构判定方法调研

不考虑连接位点信息的 N-糖结构可以看作是有根无序有标记树，为了更好地实现糖结构的同构判断，我们对树结构的同构判断方法进行了相关的调研。这里具体介绍其中六种树同构判定算法，分别为用于判断两棵树是否同构的 AHU74 算法、用于判断无根无序无标号树是否同构的 KS99 算法、基于树的序关系的字符串编码方法、用于判断有根无序有标记树是否同构的广度优先字符串编码方法和深度优先字符串编码方法，然后再对这些算法进行简短的总结。

2.2.1 判断两棵树是否同构的 AHU74 算法

1974 年，Aho 等人提出了一种用于判断两棵有根无序无标号树是否同构的算法[59]，本文简称该算法为 AHU74 算法，具体的方法如**算法 2.2** 所述。

2.2.1.1 AHU74 算法描述

算法 2.2 判断两棵树是否同构的 AHU74 算法

1. 把叶子节点赋值为整数 0；
 2. 对于树结构 T1，将第 $i-1$ 层中所有节点按其对应的整数值非降序排列，记排序后的节点集合为 L1；同样，对于树结构 T2，记第 $i-1$ 层按照对应的整数值非降序排列的节点集合为 L2。
 3. 对于树结构 T1，从左到右遍历 L1，把 L1 中节点 v 对应的整数值作为节点 v 的父节点的元组的下一个元素。最终，第 i 层的节点对应的元组为：该节点的孩子节点对应的整数值按照非降序排列；对于树结构 T2，按照相同的方法构造第 i 层中出度大于 1 的节点对应的元组。
 4. 将第 i 层的所有节点的元组按照字典序排序，分别记为 S1 和 S2。如果 S1 与 S2 不相等，则 T1 与 T2 不同构，算法停止；否则，将元组最小的节点赋整数值 1，第二小的节点赋整数值 2，依此类推。 $i=i+1$, 转到 3。
 5. 如果 T1 和 T2 的根节点对应的整数值相等，则 T1 和 T2 同构。
-

2.2.1.2 AHU74 算法示例

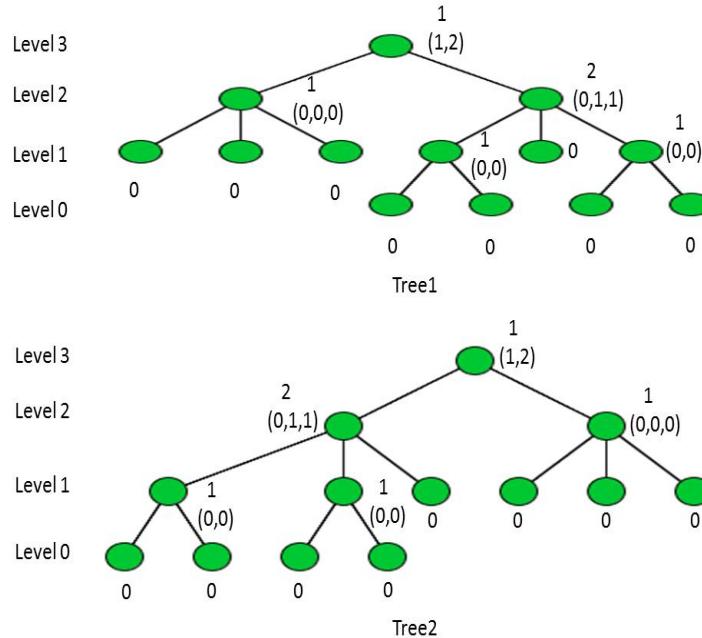


图 2.3 按照 AHU74 算法对两棵树进行同构判断的过程[59]

首先把叶子节点赋值为整数 0，然后从第一层往上分别比较两棵树同一层节点对应的元组。如果两棵树每一层的元组都相同，则这两棵树互为同构；如果两棵树的某一层对应的元组不同，则这两棵树不同构，算法停止。

对于图 2.3 所示的两棵树，我们按照 AHU74 算法判断这两棵树是否同构的过程如下：

- 1) 给叶子节点赋值：分别把两棵树的叶子节点赋值为整数 0；
- 2) 对于第 i 层 ($i=1,2,3$) 的每个节点，该节点对应的元组为其孩子节点对应的整数值按照非降序构成的元组。比如，对于 Tree2 中第二层最左边的节点，该节点的三个孩子节点对应的整数值分别为 1,1,0，则该节点对应的元组为(0,1,1)；
- 3) 分别对两棵树中第 i 层 ($i = 1, 2, 3$) 的所有节点的元组按照字典序排序，得到序列 S_1 和 S_2 。比如，Tree1 中的 Level 2 中的两个节点对应的元组分别为(0,0,0) 和(0,1,1)，则 $S_1=[(0,0,0),(0,1,1)]$ ，Tree2 中的 Level 2 中的两个节点对应的元组分别为(0,1,1) 和(0,0,0)，则 $S_2=[(0,0,0),(0,1,1)]$ 。由于 $S_1=S_2$ ，故需要继续比较两棵树的第三层。

2.2.1.3 AHU74 算法总结

- (1) 该算法的巧妙之处就在于将孩子节点对应的字符串用其序数代替，从而可以保证每个节点对应的字符串的长度刚好等于该节点的出度，很好地控制了每个

节点对应的字符串的长度，但同时也导致了这一方法的局限性——只能对树结构进行两两同构判定，且必须从叶子节点往上一层一层地判断。

(2) AHU74 算法判定两棵数是否同构的时间复杂度为 $O(n)$ 。其中， n 为树节点的数目。

(3) 该算法用于判断两棵有根无序无标号树是否同构，也可通过适当改动使其适用于有根无序有标号树，比如，把节点对应的标号值作为元组中的第一个元素。

(4) AHU74 算法需要对每一层所有节点对应的整数排序，每一层的元组按字典序排序，还需要比较两棵树同一层的元组是否相同。该算法的时间复杂度是 $O(n)$ ，而最基本的遍历方法在最好情况下的时间复杂度也为 $O(n)$ 。时间复杂度虽然级别一样，但系数上是有差别的。而且按照 AHU74 算法我们需要对枚举出的所有树结构进行两两比较，导致总体的同构判定的效率并不高。

(5) 参考文献[60]和[61]中都提到了用 AHU74 算法来调整树结构中节点对应的字符串码的正则顺序。

2.2.2 判断无根无序无标号树是否同构的 KS99 算法

该算法[62]旨在对无根无序无标号树进行字符串编码，简称为 KS99 算法，具体的方法如**算法 2.3** 所述。需要注意的是，这里称度数为 1 的节点为叶子节点。

2.2.2.1 KS99 算法描述

算法 2.3 判断树结构是否同构的 KS99 算法

1. 所有的节点赋给字符串“01”，称之为节点的标号。

2. 对于非叶子节点 v ,

(a)设 Y 为由 v 的邻接叶子节点对应的标号及 v 的去掉最左端'0'与最右端'1'的标号构成的集合；

(b)将 Y 中的元素按照字典序拼接，并在最左端加上'0'，最右端加上'1'，构成节点 v 的新标号；

(c)去掉节点 v 的所有邻接叶子节点。

2.2.2.2 KS99 算法示例

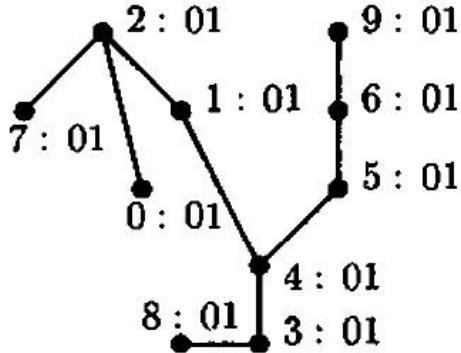


图 2.4 树结构中每个节点的编号值及初始标号值[62]

对于图 2.4 所示的树结构，我们首先将所有节点对应的标号值设置为“01”。按照该算法中叶子节点的定义，编号为 0、7、8、9 的节点为叶节点，其余的节点则为非叶子节点。对于节点 1，与其相邻的节点 2、4 均不为叶子节点，而节点 1 当前的标号为“01”，故节点 1 对应的集合 Y 为空集，从而节点 1 对应的标号仍为“01”；对于节点 2，与其相邻的节点 0、7 为叶子节点，而节点 1 为非叶子节点，且节点 2 当前的标号为“01”，故节点 2 对应的集合 $Y=\{01,01\}$ ，从而更新节点 2 的标号为“001011”。

2.2.2.3 KS99 算法总结

KS99 算法面向的是无根无序无标号树。而对于有根无序无标号树，我们可以把算法简化，如图 2.5 所示：

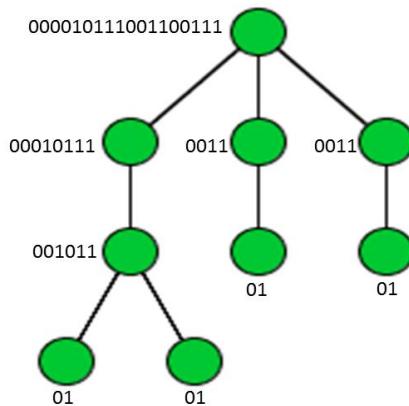


图 2.5 有根无序无标号树及其对应的字符串码

以 r 为根节点的树 T_r 对应的字符串码为 $S_{T_r} = 0S_{T_{r_1}}S_{T_{r_2}} \dots S_{T_{r_k}} 1$ ，其中， r_1, r_2, \dots, r_k 为节

点 r 的孩子节点, $S_{T_{r_1}}, S_{T_{r_2}}, \dots, S_{T_{r_k}}$ 分别为以 r_1, r_2, \dots, r_k 为根的子树 $T_{r_1}, T_{r_2}, \dots, T_{r_k}$ 对应的字符串码, 且按字典序非降序排列。

其编码思路为: 叶子节点的字符串全部初始化为“01”。对于每个非叶子节点, 将该节点的孩子节点对应的字符串按照字典序非降序连接, 并用“01”外包(好像一个个括号用来分隔), 即在该字符串的最左端加入一个‘0’字符, 最右端加入一个‘1’字符。

外包“01”是实现由编码值倒推树结构的关键一步, 它表明了每个节点所在的层次信息。我们假设根节点对应第 0 层, 该树结构的总层数为 k , 则第 i 层中每个节点对应的字符串中最左端有 $k-i$ 个‘0’字符。一层一层地剥开外包的“01”, 便可还原到同构意义上一致的树结构。还可以用[62]中提到的山脉法(mountain range)来实现根节点对应的字符串到树结构的还原。

另外, 按照 KS99 算法, 无标记树的根节点的字符串由且仅由‘0’和‘1’构成, 故可以将其看成是二进制编码。由于节点数为 n 的树的根节点的字符串长度为 $2n$, 故枚举 20 个节点的无标记树只需 40 个 bits, 可以很好地避免溢出现象。

2.2.3 基于树的序关系的字符串编码方法

该算法[63]对孩子节点对应的字符串进行拼接时, 是按照以孩子节点为根的树的序关系进行拼接的, 而本文介绍的其它算法都是按照字符串码本身的序关系拼接的。

2.2.3.1 定义树的序关系

对于两棵树 S 和 T , 先比较这两棵树的节点数目, 如果这两棵树的节点数目相等, 则继续比较这两棵树的根节点的孩子节点的数目, 如果根节点的出度也相等, 则依次比较以这两棵树的根节点的孩子节点为根的子树之间的大小关系。

记 $|T|$ 为树 T 的节点数目, t 为 T 的根节点, $\#t$ 为节点 t 的孩子节点的数目(出度)。对于两棵树 S 和 T , 我们称 $S < T$ 如果满足:

- 1) $|S| < |T|$, 或者
- 2) $|S| = |T|$, 且 $\#s < \#t$, 或者
- 3) $|S| = |T|$, $\#s = \#t = k$, 且 $(S_1, \dots, S_k) < (T_1, \dots, T_k)$, 其中 S_1, \dots, S_k 为树 S 的以 s 的

孩子节点为根的子树, T_1, \dots, T_k 为树 T 的以 t 的孩子节点为根的子树, 且满足

$$S_1 \leq \dots \leq S_k, T_1 \leq \dots \leq T_k.$$

2.2.3.2 有根无序无标号树的字符串编码方法

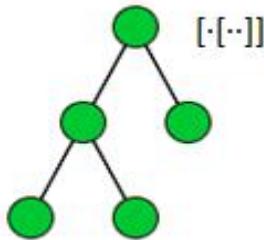


图 2.6 有根无序无标号树及其对应的字符串码

单个节点构成的树的字符串码为“·”, 树 T 对应的字符串码为 $c(T)=[c(T_1), \dots, c(T_k)]$, 其中, T_1, \dots, T_k 为以 t 的孩子节点为根的子树, $c(T_1), \dots, c(T_k)$ 为 T_1, \dots, T_k 对应的字符串码, 且满足 $T_1 \leq \dots \leq T_k$.

2.2.4 广度优先字符串编码方法

2.2.4.1 基本术语

广度优先正则字符串 (Breadth-First Canonical String): 有根无序树对应的有根有序树的广度优先字符串码中, 字典序最小的那个。简记为 **BFCS**。

广度优先正则形式 (Breadth-First Canonical Form): 有根无序树对应的字典序最小的有根有序树。简记为 **BFCF**。

2.2.4.2 广度优先字符串编码方法示例

对于有根有序树, 按照广度优先的顺序遍历树结构, 记录每个节点的标号值。并且, 用 ‘\$’ 来分隔不同的兄弟族, 用 ‘#’ 表示字符串码的末端。假设 ‘\$’ < ‘#’, 且这两个字符均大于大写字母字符。对于有根无序树, 将其对应的有序树的字符串码中字典序最小的那个作为该无序树对应的字符串码[60]。如图 2.7 所示。

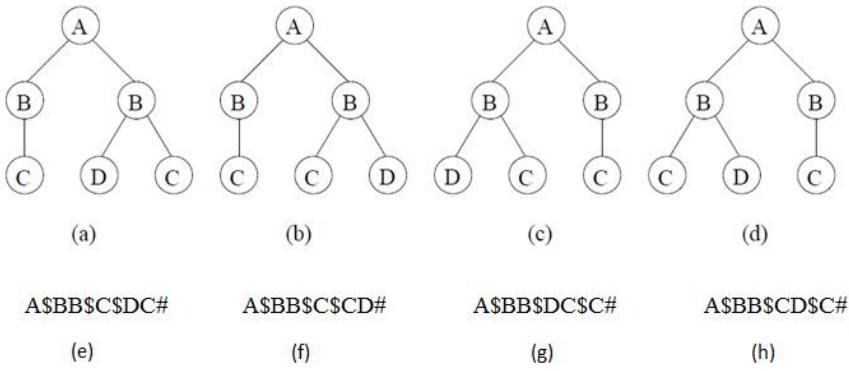


图 2.7 有根树及其对应的字符串码[60]

(a)~(d)为同一棵有根无序树对应的四棵有根有序树, (e)~(h)为这四棵树对应的字符串码。 (h)为字典序最小的字符串码, 故(h)为这棵有根无序树对应的 BFCS, (d)为这棵有根无序树对应的 BFCF

2.2.4.3 BFCS 总结

(1) 论文中给出的例子并没有很好地说明广度优先字符串编码方法, 容易造成误解。比如, 给定字符串码 A\$BB\$C\$DC#, 读者可能会认为其对应着不止一种有根无序树, 如图 2.8 所示的 T1 和 T2。

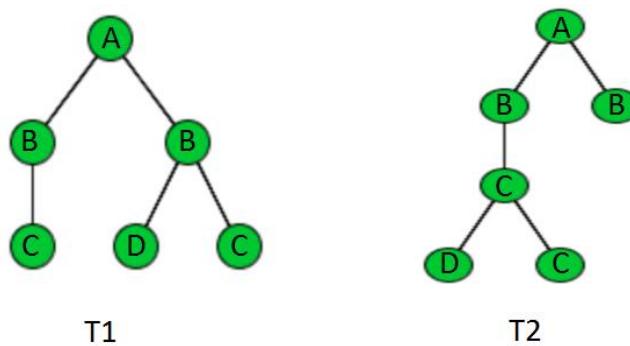


图 2.8 A\$BB\$C\$DC#可能对应的两种有根有序树

我的理解是, T1 对应的字符串码为 A\$BB\$C\$DC#, 而 T2 对应的字符串码为 A\$BB\$C\$\$DC#。即对于第 i 层的节点 v , 即使该节点的孩子节点的数目为 0, 也需要在遍历第 $i+1$ 层节点的过程中, 用 '\$' 表示当前遇到了一个新的父节点。

(2) 该算法假定的是 '\$' < '#' ,且这两个字符均大于大写字母字符。但是实际上, '\$' 的 ASCII 码比 '#' 的 ASCII 码大, 并且 '\$' 和 '#' 的 ASCII 码均比任意大写字母的 ASCII 码都小。因此, 在实现的过程中需要多加注意。

(3) 文章中认为有根无序有标记树的 BFCS 的长度至多为 $3k$, 其中, k 为树的节点的数目。从证明过程中可以看出, 对于节点数目为 k 的树, 作者把 $k-1$ 条边的 $k-1$ 个标号值也计入到 BFCS 的长度中, 但是从上述编码过程来看, BFCS 中

只有节点的标号值，并没有包含边的标号值。因此，我认为 BFCS 的长度至多为 $2k$ 。

2.2.5 深度优先字符串编码方法

2.2.5.1 基本术语

深度优先正则字符串 (Depth-First Canonical String) :有根无序树对应的有根有序树的深度优先字符串码中，字典序最小的那个。简记为 **DFCS**。

深度优先正则形式 (Depth-First Canonical Form) :有根无序树对应的字典序最小的有根有序树。简记为 **DFCF**。

2.2.5.2 深度优先字符串编码方法示例

对于有根有序树，按照深度优先的顺序遍历树结构，记录每个节点的标号值。并且，用 ‘\$’ 表示一次回溯，并用 ‘#’ 表示字符串码的末端。假设 ‘\$’ < ‘#’，且这两个字符均大于大写字母字符。对于有根无序树，将其对应的有序树的字符串码中字典序最小的那个作为该无序树对应的字符串码[60]。如图 2.9 所示。

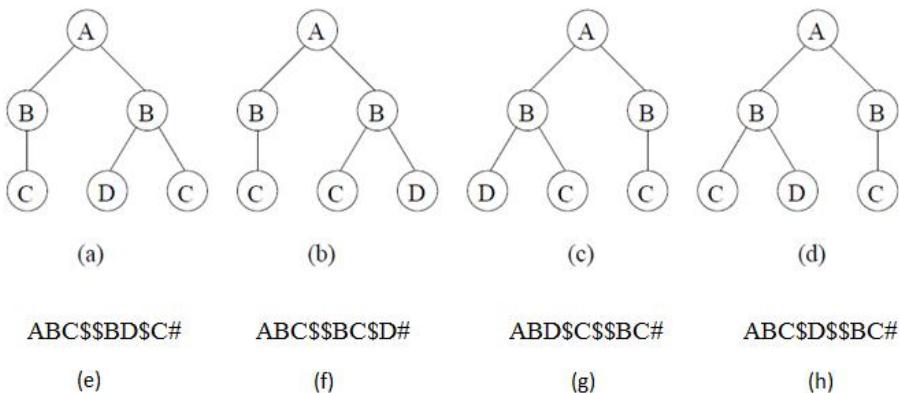


图 2.9 有根树及其对应的字符串码[60]

(a)~(d) 为同一棵有根无序树对应的四棵有根有序树，(e)~(h) 为这四棵树对应的字符串码，其中，(h) 为字典序最小的字符串，故(h) 为这棵有根无序树对应的 DFCS，(d) 为这棵有根无序树对应的 DFCF

2.2.5.3 基于 AHU74 算法构造 DFCF

2.2.1.3 节对 AHU74 算法[59]的总结中的第三条提到了，可以通过把节点对

应的标号值作为元组中的第一个元素，使得 AHU74 算法适用于有根无序有标号树。在构造 DFCF 的过程中，从底向上对有根无序树中同一层的节点按照 AHU74 算法进行编码后，根据编码值对该层的节点进行排序。具体来说，先比较节点的标号值，再比较这些节点的孩子节点的序数，如图 2.10 所示。

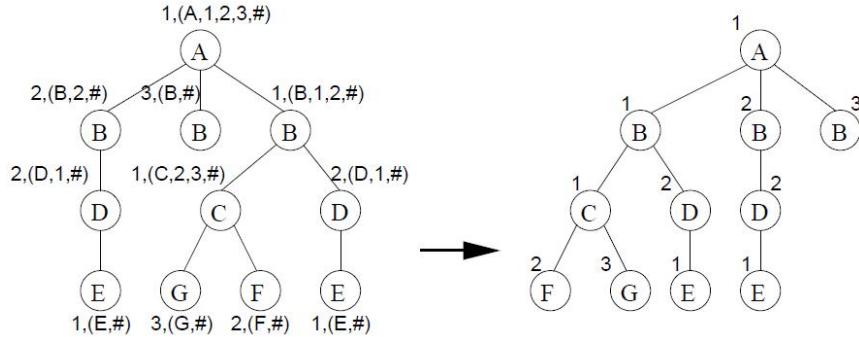


图 2.10 按照 AHU74 算法构造 DFCF 的过程

注：按照 AHU 算法构造 DFCF 的过程中，每个节点对应的元组的末端是可以不加 ‘#’ 的。

2.2.5.4 DFCS 总结

- (1) DFCF 中对符号大小关系的假设与 BFCF 一致，在实现的过程中都需注意 ASCII 码之间的真实大小关系；
- (2) 文章对于 DFCS 长度的上界的估计存在着与对 BFCS 长度上界的估计一样的问题，都计入了字符串码中并不包含的边的标号值。值得一提的是，[64]中提到的字符串编码方法也是基于深度搜索的，从孩子节点到父节点回溯时就会加一个字符‘-1’，与该算法比较类似。而[64]中对字符串长度的估计值为 $2m+1$ ，其中， m 为边的数目，这与我们的估计一致。

2.2.6 字符串长度为 $2n$ 的树结构编码方法

参考文献[61]中介绍了一种字符串码长度为 $2n$ 的树结构编码方法，[65]中展示了一个例子，如图 2.11 所示。

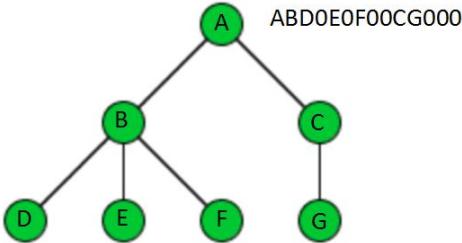


图 2.11 有根无序有标号树及其对应的字符串码

以 r 为根节点的树 T_r 对应的字符串码为 $S_{T_r} = l_r S_{T_{r_1}} S_{T_{r_2}} \dots S_{T_{r_k}} 0$, 其中, l_r 表示节点 r 对应的标号值, r_1, r_2, \dots, r_k 为节点 r 的孩子节点, $S_{T_{r_1}}, S_{T_{r_2}}, \dots, S_{T_{r_k}}$ 分别为以 r_1, r_2, \dots, r_k 为根的子树 $T_{r_1}, T_{r_2}, \dots, T_{r_k}$ 对应的字符串码, 且按字典序非降序排列。由于每个节点都需要用额外的两个字符表示, 故节点数为 n 的糖结构对应的字符串码的长度为 $2n$ 。

2.3 糖结构同构判断的编码方法

正如 2.1 节所述, 如果我们在对糖结构进行两两比较之前可以先把糖结构遍历一遍, 并且记录相应的哈希值, 则只需要对哈希值相同的糖结构进行两两比较, 从而对糖结构中节点的访问及比较操作进行有效的剪枝。哈希的冲突越低, 剪枝的效果就越好。如果我们能够设计一种一一映射的哈希方法, 将同构的糖结构映射到同一个字符串序列上, 则可以把糖结构的同构判定问题转化为字符串之间的冗余判定问题。为了实现一一映射, 我们基于[59-70]提出的不同的树结构编码方法把糖结构编码成不同的字符串, 这些字符串码的长度范围为 $2n \sim 4n$, 其中 n 为糖结构的节点数目。进一步, 我们在 KS99 这一无根无序无标号树的编码方法的基础上设计出了一系列编码方法, 基于这些编码方法, 我们不再需要对糖结构进行两两比较, 而是按照糖结构对应的编码值顺序比较, 下面对其中两种方法进行具体介绍。

2.3.1 双侧外包分层编码方法

为了使得人工解码更加方便, 我们在编码过程中让不同层次的外包有所区分, 从而使其可以直接体现出糖结构层次信息, 称之为双侧外包分层编码方法。下面对这一编码方法进行具体介绍。

我们以单个节点为单元而不是以两层糖结构为单元对糖结构进行编码, 且采用能够直接体现出糖结构层次信息的分层双侧外包, 从而使得糖结构的编码序列

便于人工解读。假设设根节点所在层为第 0 层。我们将五种单糖 Hex、HexNAc、NeuAc、NeuGc、dHex 对应的标号值分别设置为'1'、'2'、'3'、'4'、'5'。对于第 i 层的节点，将其对应的单糖的标号值与孩子节点的串接字符串（按照字典序非降序连接）连接，并将该连接好的字符串的最左端加入字符 L(i)，最右端加入字符 R(i)，其中 $L(i)=i+'A'$, $R(i)=i+'a'$ ，如此构成的字符串为该节点对应的编码值。图 2.12 展示了按照上述方法对糖结构进行编码的过程。

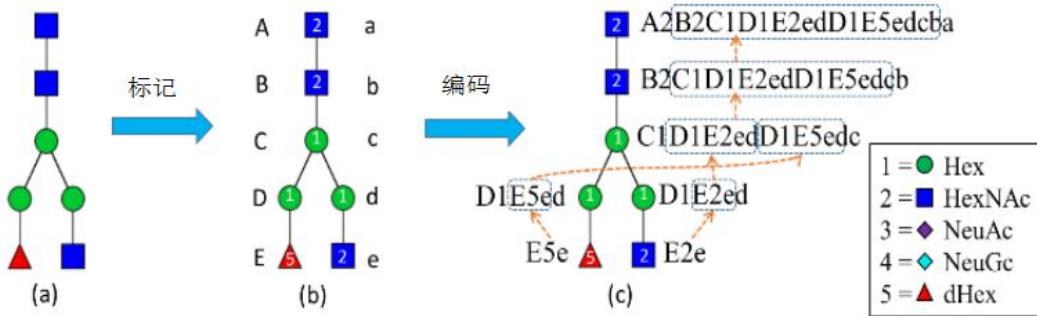


图 2.12 构造糖结构对应的字符串码的过程

(a) 给定的糖结构 (b)对该糖结构的单糖进行数值标记, 具体来说, 单糖类型为 Hex、HexNAc、NeuAc、NeuGc、dHex 的节点对应的标号值分别为 1,2,3,4,5; 根节点所在层 (第一层) 的所有节点的最左端的字符为'A', 最右端的字符为'a', 第二层的所有节点的最左端的字符为'B', 最右端的字符为'b', 依此类推 (c)每个节点对应的字符串码为该节点所在层对应的最左端字符+节点对应的标号值+节点的孩子节点的字符串码按照字典序非降序串接成的字符串+节点所在层对应的最右端字符 ('+' 表示字符串之间的串接); 比如, 对于第三层的节点来说, 该节点的两个孩子节点对应的字符串码按照字典序从小到大分别为"D1E2ed"和"D1E5ed", 则该节点对应的字符串码为 'C'+1+"D1E2ed"+"D1E5ed"+'c'="C1D1E2edD1E5edc"。由于每个节点都需要额外的三个字符表示, 故节点数为 n 的糖结构对应的字符串码的长度为 3n. 单糖类型及其表示符号分别为: HexNAc (蓝色方块), Hex (绿色圆形), dHex (红色三角形)。

在编码阶段, 我们从叶节点开始, 把糖结构中同一层的节点信息按照一定的正则顺序往上汇总直至根节点 (如图 2.12 所示), 保证同构的糖结构对应的的根节点的字符串相同。在解码阶段, 给定根节点对应的字符串, 我们从左到右遍历字符串, 如果当前字符为大写字母, 下一个字符一定是数字字符, 则在下一层加一个标号值为该数字字符的节点; 如果当前字符为小写字母, 则回溯到上一层, 从而唯一确定了正则化的糖结构 (如图 2.13 所示), 该正则化的糖结构对应于一类互为同构的糖结构, 这说明相同的字符串对应的糖结构一定互为同构。基于上述分析我们可以推知, 糖结构互为同构当且仅当这些糖结构的根节点对应的字符串码相等, 故称根节点对应的字符串码为该糖结构的字符串码。另外, 考虑到字符串之间的比较相当耗时, 我们把字符串映射为哥德尔码[71], 从而实现字符串到数字的一一映射。最终, 糖结构的同构判定问题就简化为了浮点数的冗余判定

问题。

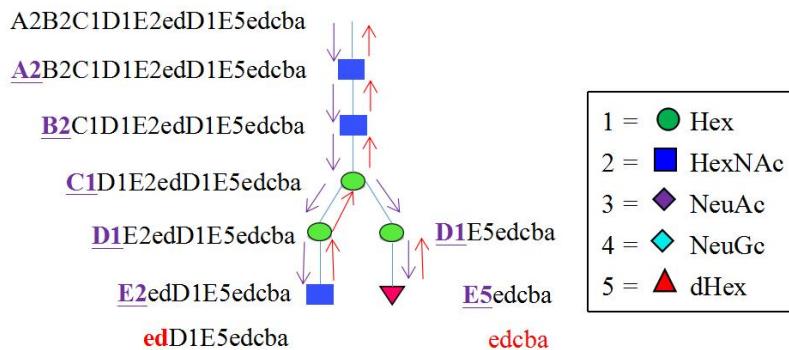


图 2.13 给定根节点对应的字符串，解析出该字符串对应的糖结构的过程示例

给定根节点对应的字符串 A2B2C1D1E2edD1E5edcba，我们从左到右遍历字符串，如果当前字符为大写字母，下一个字符一定是数字字符，则在下一层加一个标号值为该数字字符的节点；如果当前字符为小写字母，则回溯到上一层，从而推出正则糖结构。

需要注意的是，当前的编码方法采用大小写字母作为外包，只能表示层数不超过 26 的糖结构。考虑到 N-糖中单糖的种类不会超过 26 种，我们可以用字母表示单糖，数字表示层数，这样就可以避免糖结构的字符串编码受层数限制。另外，我们的糖结构编码方法可以扩展用于表示糖结构的连接位点信息。比如，图 2.14 所示的包含连接位点信息的糖结构（GlycomeDB 中 ID 值为 234）的字符串码为 A2(1-4#)B2(1-4#)C1(1-6\$)D1(1-6\$)D1(1-6\$)E1e(1-3\$)E1ed(1-3\$)D1dcba，其中，‘\$’ 表示 α ，‘#’ 表示 β 。

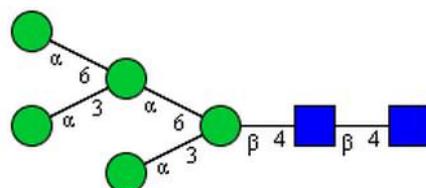


图 2.14 包含连接位点信息的糖结构

2.3.2 单侧外包围结合二层树编码方法

每个糖结构都可以看做是由多个二层糖结构拼接而成，这里以二层树为单位对糖结构的信息进行整合，称之为二层树哈希方法。假设糖链由 5 种单糖构成，并且每种单糖最多能与另外 4 个单糖相连（节点的出度至多为 4），则层数为 2 的树一共有 625 种（不包含同构冗余）。计算过程如下：

(1) 情形一：

树的根节点的出度为 1，共有 $C_5^1 * C_5^1 = 25$ 种。

(2) 情形二：

树的根节点的出度为 2，共有 $C_5^1 * (C_5^1 + C_5^2) = 75$ 种。

(3) 情形三：

树的根节点的出度为 3，共有 $C_5^1 * (C_5^1 + C_5^2 * C_2^1 + C_5^3) = 175$ 种。

(4) 情形四：

树的根节点的出度为 4，共有 $C_5^1 * (C_5^1 + C_5^2 * (1 + C_2^1) + C_5^3 * C_3^1) = 350$ 种。

由于每个糖结构都可以看成是由多个层数为 2 的树组合而成，我希望将一棵树拆成多个层数为 2 的树，并建立层数为 2 的树与自然数之间的一一映射。具体方法为：

1) 设根节点所在层为第 0 层，分别用 1, 2, 3, 4, 5 表示 Hex, HexNAc, NeuAc, NeuGc, dHex 这五种单糖，称之为单糖的哈希值；

2) 将这 625 棵层数为 2 的树的哈希值与 1~625 个自然数一一对应，从而缩小二层树的哈希范围。记经过该映射后二层树 T 的初始哈希值为 ValueSub(T)。由于最后一层对应的是一层树，故需要对单节点子树进行哈希。我们分别把单糖类型为 Hex, HexNAc, NeuAc, NeuGc, dHex 的单节点子树哈希为 626, 627, 628, 629, 630。

每个糖结构都可以看成是由多个层数为 2 的糖结构组合而成，反过来，我们可以把一个糖结构拆成多个层数为 2 的糖结构，并建立层数为 2 的糖结构与数值之间的一一映射。假设糖结构由 5 种单糖构成，并且每种单糖最多能与另外 4 个单糖相连（节点的出度至多为 4），则层数为 2 的互不同构的糖结构一共有 625 种。结合把糖结构分解成多个两层糖结构的想法，我们以两层糖结构为单元对糖结构进行编码。互不同构的两层糖结构一共有 625 种，其标号值分别对应于 1~625 之间的数字。**图 2.16** 所示的糖结构可以拆分为四种互不同构的两层糖结构，其标号值如**图 2.15** 所示：

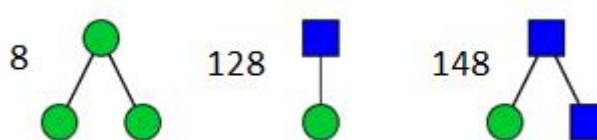


图 2.15 糖结构包含的两层糖结构及其对应的标号值

需要注意的是，标号值中的每一位由 1~9 之间的数字构成，为了方便区分标号值与分层标识，我们不再选择数字作为单侧外包，而是选择“（”作为单侧外包。糖结构的编码过程如**图 2.16** 所示：

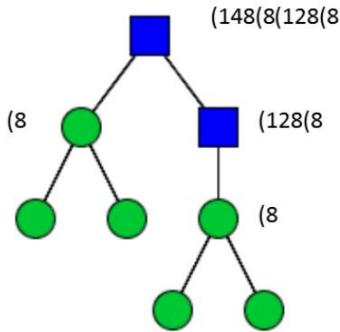


图 2.16 单侧外包结合二层树的编码过程

该编码方法利用单侧外包以及二层树的标号值很好地控制了糖结构对应的编码序列的长度, 然而, 这样的字符串码并不能直接地体现出糖结构的层次信息, 且需要掌握二层树与其标号值的对应信息, 这些无疑增大了由字符串码推出糖结构的难度。鉴于双侧外包分层编码方法兼具适合计算机存储解析及人工解读的性质, 双侧外包分层编码方法为我们最终采用的糖结构编码方法。

2.3.3 线性正则编码方法的总结

早在 2010 年, 就有工作[46]提出了糖结构的线性正则编码方法。[46]依次设计了四种针对无根糖结构和三种针对有根糖结构的表示方法, 最后选择了一种把还原端作为根节点, 根据连接位点信息来确定兄弟节点排列顺序的编码方法, 称之为 LINUCS 格式。LINUCS 格式将连接信息作为额外的规则, 如果兄弟节点的连接信息相同而对应的单糖不同, 则按照这种编码方法可能会生成多个字符串, 还需进一步从多个字符串中选择字典序最小的字符串作为编码值。GlycoCT 格式[48]为了确定分支的顺序, 增加了六个额外的规则。然而, 这些方法都是从根节点往叶节点遍历糖结构, 这就要求增加额外的比较规则来确定遍历兄弟节点的顺序, 而增加额外的规则则可能导致生成多余的字符串。我们设计的编码方法则是从叶子节点往根节点遍历, 这样确定同一层节点的正则顺序时, 就只需要比较节点对应的编码值, 而不需要额外增加规则, 也不需要生成额外的字符串。

糖结构的线性正则码不仅可以方便我们判定糖结构之间的同构冗余, 还可以在其它很多方面为我们带来便利。总体来说, 糖结构的双侧外包分层编码方法具备如下优点: 1) 该线性表示把同构的糖结构编码为相同的字符串, 从而为糖结构的同构冗余判定提供了一种简单有效的方法; 2) 我们设计的编码方法直接体现了糖结构的层次信息, 不仅便于计算机存储及解析, 还具备易于人工解读的性

质，即给定糖结构的字符串表示，人们可以很轻松地推出糖结构的糖组成及不同单糖之间的连接方式；3) 通过把分支的糖结构线性化，我们可以更加高效地查找糖结构；4) 糖结构的线性化表示方法可以使得糖结构库之间的比较与合并如同肽段库一样方便。

另一方面，尽管越来越多的糖结构库被构建，然而大部分糖库如同“孤岛”一般与其它糖库之间几乎没有任何联系。实际上，很多糖结构库有很多共有的部分，但是由于存储格式不一致，导致糖库之间的整合相当困难。尽管曾经选过 GLYDE-II 作为标准存储格式，但是大部分糖库还是在用它们自己的存储格式 [41]。没有标准的存储格式，不仅给糖库的比较与使用带来了很大的不便，还可能导致存储的信息不准确，小到简单的排版错误，大到结构表示不一致[44]。希望我们提出的编码方法能够有助于糖结构存储格式的统一。

第三章 基于邻接表的理论 N-糖结构库构建

随着当前质谱技术的发展及公开糖结构库的构建,一些依赖于数据库搜索的糖肽鉴定软件[19, 53-58, 73]也基于特定的实验需求构造了不同的糖库。GRIP[54]构造了一个由人类血清中三个最大可能的 N-糖结构的 Y 离子组成的 N-糖库。GlycoMaster DB[19]和 ArMone2.0[55]提取出 GlycomeDB 中的 N-糖作为 N-糖库。GlycoPeptideSearch[56]从 GlycomeDB 中提取出了属于 GlycO 的人类的糖。GPQuest[57]构建的糖库包含一些人类血清或血浆 N-糖库以及他们实验室从不同的人类样本中鉴定到的 N-糖。I-GPA[58]用 GPA-DB-Builder 自动构造了一个 N-糖肽数据库。pGlyco[53]通过把人类血清中三个最大通用结构的包含五糖核心的子结构库和 GlycomeDB 中的 N-糖库合并, 构造了一个 N-糖库。

然而, 这些糖库都是不完整的[36, 39, 45]。基于糖库搜索鉴定谱图时, 如果糖库不全, 正确的糖结构没有包含在糖库中, 则可能把当前打分值最好的鉴定结果看作该谱图对应的糖肽, 导致错误鉴定[74]。因此, 构造一个全面的糖库至关重要。为了得到一个更加全面的糖库, 我们基于糖结构的邻接表存储格式枚举了所有包含五糖核心的糖结构, 称之为枚举库。同时, 我们还生成了糖结构的所有包含糖结构的根节点的子结构, 从而为 N-糖肽鉴定提供理论谱图库以便于匹配打分。

3.1 理论 N-糖结构库的枚举算法

我们从五糖核心开始, 由节点数目为 n 的糖集合枚举出节点数目为 n+1 的糖集合 (图 3.1 展示了一个具体的例子)。通常, 糖结构由 Hex、HexNAc、NeuAc、NeuGc、dHex 这五种单糖构成。考虑到 NeuAc 与 NeuGc 有相似的理化性质, 我们枚举的糖结构由除了 NeuGc 之外的四种单糖构成。我们用 M 表示这四种单糖的集合, 即 $M=\{\text{Hex}, \text{HexNAc}, \text{NeuAc}, \text{dHex}\}$, $t[v]$ 表示节点 v 对应的单糖类型。具体的枚举方法如算法 3.1 所述。

算法 3.1 糖结构枚举算法

输入：节点数目为 n 的糖结构集合，记为 $\text{Set}(n)=\{(V(n), E(n))\}$

输出：节点数目为 $n+1$ 的糖结构集合，记为 $\text{Set}(n+1)=\{(V(n+1), E(n+1))\}$

Procedure Enumerate_Glycan($\text{Set}(n)$)

for each glycan $G(n) \in \text{Set}(n)$

for each vertex $u \in G(n)$

for each vertex v s.t. $t[v] \in M$ #依次枚举四种单糖作为新节点的单糖类型

$G(n+1)=(V(n) \cup \{v\}, E(n) \cup \{(u, v)\})$

$\text{Set}(n+1)=\text{Set}(n+1) \cup G(n+1)$ #将这个糖结构加入到 $\text{Set}(n+1)$ 中

end procedure

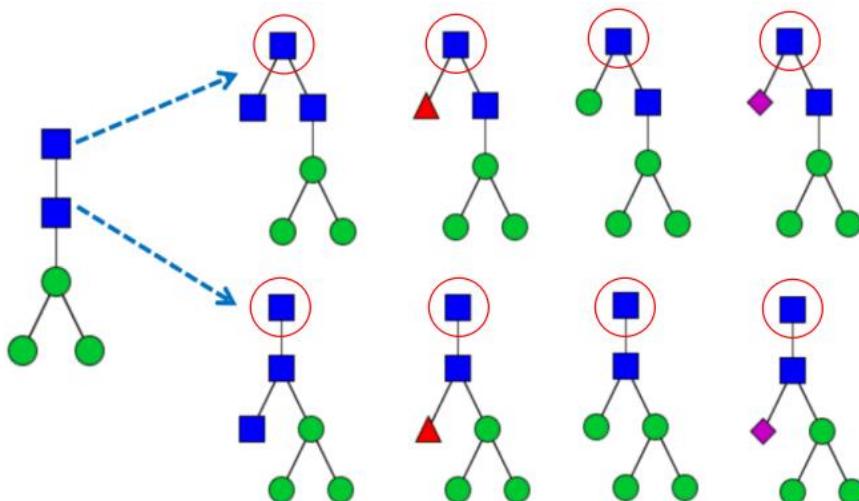


图 3.1 由一个节点数目为 5 的糖结构枚举出节点数目为 6 的糖结构

(a) 节点数目为 5 的糖结构 (b) 将(a)所示的糖结构中的一个节点分别与四种单糖连一条边构成节点数目为 6 的糖结构。其中，单糖类型及其表示符号分别为：HexNAc (蓝色方块)，Hex (绿色圆形)，NeuAc (紫色菱形)，dHex (红色三角形)。

在枚举的过程中，可能会生成一些同构的糖结构，我们需要进一步去掉糖结构中的同构冗余。通过把树型的糖结构按照其正则顺序编码为线性的字符串，我们将糖结构的去冗余转化为线性字符串之间的去冗余，从而极大地方便了糖结构的枚举。

3.2 败者树枚举更多节点数目的糖结构

我们在实现糖结构枚举的过程中，把节点相同数目的糖结构都存储在内存中，以便对所有相同节点数目的糖结构去镜像冗余，但是这种方法可能会由于内存不足而导致程序崩溃。比如，我们在 Intel(R)Core(TM) i5-4570 处理器，16GB 内存，windows 7 64 位操作系统下枚举 11 个节点的糖集合时内存就不足以存储所有的糖结构了。为了解决这一问题，我们尝试了更改存储类型、数据结构等方法，并在内存更大的服务器上运行程序。

枚举糖结构的过程中，对于节点数目为 n 的糖结构，我们依次将该糖结构的每个节点与每种单糖连一条边，构成节点数目为 $n+1$ 的糖结构。不妨设 a_n 为节点数目为 n 的糖结构数目 ($n \geq 5$)， c 为单糖的种类，则糖结构数目满足递推关系式 $a_{n+1} = cna_n$ ，又由于 $a_5 = 1$ ，故 $a_n = \frac{1}{4!}c^{n-5}(n-1)!$ 。由该递推公式可以直观地看出，糖集合的大小随着节点数目呈爆炸式增长，因此以上尝试只能在一定程度上缓解内存不足的问题，却不能彻底摆脱内存的约束。

最终，我们采用败者树归并的方法（**算法 3.2**）避免了糖库枚举受内存的限制。首先，我们从磁盘文件中读取节点数目为 n 的糖结构信息，如果糖结构数目超过一定阈值，则分批读入。然后，以当前读入内存中的糖结构为基础，生成相应的节点数目为 $n+1$ 的糖。当内存中糖结构的数目达到一定阈值之后，筛选出合理且互不同构的糖结构，并把筛选出的糖结构信息按照哥德尔码非降序输出到临时文件中。如果临时文件的数目大于 1，则用败者树归并所有的临时文件，并对哥德尔码相同的糖结构去冗余，最终把所有节点数目为 $n+1$ 的互不同构的糖结构输出到磁盘文件中。

对糖结构编码是枚举出更多节点数目糖集合的关键。我们按照 **2.3.1** 节介绍的双侧外包分层编码方法将糖结构编码成字符串，并计算字符串的哥德尔码，然后按照哥德尔码非降序读入糖结构，对于哥德尔码相同的糖结构集合，只保留其中一个，从而实现临时文件之间的去冗余。具体的归并方法如**算法 3.2** 所述。

算法 3.2 败者树归并算法

输入: n 个节点的糖结构; 阈值 A: 内存中存储的 n 个节点的读入的糖结构的最大数目

阈值 B: 内存中存储的 n+1 节点的枚举的糖结构的最大数目

输出: n+1 个节点的糖结构

- 1) 将磁盘文件中存储的还未导入到内存中的 n 个节点的糖结构导入到内存中。
如果还未导入到内存中的糖结构的数目超过阈值 A, 则只读入 A 个糖结构;
 - 2) 由 n 个节点的糖结构枚举出 n+1 个节点的糖结构;
 - 3) 当枚举的 n+1 个糖结构的数目超过阈值 B 之后, 筛选出合理且非冗余的糖结构, 并将这些糖结构按照哥德尔码非降序输出到临时文件中;
 - 4) 如果临时文件的数目为 1, 则临时文件中存储了所有节点数目为 n+1 的合理且非冗余的糖结构, 算法停止; 否则, 用败者树将所有临时文件中的糖结构按照哥德尔码从小到大依次读入, 对于哥德尔码相等的糖结构, 去掉这些糖结构中的冗余结构, 最终将非冗余的糖结构输出到最终文件中;
 - 5) 如果所有的 n 个节点的糖结构都被导入内存中了, 则算法停止; 否则, 返回到步骤 1)。
-

3.3 合理糖结构筛选策略

由 3.1 节的分析可知, 枚举过程中需要生成的糖结构的数目随着节点数目呈爆炸式增长。另一方面, 全枚举的糖库虽然包含了所有可能的糖结构, 但并不是所有的理论糖结构都是合理的, 这将不可避免地引入很多假阳。为了提高糖库的质量并且有效地控制糖结构库的规模, 我们根据 GlycomeDB 中 N-糖结构的不同单糖的分支数目的统计信息, 设置了如表 3.1 所示的单糖分支限制。如果糖结构的某类单糖的分支数目超过了既定的阈值, 则认为这个糖结构是不合理的。

表 3.1 单糖的分支数目限制

单糖类型	Hex	HexNAc	NeuAc	dHex
最大分支数目	3	3	1	1

进一步，我们在枚举的过程中按照一定的生物规则（GP Finder 使用的规则 [75]）筛选掉了不合理的糖结构，即对于糖结构的每个 Y 离子，其包含的不同种类的单糖的数目需要满足 GP Finder 的规则，从而得到了一个更加全面合理的糖结构库。具体判断条件如图 3.2 所示。

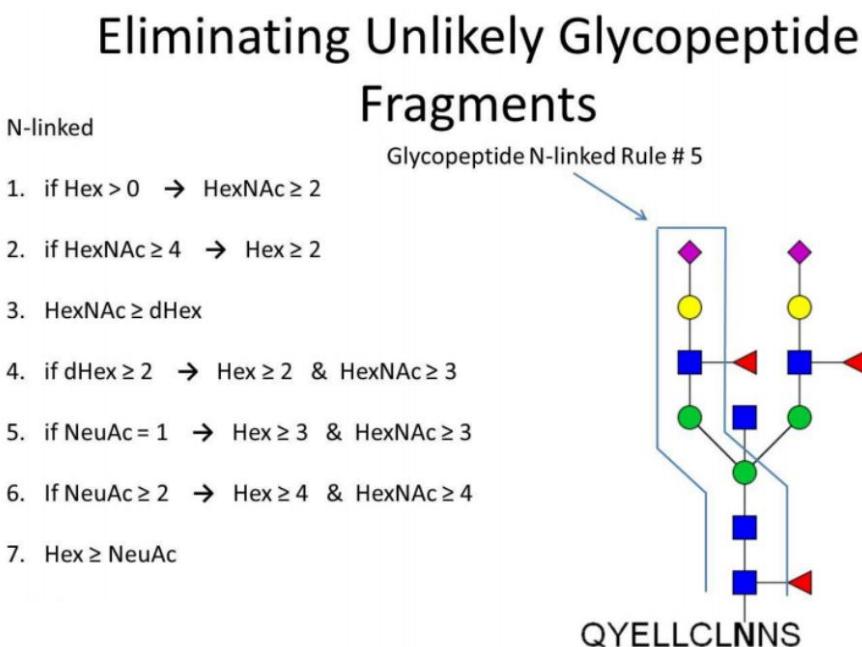


图 3.2 GP Finder 的过滤规则

如果糖结构的任一 Y 离子不满足图中所示规则，则认为这个糖结构是不合理的。

在枚举过程中，我们通过对不符合 GP Finder 规则的糖结构进行剪枝，有效地提高了糖结构的枚举效率。假设糖结构 A 是糖结构 B 的子结构，则 A 的 Y 离子集合包含于 B 的 Y 离子集合，结合合理糖结构的判断准则可以推得如下性质：

性质 1 如果 n 个节点的糖结构不合理，那么由该糖结构扩展的 n+1 个节点的糖结构也一定不合理。

利用**性质 1**，我们在枚举糖结构的时候，先从 n 个节点的互不同构的糖结构中筛选出合理的糖结构，然后再以此为基础构造节点数目为 n+1 的糖结构。合理的糖结构的子结构一定合理，反之，任一子结构不合理，则该糖结构一定不合理。基于此，我们在枚举的过程中，不仅去掉了同构冗余的糖结构，还去掉了不合理的糖结构，从而对糖集合进行了剪枝。为了说明剪枝的有效性，我们分别基于互

不同构的糖结构和互不同构且合理的糖结构枚举了节点数目为 6~10 的糖结构集合，并记录了筛选前后糖结构的数目，如表 3.2、3.3 所示。

表 3.2 不合理的糖结构剪枝前的枚举结果

	糖结构的节点数目				
	6	7	8	9	10
枚举生成的糖结构数目	20	368	5,040	59,048	639,820
互不同构糖数目	16	196	2,082	20,631	195,672

表 3.3 不合理的糖结构剪枝后的枚举结果

	糖结构的节点数目				
	6	7	8	9	10
枚举生成的糖结构数目	20	252	1,904	12,052	76,828
互不同构且合理糖数目	11	73	407	2,314	13,844

以节点数目为 10 的糖集合为例，从表 3.2、3.3 可以看出，如果没有筛选出合理的糖结构，则由互不同构的 20,631 个节点数目为 9 的糖结构总共枚举出 639,820 个节点数目为 10 的糖结构；而筛选出合理的糖结构之后，由互不同构且合理的 2,314 个节点数目为 9 的糖结构只需枚举出 76,828 个节点数目为 10 的糖结构，需要枚举的糖结构的数目为筛选前的 12%，有效地节省了内存并提高了效率。

3.4 糖结构库的扩展

最初的合并库及枚举库中，构成糖结构的单糖类型为除了 NeuGc 之外的四种单糖，即 Hex、HexNAc、dHex 与 NeuAc。NeuGc 只比 NeuAc 多一个 OH 基团，它在脊椎动物体内广泛存在，但是极少存在于人类的糖肽中，除非组织发生了癌变[74]。鉴于 NeuAc 与 NeuGc 有相似的理化性质，我们先生成只由 Hex、HexNAc、dHex、NeuAc 这四种单糖构成的糖结构，然后把糖结构中的 NeuAc 替换为 NeuGc，从而得到由五种单糖构成的糖结构。对于每个糖结构，不妨设糖结构中 NeuAc 的数目为 m，如果 m 不为 0，则选择 p 个 NeuAc，将这 p 个 NeuAc 替换为 NeuGc，其中， $0 \leq p \leq m$ 。对于有 m 个 NeuAc 的糖结构，可以扩展生成

$C_m^0 + C_m^1 + \dots + C_m^m = 2^m$ 个包含 NeuGc 的糖结构。进一步，对于扩展生成的有相同 NeuGc 数目的糖结构，我们去掉了互为同构的冗余结构。图 3.3 给出了糖结构扩展的一个具体例子，**算法 3.3** 描述了具体的扩展方法。

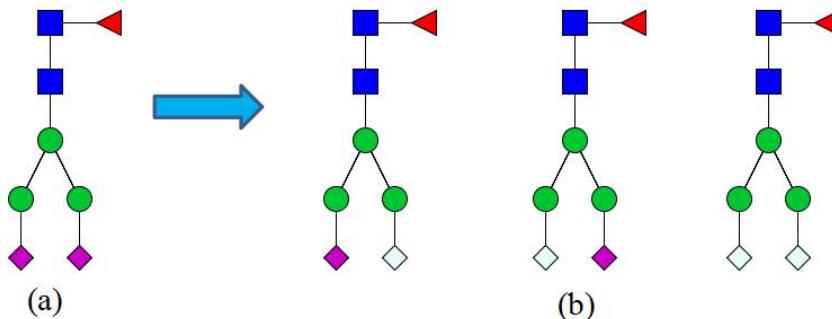


图 3.3 糖结构扩展示例

(a) 糖结构 (b) 将(a)所示的糖结构中的 NeuAc 部分地替换为 NeuGc 得到的糖结构集合。得到扩展后的糖结构集合之后，我们再进一步去掉这些糖结构中的冗余结构。其中，单糖类型及其表示符号分别为： HexNAc (蓝色方块)， Hex (绿色圆形)， NeuAc (紫色菱形)， NeuGc(青色菱形)， dHex (红色三角形)。

算法 3.3 糖结构的扩展算法

输入： 糖结构 $G=(V, E)$ ； 节点编号 v ， 初始化为 $v=1$ ； 假设糖结构 G 的节点编号为 $1, 2, \dots, |V|$ 。

输出： 扩展后的糖结构集合 \mathcal{H}

其它符号说明：

$t[v]$ 表示节点 v 对应的单糖类型

Procedure ExtendActoGc(G, v)

```

if  $v == |V|$ 
     $\mathcal{H} = \mathcal{H} \cup G$ 
    return                                #遍历完糖结构的所有节点后，算法停止
ExtendActoGc( $G, v+1$ )
if  $t[v] == \text{NeuAc}$                   #如果编号为  $v$  的节点对应的单糖类型为 NeuAc
     $t[v] = \text{NeuGc}$                       #将该节点对应的单糖类型改为 NeuGc
    ExtendActoGc( $G, v+1$ )
end procedure

```

3.5 基于邻接表生成糖结构的子结构的组成

为了在基于糖库搜索的谱图鉴定中方便地进行谱峰匹配, 对于每个非同构冗余的糖结构, 我们生成了该结构的所有非结构冗余的子结构, 进一步得到了该结构的所有非组成冗余的子结构的组成。假设某个糖结构的节点数目为 m , 对于每个节点数目为 $n(1 \leq n \leq m)$ 的子结构, 增加一个与它的节点相邻的新节点, 从而构造节点数目为 $n+1$ 的子结构, 直到生成的子结构的节点数目等于这个糖结构的节点数目 m 为止。**算法 3.4** 具体描述了生成糖结构的子结构的方法。

算法 3.4 糖结构的子结构生成算法

输入: 节点数目为 N 的糖结构 $G = (V, E)$

输出: 糖结构 G 的互不同构的子结构集合, 记为 $\mathcal{F} = \{\mathbf{F}_n | 1 \leq n \leq N\}$, 其中 \mathbf{F}_n 为节点数目为 n 的子结构集合

其它符号说明:

$R(G)$ 表示糖结构 G 的根节点对应的单糖类型;

$G.\text{Adj}[v]$ 表示节点 v 在 G 中的邻接节点集合;

y_n 表示节点数目为 n 的子结构;

$\mathbf{Y}_n = \{(V_n, E_n)\}$ 表示 G 的节点数目为 n 的子结构集合(可能包含同构的糖结构);

$ID(\mathbf{Y}_n)$ 表示节点数目为 n 的所有子结构的节点编号的集合;

$ID(G)$ 表示糖结构 G 的节点编号。

Procedure Generate_SubGlycans(G)

初始化 $\mathbf{Y}_1 = \{(\{R(G)\}, \{\emptyset\})\}$

for $n = 1, \dots, N - 1$

for 子结构 $y_n = (V_n, E_n) \in \mathbf{Y}_n$

for 节点 $u \in y_n$

for 节点 $v \in G.\text{Adj}[u]$ #对于 u 的每个邻接节点 v

$y_{n+1} = (V(n) \cup \{v\}, E(n) \cup \{(u, v)\})$

if $ID(y_{n+1}) \notin ID(\mathbf{Y}_{n+1})$ #如果新构造的糖结构的节点编号不冗余

$\mathbf{Y}_{n+1} = \mathbf{Y}_{n+1} \cup y_{n+1}$ #把新构造的糖结构加到 \mathbf{Y}_{n+1} 中

if y_{n+1} 与 \mathbf{F}_{n+1} 中的糖结构均不同构

$\mathbf{F}_{n+1} = \mathbf{F}_{n+1} \cup y_{n+1}$

$\mathcal{F} = \mathcal{F} \cup \mathbf{F}_{n+1}$

end procedure

在生成糖结构的子结构的过程中，对于节点编号完全相同的子结构，我们只保留其中一个。但是，对于同构的子结构，只能在所有的子结构都生成之后才能去掉这些糖结构中的同构冗余，图 3.4 展示了一个例子来说明。

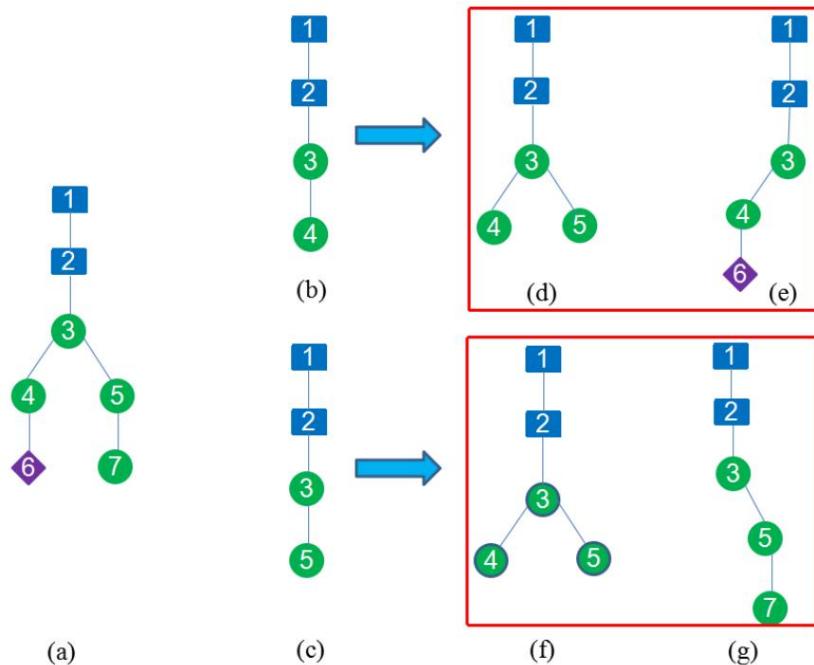


图 3.4 糖结构的部分子结构生成示例

(a)一个节点数目为 7 的糖结构，记为糖结构 A。(b)~(c)为该糖结构的节点数目为 4 的子糖集合，分别记为 G_1 和 G_2 。可以看出 G_1 和 G_2 互为同构，但是，这两个子糖的节点编号不同。 G_1 的节点编号为 {1,2,3,4}，而 G_2 的节点编号为 {1,2,3,5}，且在糖结构 A 中，与编号为 4 的节点相邻的节点的单糖类型为 NeuGc，而与编号为 5 的节点相邻的节点的单糖类型为 Hex。故由这两个子糖加上一个新节点构成的子糖集合也不一样，分别为 (d)、(e) 和 (f)、(g)。(d)~(g) 为由节点数目为 4 的子糖加上一个新节点构成的节点数目为 5 的子糖集合。(d) 与 (f) 对应的子糖节点编号完全相同，因此，可以去掉其中一个。

通过进一步分析，我们发现糖结构的子结构集合满足如下性质：

性质 2 如果糖结构 A 是糖结构 B 的子结构，那么糖结构 A 的子结构集合是糖结构 B 的子结构集合的子集。

利用**性质 2**，我们可以基于子结构的子结构来构造母结构的子结构，类似于动态规划的思想，具体方法为：如果糖结构 A 是由糖结构 B 的节点 m 增加一条边 e 构成，则对于糖结构 B 中的所有子结构，如果某个子结构恰好包含节点 m，则将该子结构加上边 e，构成新的子结构，如此得到糖结构 A 的子结构集合。这种生成子结构的方法利用动态规划有效地避免了一些冗余计算，但是需要更多的

空间来存储糖结构的子结构。

3.6 糖结构的输出信息

在设计出糖结构的线性正则编码方法之前, 我们用两个文件分别存储了糖结构及其 Y 离子的组成等信息以及糖结构的邻接表信息。根据糖结构中每个节点对应的单糖类型及其拓扑关系, 我们可以计算出糖组成。糖组成由五维向量表示, 依次为糖结构中 Hex、HexNAc、NeuAc、NeuGc、dHex 这五种单糖的数目。输出文件 1 中给出了糖库的 ID, 糖结构的 ID, 糖结构的来源及组成, 糖结构对应的所有 Y 离子的质量及组成。其中, 糖结构的 ID 是按照质量大小赋值的, 糖结构的质量越小, ID 越小。

糖结构及其 Y 离子的组成可以用于衡量实验谱图与理论谱图的匹配程度, 却不足以确定糖的结构, 因此, 我们给出了输出文件 2, 该文件中记录了糖结构的根节点、每类单糖对应的节点标号以及节点之间的拓扑关系, 结合这三种信息就可以确定出糖的具体结构。除此之外, 输出文件 2 中记录了糖结构的 ID, 且该 ID 与输出文件 1 中糖结构的 ID 是一致的, 以便于根据糖结构的 ID 找到两个文件中相对应的糖结构信息。

根据这两个输出文件可以方便地比较糖库文件的异同: 先根据输出文件 1 比较糖库文件的糖组成, 如果组成不同, 则对应的糖结构一定不同; 如果组成相同, 则可以根据 ID 值在输出文件 2 中查找具体的结构并对比分析。

综上, 输出文件满足如下功能: (1) 可以根据糖结构 ID 查询糖结构、糖组成及其质量、糖的所有子结构组成及其质量; (2) 给定糖结构或者糖组成, 可以查询糖库中是否存在对应的糖; (3) 给定子结构, 可以查询其对应的所有母结构及其质量。

设计出糖结构编码方法之后, 我们把树形结构映射为了线性的字符串码, 并把糖结构的线性字符串码输出到文件中。对不同的糖库进行比较时, 与传统的邻接表存储方式相比, 我们不再需要分别比较糖结构中每个节点对应的单糖类型及其拓扑信息, 只需比较糖库中的字符串码, 使得糖库之间的比较与合并如同肽段库一样方便。

第四章 基于糖结构线性正则表达式的 N-糖结构库构建

在第二章中我们介绍了糖结构的线性正则编码方法, 该编码方法旨在为多个糖结构的同构判定问题提供一个理论上优美且应用上高效的判断方法, 把同构的糖结构编码为相同的字符串, 从而把糖结构的同构判定问题转变为了字符串的冗余判定问题。对于第三章中介绍的构建理论糖结构库的方法, 我们首先将磁盘文件中按照邻接表格式表示的糖结构的拓扑信息存储在计算机中, 然后把树型的糖结构映射为线性正则字符串码, 最后把新枚举的糖结构按照邻接表格式输出到磁盘文件中。一个自然又大胆的想法就是, 如果我们直接基于糖结构的线性正则字符串码来枚举糖结构, 那么我们一方面可以节省存储糖结构的拓扑信息的空间, 另一方面可以省去糖结构的拓扑结构转化为线性正则字符串码的操作。如此以来, 糖结构库构建的时空效率都能够得到优化。在按照这一思路实现糖结构库构建的过程中, 我们还意外地发现了一些可行的剪枝策略, 这些剪枝策略进一步提高了糖结构库构建的时空效率。下面我们将依次介绍根据糖结构的线性正则字符串码实现糖结构的枚举及子结构的生成算法。

4.1 基于糖结构线性正则表达式的枚举方法

通过文献调研, 我们发现了一个用于对无根无序无标号树进行线性正则编码的算法, 称之为 KS99 算法[62]。为了使得人工解码更加方便, 我们在 KS99 算法的基础上设计了一种新的糖结构编码方法, 让不同层次的外包有所区分, 具体的编码过程如 2.3.1 节所述。

糖结构的线性正则表达式除了可以提供一种理论优美且应用高效的糖结构同构判定方法, 还具备如下优点: 1) 糖结构线性正则表示方法可以便于高效地查找糖结构, 使得糖结构库之间的比较与合并如同肽段库一样方便; 2) 直接体现了糖结构的层次信息, 不仅便于计算机存储及解析, 还具备易于人工解读的性质, 即给定糖结构的字符串表示, 人们可以很轻松地推出糖结构的糖组成及不同单糖之间的连接方式; 3) 每一层用不同的字符外包就不需要记录以每个节点为根的子树对应的正则字符串, 方便地找到一个节点的祖先节点, 从而便于实现直接根据糖结构的字符串码枚举新的糖结构。

有了糖结构的字符串码之后, 随之而来的一个问题就是, 我们是否能够直接

根据糖结构由上述编码方法生成的字符串码来枚举出新的糖结构呢？根据糖结构对应的字符串码实现糖结构的枚举可以分解成如下五个小问题，

1. 如何设计正则顺序能够在只存储根节点对应的字符串码（节省空间）且尽量减少字符串码的遍历操作（节省时间）的同时简化算法的设计呢？
2. 糖结构中如何加入新的节点能尽量减少冗余枚举呢？大致可以分为哪几类插入操作呢？
3. 加入新的节点之后，如果破坏了正则形式，如何处理呢？
4. 如何根据糖结构对应的字符串计算出糖结构中每个节点的分支数目呢？
5. 对于有重复的字符串片段，应该如何处理呢？处理的同时要保证之前的枚举算法、修改正则顺序的算法以及计算节点的分支数目的算法依然正确。

下面依次回答上述五个问题。

4.1.1 正则顺序的设计

为了算法描述的方便，我们简称含根子结构为子结构，将单糖类型 Hex、HexNec、NeuAc、NeuGc 及 dHex 分别标号为 1、2、3、4、5。本文中介绍糖结构枚举的过程中只考虑除了 NeuGc 外的四种单糖，算法描述过程中提到的 label 范围为[1, 5]实际上不包含 label 值 4，即 NeuGc。

如果兄弟节点之间按照其对应的字符串码的字典序非降序排列，则在图 4.1 (a) 所示的糖结构的根节点上加上单糖类型为 Hex 和单糖类型为 NeuAc 的节点，得到的糖结构如图 4.1(b) 所示。

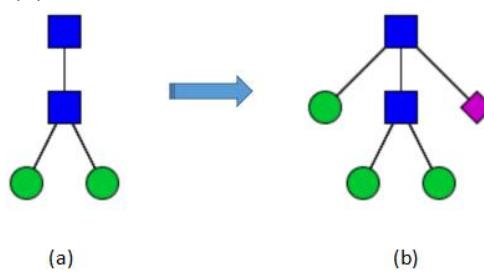


图 4.1 正则顺序为字典序的糖结构枚举示例

(a) 中所示糖结构的根节点上加上单糖类型为 Hex 和 NeuAc 的节点得到糖结构(b)，其中，兄弟节点之间按照字典序排序

如果我们能够设计出一种正则顺序，使得非叶子节点对应的子字符串一定大于叶子节点对应的字符串，那么我们可以有效地将叶子节点与非叶子节点分开，即糖结构中任一层的任一叶子节点一定在非叶子节点的左边，从而有助于简化字符串的遍历操作及算法的设计。如此我们在图 4.1 (a) 所示的糖结构的根节点上加

上单糖类型为 Hex 和单糖类型为 NeuAc 的节点之后得到的糖结构便如图 4.2(b) 所示。

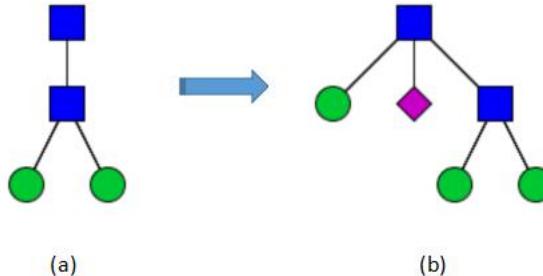


图 4.2 希望设计一种正则顺序能很好地区分开叶子节点和非叶子节点

(a) 中所示糖结构的根节点上加上单糖类型为 Hex 和 NeuAc 的节点得到糖结构(b), 其中, 兄弟节点之间按照专门设计的正则顺序排序

基于上述分析, 我们希望设计一种正则顺序, 满足如下要求:

- 1) 水平方向: 糖结构越宽, 则对应的字符串越大;
- 2) 垂直方向: 糖结构越长, 则对应的字符串越大。

为了达到上述要求, 我们不再按照字典序比较字符串, 而是按照数字之间的比较方法来比较字符串的大小。首先, 比较字符串的长度, 长度越长, 则字符串越大; 如果字符串的长度相同, 再从串首开始依次比较每个字符的大小。

我们在枚举的过程中, 会选定一个节点, 然后将新的节点与这个节点相连 (即将新的节点连在该节点的下一层)。实际上, 如果选定的这个节点为非叶子节点, 那么加入新的节点之后, 以该节点为根的子结构会变宽, 如图 4.3(a) 所示; 如果选定的这个节点为叶子节点, 那么加入新的节点之后, 以该节点为根的子结构会变长, 如图 4.3(b) 所示。

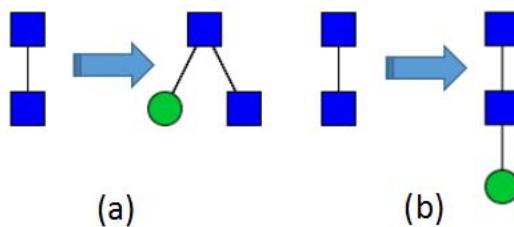


图 4.3 新的正则顺序下糖结构枚举示例

(a) 在糖结构的非叶子节点上加一个新的节点 (b) 在糖结构的叶子节点上加一个新的节点

按照这种正则顺序, 使得加入新的节点之后, 新节点连接的节点对应的字符串不会变小, 则我们在调整正则顺序的时候, 只需要比较该节点对应的字符串与其右边的兄弟节点对应的字符串的大小, 而不需要比较该节点对应的字符串与其左边的兄弟节点对应的字符串的大小。此外, 对于孩子节点中没有叶子节点的节点, 我们将新的节点与该节点相连时, 可以直接确定新的节点在下一层的节点中

的位置为最左端，无论新的节点对应哪种单糖类型。因为以叶子节点为根的子字符串的长度一定小于以非叶子节点为根的子字符串的长度，并且以叶子节点为根的子字符串的宽度一定不超过以非叶子节点为根的子字符串的宽度，而按照我们设计的正则顺序，字符串的长度越小对应的字符串一定越小，且字符串的宽度越小对应的字符串一定越小，从而以叶子节点为根的子字符串一定小于以非叶子节点为根的子字符串。

4.1.2 枚举方法的设计

为了方便描述算法，记节点 v 对应的单糖标号为 $\text{Label}(v)$ ，则基于糖结构的字符串码枚举糖结构的算法如下：

算法 4.1 基于糖结构的字符串码的枚举算法

输入： 节点数目为 N 的糖结构对应的字符串码

输出： 节点数目为 $N+1$ 的糖结构对应的字符串码

从左到右依次遍历糖结构的字符串码，如果当前字符对应的节点有孩子节点，则

如果当前字符对应的节点的孩子节点中有 m 个叶子节点，依次为 v_1, v_2, \dots, v_m ，

则

在 v_m 的最右端插入新的兄弟节点，其 Label 范围为 $[\text{Label}(v_m), 5]$ ，并调整每个新生成的糖结构的正则顺序；

以 v_m 为父节点，插入新的 Label 范围为 $[1, 5]$ 的孩子节点，调整每个新生成的糖结构的正则顺序；

如果当前字符对应的节点的孩子节点中没有叶子节点，则

在该节点的孩子节点的最左端加入新的叶子节点， Label 范围为 $[1, 5]$ 。

枚举糖结构的过程中，给定节点数目为 n 的糖结构，我们将新的对应着不同单糖类型的节点与糖结构中的已有节点相连，从而构成节点数目为 $n+1$ 的糖结构。我们将糖结构中的已有节点分为如下两种情形来考虑，分别为节点有孩子节点且孩子节点中有叶子节点的情形，和节点有孩子节点且孩子节点中没有叶子节点的情形。下面我们通过具体阐述辅以实例来解释算法设计的原因。

4.1.2.1 节点的孩子节点中有叶子节点的情形

如果当前节点的孩子中有叶子节点，则将新的节点插在最右端叶子节点的右边而没有插在最左端的叶子节点的左边或者时叶子节点之间，且将新的节点插在最右端叶子节点的下一层而没有插在最左端和中间的叶子节点的下一层，是为了避免冗余枚举及带来的额外的调整正则顺序的操作。可以把插入操作看成是数字组合的问题，只在每个数字的右边插入相等或更大的数字。

下面两个例子分别说明了为什么只在最右端的叶子节点的右边以及最右端的叶子节点的下一层插入了新的节点。对于如图 4.4 所示的糖结构，按照我们的算法，我们会在节点 8 的右边和下一层加入新的单糖，插在右边的新的单糖可以为 NeuAc、dHex 这两种，插在下边的新的单糖可以为四种单糖中的任一种。

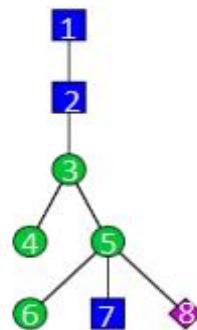


图 4.4 用来说明算法设计的糖结构

例 1 对于孩子节点中有叶子节点的节点，如果不是在最右端叶子节点的右边加入新的节点，那么得到的糖结构虽然没有破坏正则顺序，但这种糖结构是冗余的。具体来说，如果我们将新的节点插入到节点 6（节点 5 的孩子节点中最左端的叶子节点）的左边，则得到的糖结构如图 4.5 所示。

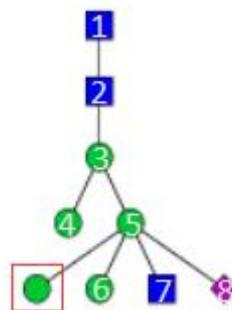


图 4.5 将图 4.4 中所示糖结构的节点 6 的左边插入单糖类型为 Hex 的节点得到的糖结构

由于节点 6 对应的单糖类型 Hex 的标号值为 1，故在节点 6 的左边只能插入

节点对应的标号值不能超过 1，即插入的新节点的单糖类型只能为 Hex。事实上，这个糖结构可以由如图 4.6 所示的糖结构的节点 8 的右边插入单糖类型为 NeuAc 的节点得到。

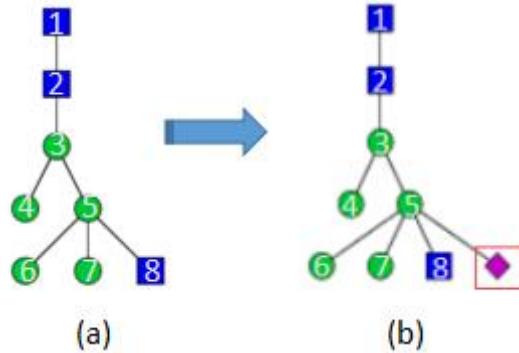


图 4.6 兄弟节点的最右端插入新的节点

将(a)所示的糖结构的节点 8 的右边插入单糖类型为 NeuAc 的节点，得到(b)所示的糖结构也即图 6 所示的糖结构

例 2 对于孩子节点中有叶子节点的节点，如果不是在最右端叶子节点的下一层加入新的节点，则一定会破坏糖结构中兄弟节点的正则顺序，导致得到的新的字符串不再满足规定的正则顺序。图 4.7 展示了一个在最左端叶子节点的下一层加入新的节点之后糖结构对应的正则顺序被破坏的例子。而当我们调整糖结构的正则顺序之后，会发现这个糖结构其实是冗余的，图 4.8 所示的调整好正则顺序的糖结构可以由节点数目为 8 的满足正则顺序的糖结构在最右端叶子节点的右边加入单糖 NeuAc 得到，如图 4.9 所示。如此不仅需要额外的调整正则顺序的操作，还生成了多余的糖结构并增加了去冗余的负担。

如果我们将新的单糖插入到最左端叶子节点的下一层，则得到的糖结构如图 4.7 所示。

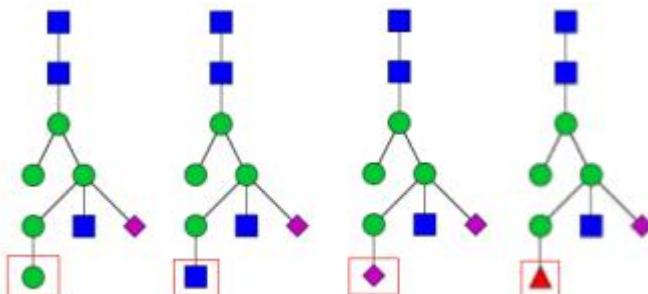


图 4.7 将四种单糖分别加入到图 4.6 所示糖结构的最左端叶子节点的下一层得到的糖结构

事实上，由于在最左端的叶子节点的下一层中加入新的节点之后，最左端的叶子节点对应的子字符串的长度一定大于其它叶子节点对应的子字符串的长度，

从而最左端的叶子节点对应的字符串的正则顺序一定大于其它叶子节点对应的子字符串的正则顺序，因此调整正则顺序之后的糖结构如图 4.8 所示。

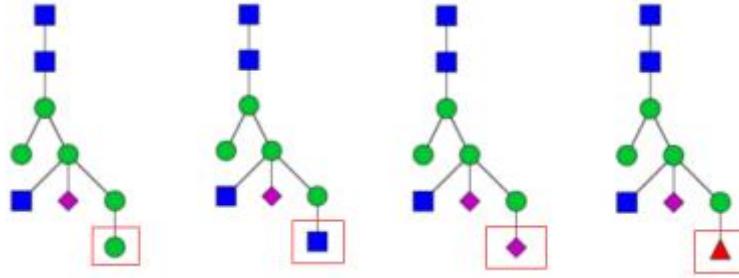


图 4.8 图 4.7 中所示的糖结构调整好正则顺序得到的糖结构

而图 4.8 所示的糖结构实际上可以由图 4.9 所示的糖结构在倒数第二层的叶子节点的最右端插入单糖类型为 NeuAc 的节点得到。

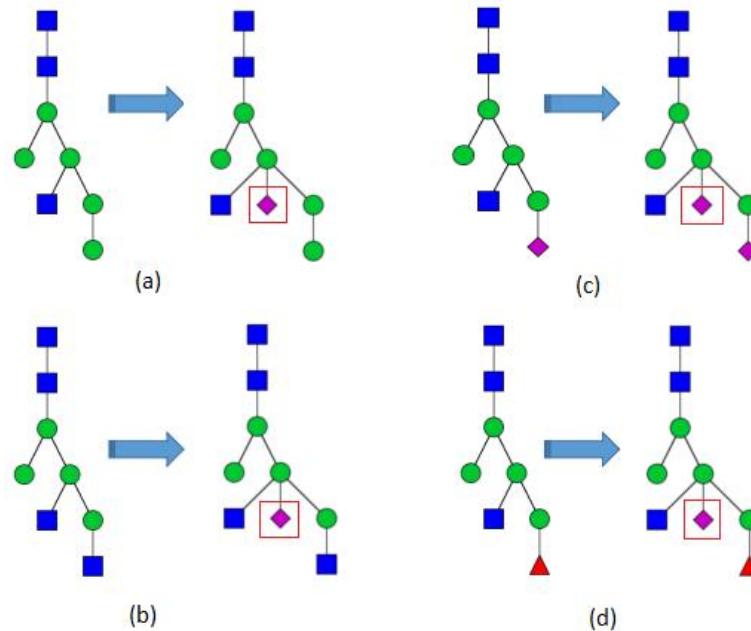


图 4.9 构造 4.8 所示糖结构的另一种方法

子图中左边的糖结构的最右端叶子节点的右边插入新的单糖类型为 NeuAc 的节点后得到图 4.8 所示的糖结构

实际上，上述剪枝策略可行的根本原因在于初始糖结构中只有一类叶子节点。如果有两类及以上叶子节点，则只从最右边叶子节点入手的枚举结果将是不完备的。如果以图 4.10 (a)、(b) 所示的糖结构为基础来枚举，那么我们可以不用在最左端叶子节点的左边及其下一层插入新的叶子节点，其中图 4.10(a)的糖结构

即为五糖核心。而如果以图 4.10(c) 所示的糖结构为基础来枚举，该糖结构的叶子节点对应着两种单糖，则只从最右边叶子节点入手的枚举结果将是不完备的。

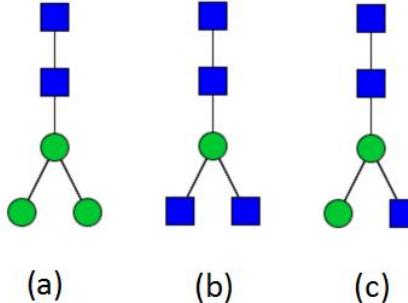


图 4.10 糖结构的枚举模板

糖结构的某一层中，如果同时存在叶子节点和非叶子节点，则叶子节点一定排在非叶子节点的左边（叶子节点对应的字符串长度一定小于任一非叶子节点对应的字符串的长度），因此按照算法中描述的将新的节点插在最右端的叶子节点的右边（也即最左端的非叶子节点的左边）且新节点对应的标号值不小于最右端叶子节点的标号值，这一插入操作不会破坏糖结构的正则顺序，也就不需要后续对正则顺序进行进一步的调整了。

4.1.2.2 节点的孩子节点中没有叶子节点的情形

对于孩子节点中没有叶子节点的情形，我们将新的叶子节点插在该节点的孩子节点的最左端，即将新的节点插在最左端的非叶子节点的左边，标号范围为[1, 5]，因为新节点对应的字符串长度一定小于任一非叶子节点对应的字符串的长度，图 4.11 展示了在五糖核心的节点 1（孩子节点中没有叶子节点）上添加新的节点得到的糖结构。

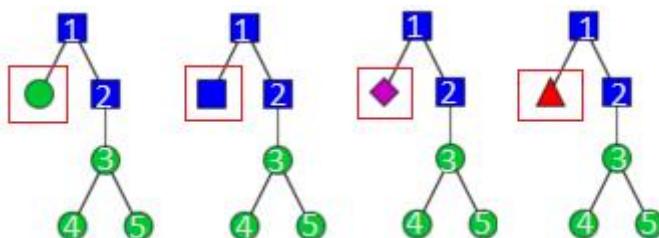


图 4.11 在五糖核心的节点 1 上添加新的节点得到的糖结构

如果糖结构的线性正则字符串码采用的正则顺序是字典序而非 4.1.1 节介绍

的正则顺序，那么，对于节点有孩子节点且孩子节点中没有叶子节点的情形，则不能只在最左端的非叶子节点的左边添加对应着不同单糖类型的新的节点。图 4.13 展示了在图 4.12 所示的糖结构的根节点（孩子节点中没有叶子节点）上添加新的节点得到的糖结构。

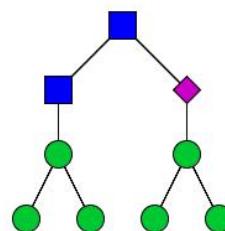


图 4.12 糖结构

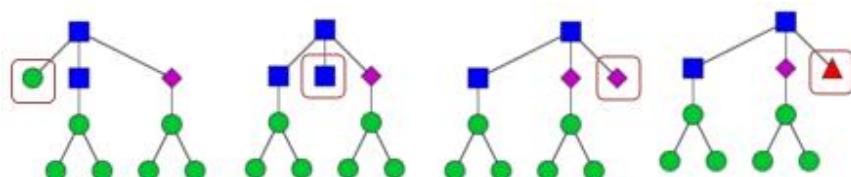


图 4.13 添加不同类型的单糖得到的糖结构

如果用字典序，最大的缺点就是算法不会那么简洁；此外，还有字符串码需要编码的次数变多等问题。

我们按照上述算法插入新的节点之后，仍然可能出现兄弟节点之间的正则顺序被破坏的现象，如图 4.14 所示，在糖结构的节点 4 的孩子节点中最右端叶子节点 6 的下一层插入单糖类型为 Hex 的节点之后，节点 4 对应的字符串码大于节点 5 对应的字符串码。下一节中我们会介绍如何处理这些不满足正则顺序的糖结构。

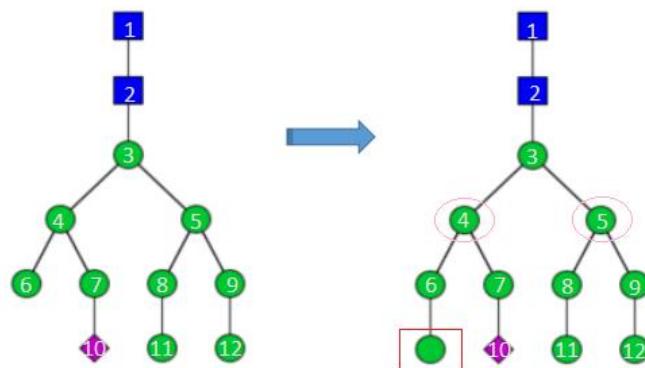


图 4.14 扩展得到不符合正则顺序的糖结构

4.1.3 调整正则顺序方法的设计

我们按照上述算法插入新的节点之后，可能出现兄弟节点之间的正则顺序被破坏的现象。一个很自然的想法就是，既然最终枚举得到的糖结构都是满足正则顺序的，那么我们可否只生成满足正则顺序的糖结构呢？下面展示的例子否定了这一想法。

以五糖核心为起点进行枚举，如果只添加符合正则条件的节点，则无法生成图 4.15 所示的糖结构。由于枚举过程中所有的糖结构都必须包含五糖核心，因此图 4.15 所示的糖结构只可能是由图 4.16 所示的糖结构在节点 3 的孩子节点中最右端叶子节点 6 的右边加入一个新的节点得到。

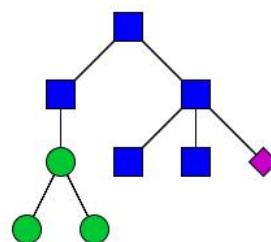


图 4.15 如果只添加符合正则条件的节点则无法生成该糖结构

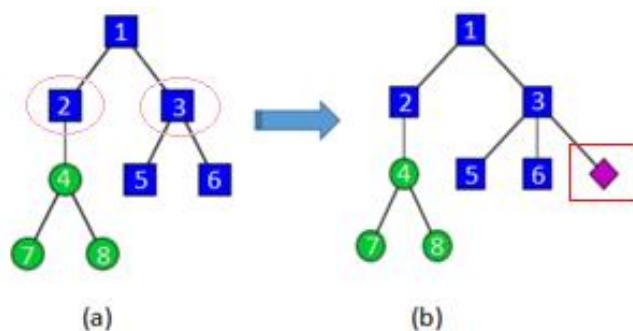


图 4.16 糖结构的扩展示意图

糖结构(b) (图 4.15 所示的糖结构) 只能由糖结构(a)扩展生成

然而，图 4.16 (a)所示的糖结构中，节点 3 对应的字符串码小于节点 2 对应的字符串码，因此这个糖结构不满足正则顺序。也就是说，我们按照上述要求是无法生成图 4.15 所示的糖结构的。

如果我们换一种正则顺序，是否就可以只生成满足正则顺序的糖结构了呢？比如，换成字典序。图 4.17(a)所示的糖结构只能由糖结构(b)扩展生成。但是，糖结构(b)中，第二层左边的节点对应的线性正则字符串码为 B2C1D1dcb，右边

的节点对应的线性正则字符串码为 B2C1D1dD1dcb, 而左边节点的字符串码的字典序大于右边节点的字符串码的字典序。因此糖结构(b)不符合正则顺序。

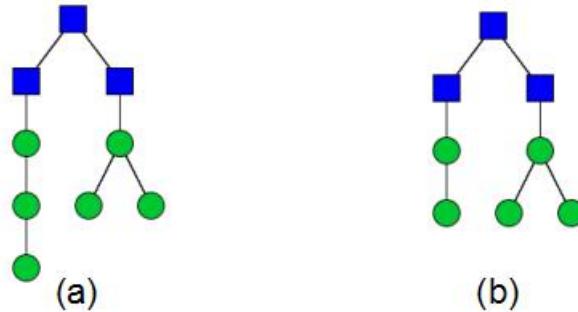


图 4.17 正则顺序为字典序下的扩展示意图

(a)所示的糖结构只能由糖结构(b)扩展生成

出现上述枚举不完全的根本原因在于我们的糖结构库是从一个节点数目为 5 的糖结构 (五糖核心) 开始枚举的, 而不是从 n 个节点的所有糖结构开始枚举出 $n+1$ 个节点的所有糖结构。而如果我们的糖结构不是只有五糖核心, 则图 4.15 所示的糖结构实际上可以在一个不包含五糖核心的糖结构基础上插入新的节点且保持糖结构不违反规定的正则顺序, 如图 4.18 所示。

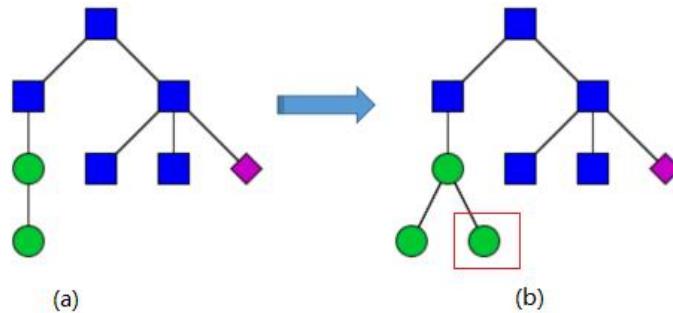


图 4.18 糖结构没有五糖核心限定下的扩展示意图

糖结构(b) (图 4.15 所示的糖结构) 可以由(a)所示的不包含五糖核心的糖结构在不破坏正则顺序的情况下扩展生成

因此, 我们需要进一步调整新构造的字符串码的正则顺序。如果当前字符串包含新加入的字符串片段, 则其祖先节点对应的字符串也包含新加入的字符串片段。回顾第 1 节中我们设计的正则顺序可以保证加入新的节点之后, 该节点及其祖先节点对应的字符串码不会变小, 因此, 我们只需要将该节点的祖先节点与其右兄弟节点进行比较, 如图 4.19 所示。如果加入新的节点之后, 该节点的祖先节点对应的字符串码大于其右兄弟节点对应的字符串码, 则将这两个字符串码调换。

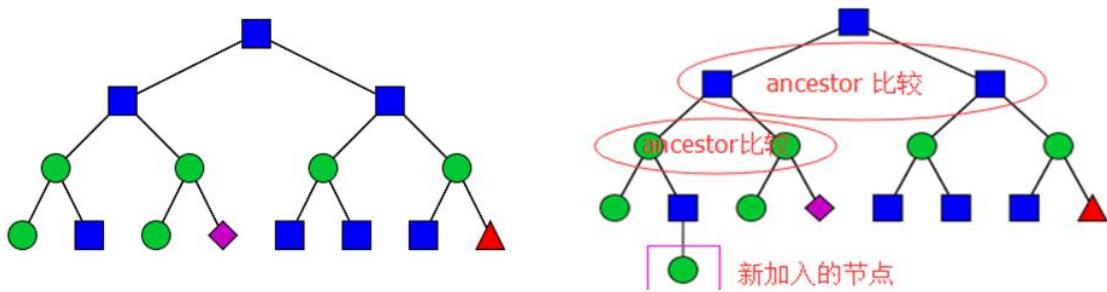


图 4.19 正则顺序的调整示意图

当新加入一个节点之后,依次比较以该节点的祖先节点为根的子串与以其右边的兄弟节点为根的子串的大小,保证该节点上面所有层中兄弟节点是按照正则顺序排列的。

另外,我们在枚举的过程中会筛选掉一些不合理的糖结构,虽然这会导致给定节点数目的糖结构并不完整,但那些为了避免冗余枚举的剪枝操作将依然适用。这里提到的剪枝操作指的是对于孩子节点中有叶子节点的糖结构,只在最右端叶子节点的右边及下一层插入新的节点,而没有在最左端的叶子节点的右边及下一层插入新的节点,也没有在叶子节点之间及中间叶子节点的下一层插入新的节点。采用这一剪枝策略的原因在于 $n+1$ 个节点的糖结构其实可以由多种不同的 n 个节点的糖结构扩展生成,我们只需要保留一种生成方式即可。而这些可能扩展生成同一个新的糖结构的糖结构其实是新的糖结构的部分 Y 离子。按照我们判断糖结构合理的规则,糖结构中任一子结构不合理则该糖结构被认为是不合理的。从而,这一剪枝策略不仅在加上合理糖结构判断规则之后不会漏掉糖结构,还可以有效地对不合理的糖结构进行剪枝。

4.1.4 计算糖结构中每个节点的分支数目的算法

需要注意的是,调整字符串的正则顺序时,糖结构的分支数目也需随之改变。由于我们对糖结构的每种单糖的分支数目设置了一定的阈值限制,我们需要根据糖结构对应的字符串计算出该糖结构中每个节点对应的分支数目。为了方便在加入新的节点的同时,更新及交换节点对应的分支数目,我们把糖结构的中节点的分支数目串起来用字符串表示。

图 4.20 所示的糖结构对应的字符串码为 A2B2C1D2dcC1D1dD1dcba,我们会按照字符串中对应的节点的顺序依次记录每个节点的分支数目,并用字符串表示为 1210200。图 4.21 为在图 4.20 所示的糖结构的基础上插入了一个新的单糖类型为 NeuAc 的节点,对应于字符串码来说,就是在 A2B2C1D2 与 dcC1D1dD1dcba 之间插入了 E3e,从而得到新的糖结构对应的字符串码为

A2B2C1D2E3edcC1D1dD1dcba，则新的分支字符串也更新为 12110200。然而，当前的字符串码已经不再符合正则顺序了，因此我们需要对这个字符串码内部的顺序进行调整。

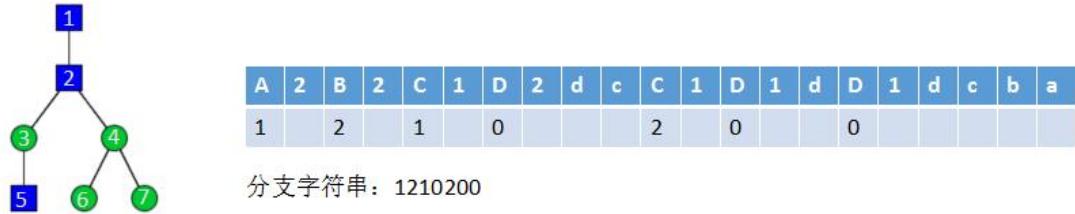


图 4.20 糖结构及其对应的字符串和分支字符串



图 4.21 加入新节点后的糖结构及其对应的字符串和分支字符串

具体的调整方法为，找到插入的新节点对应的父节点即节点 5，节点 5 没有兄弟节点，则继续往上一层找到节点 5 的父节点即节点 3，节点 3 有右相邻的兄弟节点即节点 4，比较节点 3 和节点 4 对应的子字符串的大小，发现节点 3 对应的子字符串 C1D2E3edc 比节点 4 对应的子字符串 C1D1dD1dc 大，于是交换节点 3 和节点 4 对应的子字符串（对于糖结构则是将以节点 3 为根的子树与以节点 4 为根的子树互换），而节点 4 没有右兄弟节点了，且节点 3 的父节点即节点 2 没有兄弟节点，因而调整完毕，而分支字符串也会随着糖结构的字符串码的交换同步交换。得到的调整后的糖结构及其对应的字符串和分支字符串如图 4.22 所示。

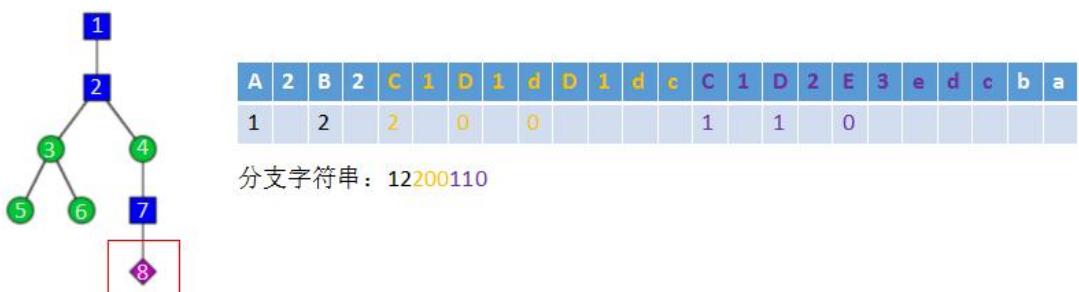


图 4.22 调整正则顺序之后的糖结构及其对应的字符串和分支字符串

4.1.5 相同兄弟节点子字符串的处理

如果糖结构对应的字符串码中相邻的子字符串相同，则说明这些子字符串对应的节点互为兄弟节点，且以这些兄弟节点为根的子树互为镜像结构，如图 4.23 所示。如果以兄弟节点为跟的子树互为镜像结构，则在这些子树上添加相同的新节点得到的新的子结构依然互为镜像结构。因此，我们只在互为镜像的子结构中选择其中一个子结构，并在该子结构的基础上加入新的节点即可。考虑到加入新的节点之后该节点的祖先对应的字符串码不会变小，因此，我们只在以最右端的兄弟节点为根节点的子结构上加入新的节点。

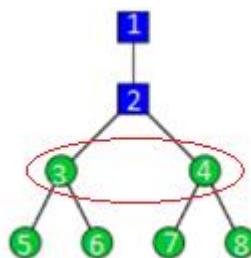


图 4.23 相邻兄弟节点对应的字符串码相同的情形
该糖结构中以节点 3 和节点 4 为根的子树互为镜像结构

图 4.23 所示的糖结构中，以节点 3 为根的子树对应的字符串码和以节点 4 为根的子树对应的字符串码均为 C1D1dD1dc。如果我们在节点 3 而不是节点 4 上加入新的节点，则会导致节点 3 对应的字符串码大于节点 4 对应的字符串码，于是就需要调整糖结构的正则顺序，即交换加入新节点后节点 3 和节点 4 对应的字符串码（对应于糖结构则是将以节点 3 为根的子树与以节点 4 为根的子树互换），调整完正则顺序的字符串码与直接在以节点 4 为根的子树上添加相同的节点得到的字符串完全一致。因此，我们在以最右端的兄弟节点为根节点的子结构上加入新的节点才能保证新生成的糖结构满足正则顺序，从而有效地减少调整正则顺序及生成冗余结构带来的冗余操作。

4.1.6 基于糖结构的字符串码枚举糖结构示例

为了让大家对枚举算法有更加直观的理解，下面我们接下来具体介绍以图 4.24 所示的糖结构为基础枚举出新的糖结构的过程。

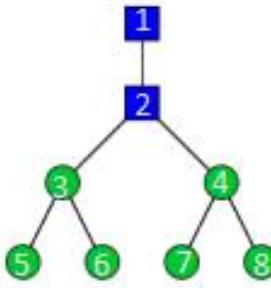


图 4.24 以该糖结构为基础枚举新的糖结构

注意到这个糖结构对应的字符串码 A2B2C1D1dD1dcC1D1dD1dcba 中有两段相邻的子字符串完全相同, 分别对应于以节点 3 为根的子树和以节点 4 为根的子树。而基于互为镜像的子结构进行枚举会生成的新的糖结构依然互为镜像。因此, 我们不在节点 3、5、6 上添加新的边与节点, 即我们会直接跳过以节点 3 为根节点的子树中的任一节点。

对于节点 1, 其孩子节点中没有叶节点, 因此在该节点的孩子节点的最左端加入新的叶子节点, Label 范围为 [1,5], 如图 4.25 所示。

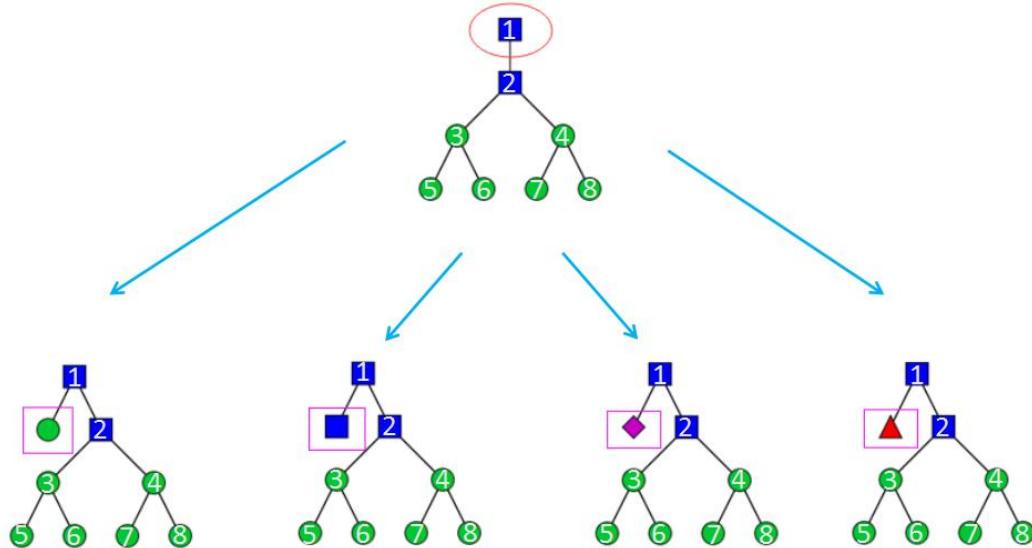


图 4.25 在 4.24 所示的糖结构的节点 1 的孩子节点左边添加新的节点得到的糖结构

节点 2 的的孩子节点中也没有叶节点, 与节点 1 类似, 在其孩子节点的最左端加入新的叶子节点, Label 范围为 [1,5]。

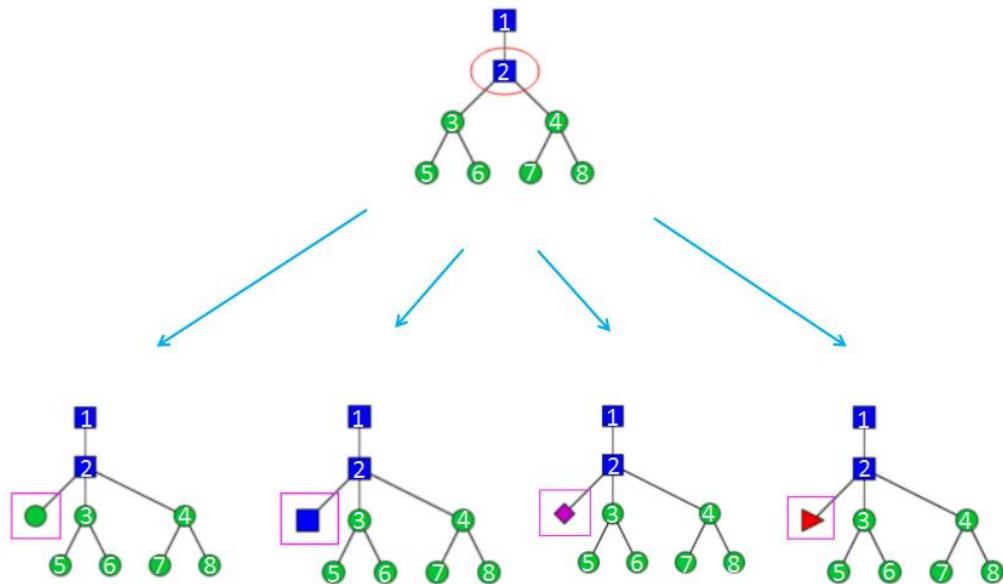


图 4.26 在 4.24 所示的糖结构的节点 2 的孩子节点左边添加新的节点得到的糖结构

对于节点 3, 以节点 3 为根节点的子树与以节点 4 为根节点的子树互为同构, 因此不在以节点 3 为根的子树中的节点包括节点 3、节点 5 和节点 6 上添加新的节点。

对于节点 4, 其有两个孩子节点且均是叶节点。为了避免冗余枚举, 我们不在最左端的叶子节点的左端插入新的节点, 也不在叶子节点之间插入新的兄弟节点, 而是在最右端的叶子结构的右边插入新的节点。由于节点 4 的两个孩子节点对应的单糖均为 Hex, 其标号值均为 1, 因此, 我们在孩子节点 8 的最右端插入标号值不小于 1 的新的节点, 即将四种类型的单糖分别与节点 4 相连, 构成节点数目为 9 的糖结, 如图 4.27 所示。

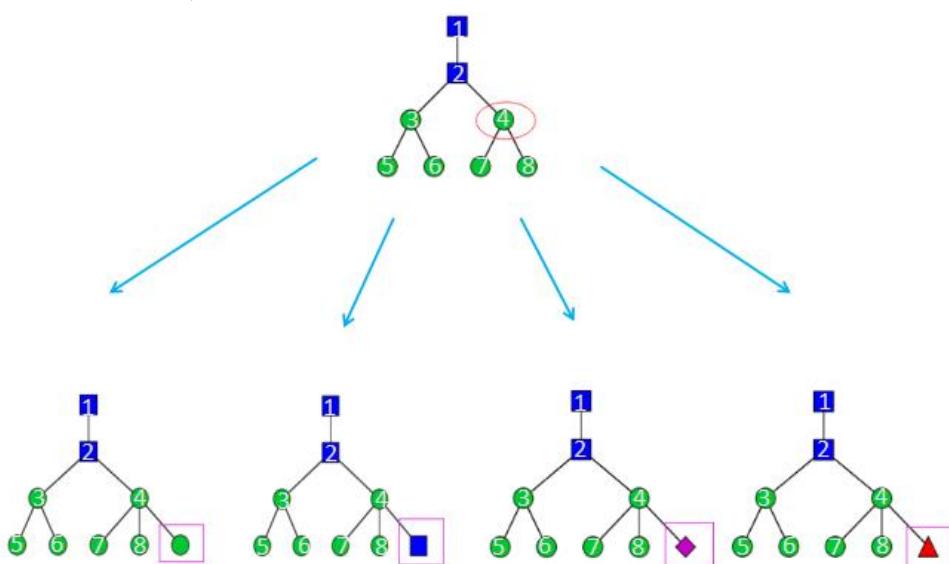


图 4.27 在图 4.24 所示的糖结构的节点 4 的最右端孩子节点的右边添加新的节点得到的糖结构

另外，我们以节点 4 的孩子节点中最右端的叶子节点即节点 8 为父节点，插入新的标号值范围为[1,5]的孩子节点，即将四种类型的单糖分别与节点 8 相连，如图 4.28 所示。

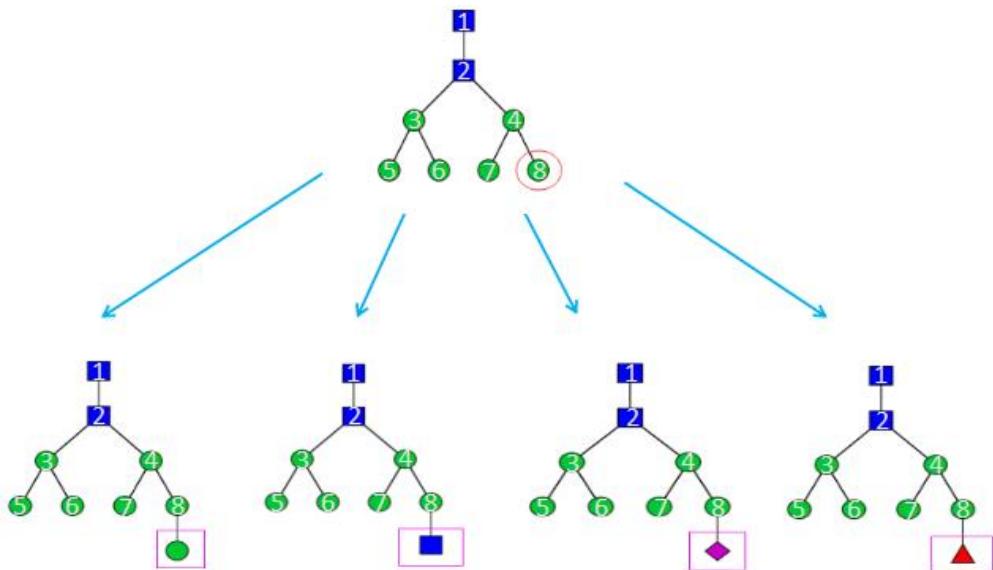


图 4.28 在图 4.24 所示的糖结构的节点 4 的最右端孩子节点的下一层添加新的节点得到的糖结构

4.1.7 枚举算法总结

当前的枚举方法有效地减少了很多水平冗余糖结构的生成，但在垂直冗余方面还亟待改进。下面具体介绍一下垂直冗余。比如，对于图 4.29 所示的糖结构，这个糖结构可以图 4.30 所示的糖结构添加一个单糖类型为 NeuAc 的新节点得到，也可以由图 4.31 所示的糖结构添加一个单糖类型为 Hex 的新节点得到。由于这是垂直方向节点的增加造成的冗余，因而称之为垂直冗余。

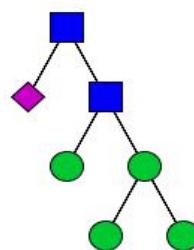


图 4.29 用来说明垂直冗余的糖结构

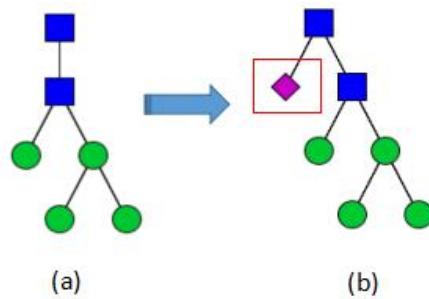


图 4.30 糖结构扩展方法一

糖结构(b) (图 4.29 所示的糖结构) 可以由糖结构(a)加一个单糖类型为 NeuAc 的节点得到

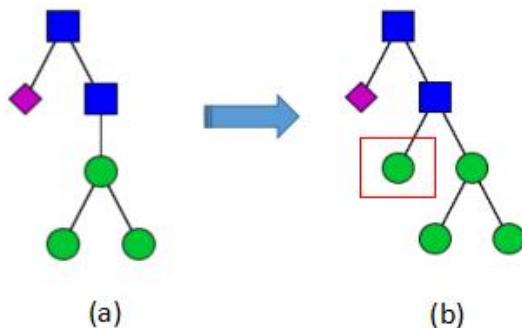


图 4.31 糖结构扩展方法二

糖结构(b) (图 4.29 所示的糖结构) 可以由糖结构(a)加一个单糖类型为 Hex 的节点得到。

4.2 基于糖结构的字符串码生成糖的子结构的组成

为了便于 N-糖肽鉴定软件进行谱峰之间的匹配，我们生成了糖结构的含根子结构的组成（通常简称为子结构），从而为糖肽鉴定提供了理论谱图库。如 3.5 节所述，我们基于糖结构的邻接表存储格式，在节点数目为 n 的子结构的基础上，增加一个与该子结构中任一节点相邻的新节点，得到节点数目为 $n+1$ 的子结构，依此生成糖结构的所有子结构。然而，这种生成子结构的方法伴随着很多冗余子结构的生成，导致时间效率较低。另一方面，糖结构的字符串存储格式相比邻接表的存储格式有着更高的空间效率。于是，我们基于糖结构的字符串码直接生成糖结构的子结构的组成，通过发现节点的子结构由其孩子节点的子结构的组合构成子结构再在其上一层加上该节点得到这一本质，我们设计出了一种高效的子结构生成算法。下面具体介绍基于糖结构的字符串码生成糖结构的组成的方法。

4.2.1 糖结构的子结构的组成生成方法

不妨设 r_1, r_2, \dots, r_k 为节点 r 的孩子节点, 则以 r 为根的树 T_r 的子结构的根节点均为 r , 根节点连接的部分为以 r_1, r_2, \dots, r_k 为根的子树 $T_{r_1}, T_{r_2}, \dots, T_{r_k}$ 的子结构的组合。如图 4.32 中, (b) 为(a) 所示的糖结构对应的线性正则字符串码, (c) 展示了以糖结构的节点 1 为根的子结构的生成过程。算法的基本思想在于, 一个节点的子结构实际上是由其孩子节点的组合构成的。具体来说, 对于分支数目超过 1 的节点, 以该节点为根节点的子结构由以该节点的孩子节点为根节点的子结构的组合构成的子结构再在其上一层加上该节点得到; 对于分支数目为 1 的节点, 以该节点为根节点的子结构为以该节点的孩子为根节点的子结构在其上一层加上该节点得到 (实际上也是前一种情形的特殊形式)。从而, 我们可以有效地根据子结构的 Y 离子生成母结构的 Y 离子。

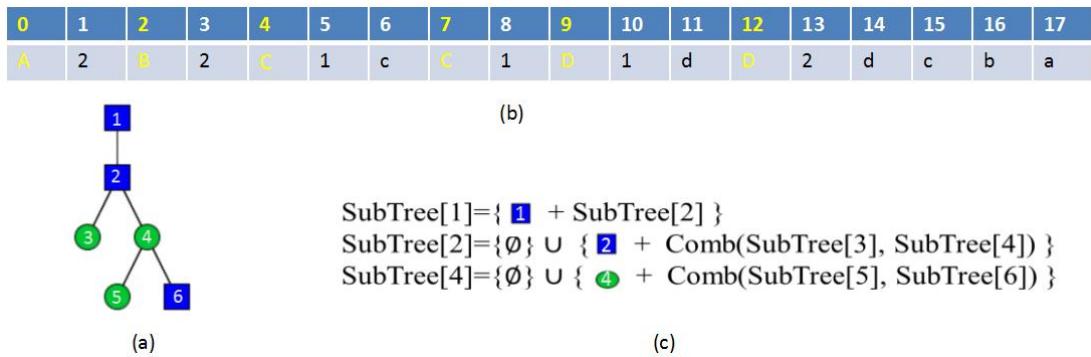


图 4.32 基于糖结构的字符串码生成子结构的方法示例

- (a) 糖结构及其节点对应的编号值
- (b) 糖结构按照双侧外包分层方法编码得到的字符串码
- (c) 生成该糖结构的子结构的递推公式, 其中, $\text{SubTree}[i]$ 表示以字符串码中编号值为 i 的字符对应的节点为根的糖结构对应的子结构集合, “Comb”表示进行组合操作, “+” 表示在下一层加糖结构

为了更好地理解子结构的生成算法, 下面我们对图 4.32 (c) 中所示的糖结构的子结构的构造过程进行详细介绍。从糖结构的字符串码中我们可以方便地解析出给定节点的孩子节点信息, 我们不妨从节点与孩子节点的子结构的关系这一角度仔细观察一下图 4.32 中所示的糖结构。节点 1 只有一个孩子节点即节点 2, 以节点 1 为根的子结构是由以节点 2 为根的子结构的上一层加上节点 1 构成的; 节点 2 有两个孩子节点, 分别为节点 3 和节点 4, 以节点 2 为根的子结构为以节点 3 为根的子结构与以节点 4 为根的子结构组成生成的子结构的上一层加上节点 2 构成的; 节点 4 有两个孩子节点, 分别为节点 5 和节点 6, 以节点 4 为根的子结

构为以节点5为根的子结构与以节点6为根的子结构组成生成的子结构的上一层加上节点4构成的。

因此，如果我们想要得到以节点 1 为根的子结构，就需要生成以节点 2 为根的子结构；

SubTree[1]={ 1 + SubTree[2] }

如果想要得到以节点 2 为根的子结构，就需要分别生成以节点 3 和节点 4 为根的子结构，然后通过这两组子结构的组合得到以节点 2 为根的子结构；

`SubTree[2]= $\{\emptyset\} \cup \{ \textcolor{blue}{2} + \text{Comb}(\text{SubTree}[3], \text{SubTree}[4]) \}$`

如果想要得到以节点 4 为根的子结构，就需要分别生成以节点 5 和节点 6 为根的子结构，然后通过这两组子结构的组合得到以节点 4 为根的子结构。

`SubTree[4]= $\{\emptyset\}$ \cup { ④ + Comb(SubTree[5], SubTree[6]) }`

下面我们从糖结构的叶子节点开始，自底向上依次生成以糖结构的不同节点为根的结构对应的子结构的组成，直到生成以糖结构的根节点为根的结构对应的子结构的组成，其带电形式即为糖结构的 Y 离子。

节点 3、节点 5 和节点 6 均为叶子节点，以这些节点为根的子结构为空集和节点本身。

`SubTree[3] = { \emptyset , ③} SubTree[5] = { \emptyset , ⑤}`

SubTree[6] = { \emptyset , 6 }

生成节点 5 和节点 6 的子结构之后，节点 4 的子结构

`SubTree[4]= $\{\emptyset\} \cup \{ \textcolor{red}{4} + \text{Comb}(\text{SubTree}[5], \text{SubTree}[6]) \}$`

$$= \{ \emptyset, \textcolor{red}{4}, \textcolor{blue}{4}, \textcolor{blue}{4}, \textcolor{blue}{4} \}$$

生成节点 3 和节点 4 的子结构之后，节点 2 的子结构

`SubTree[2]= $\{\emptyset\} \cup \{\blacksquare + \text{Comb}(\text{SubTree}[3], \text{SubTree}[4])\}$`

$$= \{ \emptyset, \text{ } \begin{array}{c} 2 \\ \text{---} \\ 4 \\ \text{---} \\ 5 \end{array}, \text{ } \begin{array}{c} 2 \\ \text{---} \\ 4 \\ \text{---} \\ 6 \end{array}, \text{ } \begin{array}{c} 2 \\ \text{---} \\ 4 \\ \text{---} \\ 5 \\ \text{---} \\ 6 \end{array}, \text{ } \begin{array}{c} 2 \\ \text{---} \\ 4 \\ \text{---} \\ 5 \\ \text{---} \\ 6 \end{array}, \text{ } \begin{array}{c} 2 \\ \text{---} \\ 3 \\ \text{---} \\ 4 \end{array}, \text{ } \begin{array}{c} 2 \\ \text{---} \\ 3 \\ \text{---} \\ 4 \end{array}, \text{ } \begin{array}{c} 2 \\ \text{---} \\ 3 \\ \text{---} \\ 4 \\ \text{---} \\ 5 \end{array}, \text{ } \begin{array}{c} 2 \\ \text{---} \\ 3 \\ \text{---} \\ 4 \\ \text{---} \\ 5 \end{array}, \text{ } \begin{array}{c} 2 \\ \text{---} \\ 3 \\ \text{---} \\ 4 \\ \text{---} \\ 6 \end{array}, \text{ } \begin{array}{c} 2 \\ \text{---} \\ 3 \\ \text{---} \\ 4 \\ \text{---} \\ 6 \end{array}, \text{ } \begin{array}{c} 2 \\ \text{---} \\ 3 \\ \text{---} \\ 5 \\ \text{---} \\ 6 \end{array}, \text{ } \begin{array}{c} 2 \\ \text{---} \\ 3 \\ \text{---} \\ 5 \\ \text{---} \\ 6 \end{array} \}$$

4.2.2 生成糖结构的子结构加速策略探讨

在实现的过程中，如果两个集合中的元素相同，那么对这两个集合中的糖结构进行组合则会生成互为同构的糖结构。改进的方法是，如果两个集合中的糖结

构相同，则把组合操作改为排列操作，从而避免冗余糖结构的生成。

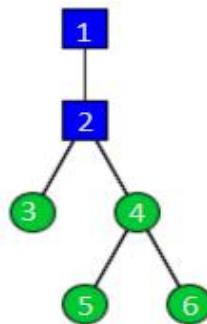


图 4.33 节点 5 和节点 6 的子结构集合完全相同

比如，对于图 4.33 所示的糖结构中，以节点 5 为根的子结构集合与以节点 6 为根的子结构集合完全相同。这样这两个集合进行组合，得到的子结构分别为 $0+0$, $0+1,1+0,1+1$ 。其中， 0 表示空集， 1 表示 Hex 单糖。而 $0+1$ 与 $1+0$ 对应着两个完全相同的子结构。更一般的情况可能生成互为同构的子结构。如果我们采用排列而不是组合的方法，则得到的子结构为 $0+0,0+1,1+1$ ，而不会生成 $1+0$ 这一与 $0+1$ 完全相同的子结构。

如果两个需要进行组合的两个子结构集合不是像图 4.33 所示完全相同，而是部分相同呢？对于如图 4.34 所示的糖结构，节点 4 和节点 5 对应的子结构集合完全相同。然而我们进行组合操作的时候，是将节点 2 的三个孩子节点即节点 3、4、5 对应的子结构集合依次进行组合的。分别记节点 3、4、5 对应的子结构集合为 A、B、C，则 A 先与 B 进行组合，得到子结构集合 $A \text{ Comb } B$ ，然后 $A \text{ Comb } B$ 再与 C 进行组合。那么对于只有部分元素相同的两个集合，我们如何权衡组合与排列操作呢？我们的做法是 $(A \text{ Comb } B) \text{ Comb } C = A + B \text{ Comb } C + \{A\} \text{ Perm } A$ 。

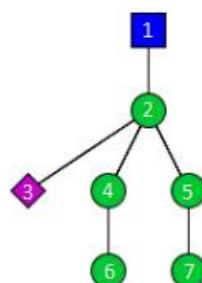
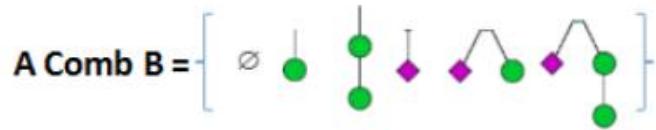


图 4.34 糖结构示例

节点 3 与节点 4 对应的子结构集合不相同，而节点 4 和节点 5 对应的子结构集合完全相同

$$A = \left[\emptyset, \downarrow \right] \quad B = \left[\circ, \begin{array}{c} \bullet \\ | \\ \bullet \end{array} \right] \quad C = \left[\circ, \begin{array}{c} \bullet \\ | \\ \bullet \end{array} \right]$$



下面我们将加速策略扩展为更为一般的情况。假设糖结构中同一层节点对应的子字符串（以该节点为根节点的子字符串）分别为 A、B、C、D。假设 B 和 C 对应的子字符串完全相同。记 A 和 B 进行组合操作之后的字符串集合为 Comp1，C 对应的字符串集合为 Comp2。则先将 Comp1 中除了 Comp2 中的元素与 Comp2 进行组合，并且保证 Comp1 中前面的一些元素仍为 Comp2 中的元素（这样如果 C 右边的子字符串 D 与 C 的子字符串相同，方便进行后续的排列操作），然后再将 Comp2 中除了空字符串的部分与 Comp2 进行排列构成一个新的字符串集合（保证组合成的新字符串中没有 Comp2 中的元素），最后把这个新的字符串集合加入到 Comp1 中。

另外需要注意的是，该算法是针对构造糖结构的子结构的组成的需求量身定做的，如果想生成糖结构的所有非冗余子结构，则还需要考虑字符串的正则顺序调整等问题，因为不同的子结构组合的过程中可能会破坏字符串的正则顺序。

4.3 枚举算法的效率评测

以下评测实验的测试环境如下：

表 4.1 测试环境

版本	Windows 7 专业版
内存 (GB)	8.00
处理器	Intel(R) Xeon(R) CPU E3-1240 v3 @ 3.40GHz

4.3.1 基于字符串码构建糖结构库的剪枝策略效率评测

我们在基于糖结构的字符串码构建 N-糖结构库的过程中，除了采用了 4.1.5 节中介绍的跳过兄弟节点对应的相同的子字符串这一剪枝策略之外，还有四个剪枝策略，下面具体描述这四种剪枝策略。假设糖结构中，节点 R 有 M 个为叶子节点的孩子节点，这 M 个孩子节点 v_1, v_2, \dots, v_M 对应的标号值分别为

L_1, L_2, \dots, L_M , 且满足 $L_1 \leq L_2 \leq \dots \leq L_M$ 。则

- 1) 我们不在最左端的叶子节点 v_7 的左边加上新的节点;
- 2) 我们不在叶子节点之间加入新的节点;
- 3) 我们不在最左端叶子节点 v_7 的下一层加上新的节点;
- 4) 我们不在中间叶子节点的下一层加上新的节点。

下面我们分别对上述剪枝策略的时间效率及空间效率进行测试，如表 4.2 和表 4.3 所示。测试数据集为合理且非同构的 6~10、6~11 以及 6~12 个节点的糖结构。其中，合理且互不同构的 6~10 个节点糖结构共有 16,649 个，合理且互不同构的 6~11 个节点糖结构共有 104,042 个，合理且互不同构的 6~12 个节点糖结构共有 679,290 个。

表 4.2 剪枝策略的时间效率测试

基于字符串枚举所需时间	不采取剪枝策略 (秒)	采取剪枝策略 (秒)
枚举 6~10 个节点糖结构	4.9	3.5
枚举 6~11 个节点糖结构	57	37
枚举 6~12 个节点糖结构	402	247

表 4.3 剪枝策略的空间效率测试

基于字符串枚举所需时间	不采取剪枝策略	采取剪枝策略
枚举 6~10 个节点糖结构	129,760	76,370
枚举 6~11 个节点糖结构	867,944	508,373
枚举 6~12 个节点糖结构	6,033,488	3,506,183

4.3.2 基于字符串码生成糖结构的子结构的算法效率评测

我们在 3.5 节介绍了基于邻接表生成糖结构的子结构的方法，在 4.2 节又进一步介绍了基于糖结构的线性正则表达式生成糖结构的子结构的方法，我们分别用这两种算法生成 GlycomeDB 中的 N-糖与人类血清中三个最大通用结构的所有子结构合并得到的 7,884 个糖结构的 Y 离子，基于邻接表的子结构生成方法耗时 650s，而基于糖结构的线性正则表达式的子结构生成方法耗时 2s。基于邻接表生

成子结构的过程中，我们是由所有节点数目为 n 的子结构构造节点数目为 n+1 的子结构。对于每个节点数目为 n 的子结构，对于该子结构的每个节点，我们找到与该节点相邻且不在该子结构中的新节点，将该节点与新的节点相连构成节点数目为 n+1 的子结构。而基于糖结构的线性正则表达式的子结构生成方法的主要思想在于，糖结构的子结构是由以糖结构的孩子节点为根的结构的子结构的组合再上接根节点构成。通过这一本质的发现，我们不再盲目地以所有节点数目为 n 的子结构生成节点数目为 n+1 的子结构，而是以节点数目为 n 的兄弟节点的节点数目为 1~n 的子结构的组合构造以这些兄弟节点的父节点为根的节点数目为 n+1 的糖结构的子结构。

4.3.3 基于字符串码构建糖结构库的算法效率评测

我们分别用基于糖结构的字符串码和基于糖结构的邻接表构造理论糖结构的方法枚举出 6~10、6~11 以及 6~12 个节点的糖结构，对齐了这两种不同枚举算法的枚举结果，并对两种方法的枚举时间进行了统计，如表 4.4 所示。

表 4.4 基于字符串码构建糖结构库的算法效率评测

	基于糖结构枚举 (秒)	基于字符串枚举 (秒)
枚举 6~10 个节点糖结构	24	4
枚举 6~11 个节点糖结构	213	32
枚举 6~12 个节点糖结构	2902	296

第五章 合并库的构建及枚举库的应用实验

由于同构的糖结构对应的理论谱图相同，我们称同构的糖结构互为冗余。根据第二章中介绍的线性正则编码方法对糖结构进行编码，把同构的糖结构编码为相同的字符串，从而为糖结构的同构冗余判定提供了一种简单有效的方法。根据这一有效的糖结构去冗余方法，我们首先合并了由所有可能的高甘露、混合型和复杂型糖结构[76]构成的 N-糖库与 GlycomeDB 中提取的 N-糖库，构成了一个包含互不同构的糖结构的库，称之为合并库。我们分析了枚举库与合并库之间的包含关系，发现有些已经被记录在 GlycomeDB 中的 N-糖结构却不符合 GP Finder 规则，说明 GP Finder 规则还有待进一步完善。最后，我们分别将合并库和枚举库中的糖结构按照 Y 离子组成归并，通过对基于合并库和基于枚举库的谱图鉴定结果，我们发现枚举库有助于发现潜在的新的糖结构。

5.1 糖结构合并库的构建

GlycomeDB[32-34]集成了七个公开的糖库并统一了糖结构的存储格式，成为当前得到广泛应用的糖库。通过对 GlycomeDB 中糖结构的分析，我们发现 GlycomeDB 库中存在着大量的同构冗余——5,052 个 N-糖中，有 3,765 个糖结构是同构冗余的。进一步，我们根据已知的生物规则[76]生成了人类血清中可能的高甘露、混合型及复杂型 N-糖结构，得到了一个仅包含 1,878 个互不同构的 N-糖结构的库，称之为人类血清通用结构库。通过比较 GlycomeDB 的 1,287 个互不同构的 N-糖与人类血清通用结构库中的 1,878 个 N-糖，我们发现二者共有的糖结构数目只有 439 个，这一结果反映了这两个糖库的不完整性。考虑到当前糖库的局限性及糖库在谱图鉴定中的重要作用，我们合并了 GlycomeDB 的 N-糖与人类血清通用结构库，构成了一个包含更多常见糖结构的库，称之为合并库。

5.1.1 合并 GlycomeDB 的 N-糖库及人类血清通用结构库

对于给定的糖结构 A，称与 A 有相同根节点的子结构（包括 A）为糖结构 A 的子结构，也称为 A 的 Y 离子。我们生成并合并了图 5.1 所示的人类血清中三个最大可能的 N-糖结构的子结构，构造了由所有可能的高甘露、混合型和复杂型糖结构[76]构成的 N-糖库，称之为人类血清通用结构库。考虑到人类血清通用结构库与 GlycomeDB 的 N-糖库的存储方式不同，我们先统一了这两个库的存储

格式，然后对这两个库之间互为同构的糖结构去冗余，最终得到一个包含更多常见糖结构的非冗余库，称之为合并库。

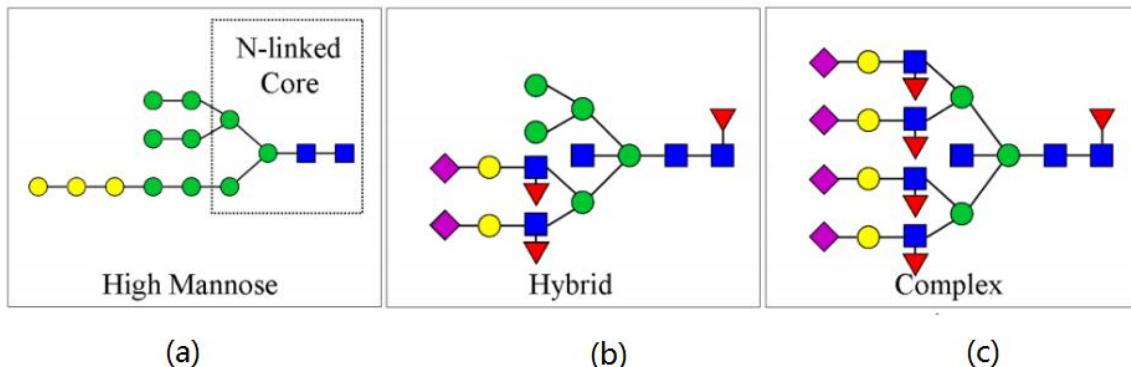


图 5.1 人类血清中三个最大可能的 N-糖[76]

(a)最大可能的高甘露糖结构 (b)最大可能的混合型糖结构 (c)最大可能的复杂型糖结构

5.1.2 合并库中糖结构的分布及来源

我们生成了人类血清中三个最大可能的 N-糖结构的子结构，然后，对这些子结构去冗余，最终得到包含 1,878 个互不同构的糖结构的人类血清通用结构库。GlycomeDB 的 8,017 个糖结构中，包含五糖核心的糖结构即 N-糖共有 5,052 个。我们对提取出的 N-糖去同构冗余，最终得到 1,287 个互不同构的糖结构。

通过比较 GlycomeDB 的 N-糖库与人类血清通用结构库，我们发现 GlycomeDB 库中有 848 个 N-糖结构不在人类血清通用结构库中，而人类血清通用结构库独有的 N-糖结构数目为 1,439，两个库共有的 N-糖结构有 439 个，如图 5.2 所示。我们将人类血清通用结构库中的 1,878 个 N-糖结构与 GlycomeDB 独有的 848 个 N-糖结构合并，得到一个包含 2,726 个互不同构的 N-糖结构的库。我们将合并库中的 2,726 个 N-糖结构中的 NeuAc 部分地替换为 NeuGc 之后，得到由五种单糖构成的 N-糖结构库，包含 1,615 个不同的糖组成，7,682 个互不同构的 N-糖结构。图 5.2 展示了这两种合并库中 GlycomeDB 的 N-糖库与人类血清通用结构库的交并差。

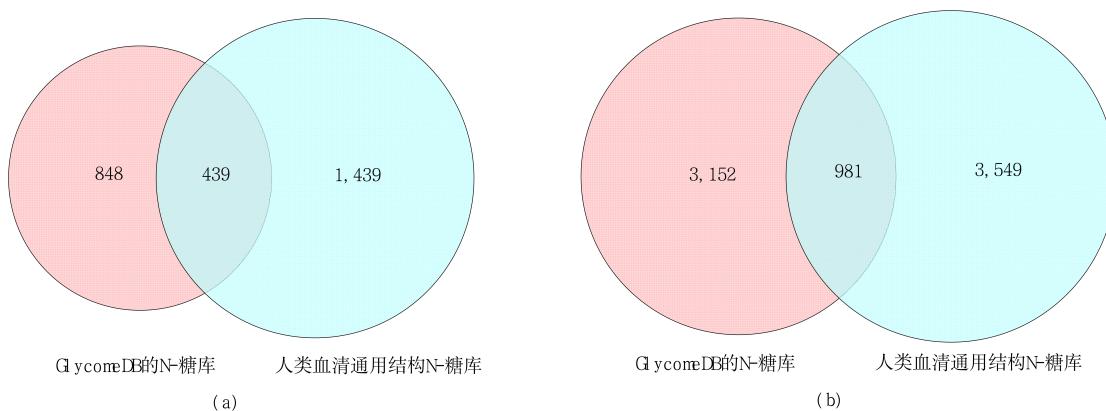


图 5.2 GlycomeDB 的 N-糖库与人类血清通用结构库的交并差

(a)由除了 NeuGc 之外的四种单糖构成的合并库的交并差 (b)由包含 NeuGc 的五种单糖构成的合并库的交并差

合并库中的糖结构分别来自最大可能的高甘露糖结构、最大可能的混合型糖结构、最大可能的复杂型糖结构以及 GlycomeDB 的 N 糖库。为了更加方便地分析糖肽的微观不均一性[56],我们记录了糖结构的来源。我们对每种糖的来源进行了编码, 上述四种来源对应的编码值分别为: 0001, 0010, 0100, 1000。如果某个糖结构有不止一种来源, 则将其对应的编码值进行或操作。假设某个糖结构既是最大可能的高甘露糖结构的子结构, 也是最大可能的混合型糖结构的子结构, 则其对应的编码值为 0001|0010=0011。合并库的 7,682 个互不同构的 N-糖结构中, 有 33 个糖结构来自最大高甘露糖, 540 个糖结构来自最大混合型糖, 4,140 个糖结构来自最大复杂型糖, 4,133 个糖结构来自 GlycomeDB 的 N-糖库。这 7,682 个 N-糖结构的来源分布如图 5.3 所示。

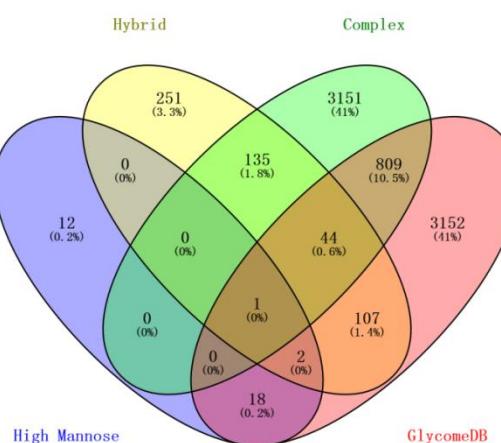


图 5.3 由五种单糖构成的合并库中糖结构来源的分布图

5.2 合并库与枚举库的包含关系分析

基于糖结构的字符串码，我们比较了枚举库和合并库中 6~10 个节点由除了 NeuGc 之外的四种单糖构成的 N-糖结构，发现合并库中有 11 个糖结构没有包含在枚举库中。其中，节点数目为 8 的糖结构有 1 个，节点数目为 9 的糖结构有 4 个，节点数目为 10 的糖结构有 6 个，且这些糖结构均来自 GlycomeDB。通过进一步分析，我们发现这 11 个糖结构都不符合生物规则，因而在按照规则筛选糖结构的过程中被过滤掉了。这说明了用于筛选合理糖结构的规则还不完全适用于所有的糖结构。图 5.4 展示了这 11 个不符合规则的糖结构及糖结构中不符合 GP Finder 规则的子结构。

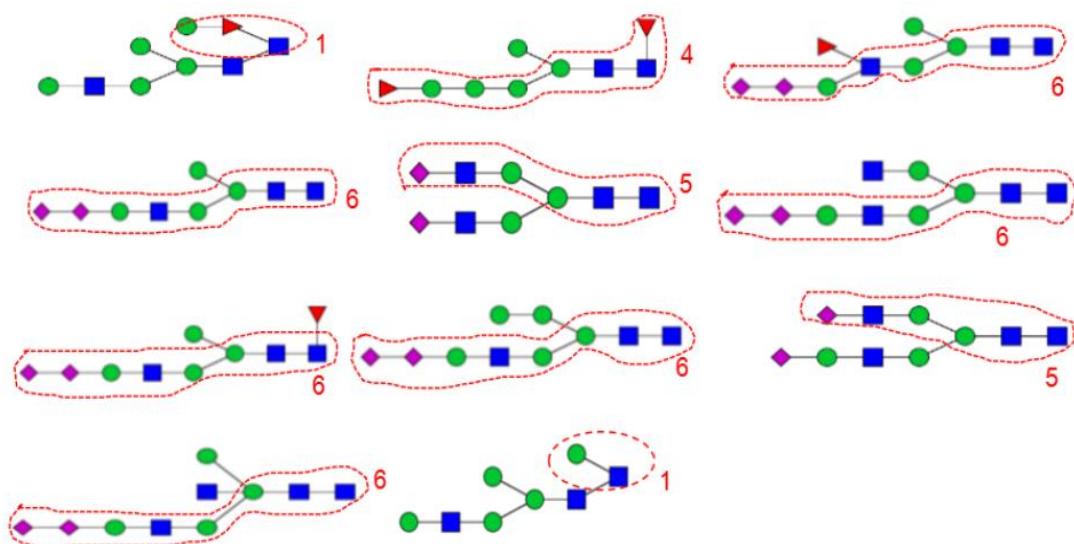


图 5.4 包含在合并库(6~10 个节点)却不在枚举库(6~10 个节点)中的糖结构

分析这些糖结构的位于虚线框中的 Y 离子的组成可以发现，虚线框中的均不符合 GP Finder 规则（具体违反规则的序号在图中已经标出），因而在枚举过程中被过滤掉了。比如，对于位于左上角的第一个糖结构，该糖结构中虚线框的 Y 离子的组成为[1,1,0,0,1]，这一 Y 离子不满足生物规则中“如果 Hex 的数目不为 0，则 HexNAc 的数目一定不小于 2”的约束条件，即不符合 GP Finder 规则 1，故被判定为不合理的糖结构。

进一步，对于 5.1 节中介绍的由 2,726 个互不同构的 N-糖结构构成的库，我们发现合并库中一共有 2,582 个糖结构是符合 GP Finder 规则的，而 144 个糖结构不符合 GP Finder 规则。由于枚举库中筛除了所有不符合 GP Finder 规则的糖结构，因此，合并库中这 144 个糖结构均不包含在枚举库中。合并库中不符合 GP Finder 规则的糖结构的分布图如图 5.5 所示。

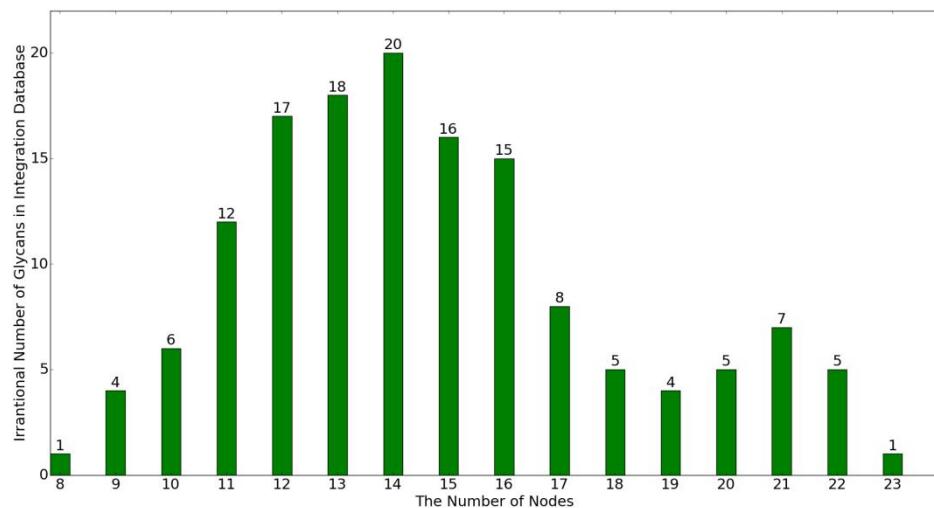


图 5.5 合并库中不符合 GP Finder 规则的糖结构的分布图

5.3 糖结构等价类的分析

当前的枚举库和合并库中不包含互为同构的糖结构，却仍然包含一些理论 Y 离子完全相同的糖结构。比如，图 5.6 所示的三个糖结构，这三个糖结构的理论碎裂情况完全相同。一方面，Y 离子相同的糖结构对应的理论谱图完全相同，因此糖肽鉴定软件如 pGlyco 2.0 无法根据糖结构的理论碎裂情况区分出这些糖结构。另一方面，Y 离子相同的糖结构可能会给糖肽鉴定带来很大的干扰，导致软件的性能下降。鉴于此，我们将糖库中 Y 离子完全相同的糖结构归并为同一个类，称之为等价类。

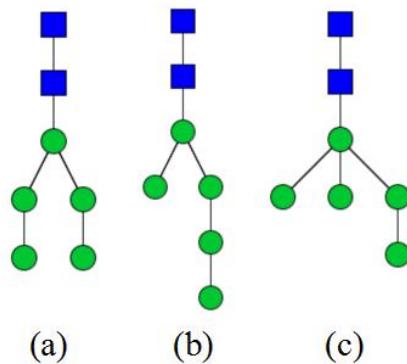


图 5.6 理论 Y 离子完全相同的糖结构示例

下面具体说明枚举库中属于同一等价类的糖结构对 pGlyco 2.0 软件糖肽鉴定过程的影响。我们统计了枚举库中节点数目为 11 的糖结构，发现包含超过 20

个糖结构的等价类有 574 个，最大的等价类包含 307 个糖结构，其对应的糖组成为[9,2,0,0,0]。pGlyco 2.0 软件会在粗打分阶段选择 Top200 的糖结构，在细打分阶段选择 Top20 的糖结构。给定一张谱图，假如这张谱图对应着组成为[8,3,0,0,0]的糖结构，而粗打分阶段，组成为[9,2,0,0,0]的糖结构排名第一，那么同时会有另外 306 个组成也为[9,2,0,0,0]与之并列第一。于是，在粗打分阶段，正确的糖结构[8,3,0,0,0]的打分排名不在 Top200 之内，也就不能进入下一轮的细打分，导致最终的鉴定结果错误。图 5.7 中给出了枚举库中节点数目为 11 的由五种单糖构成的糖结构的等价类分布情况。

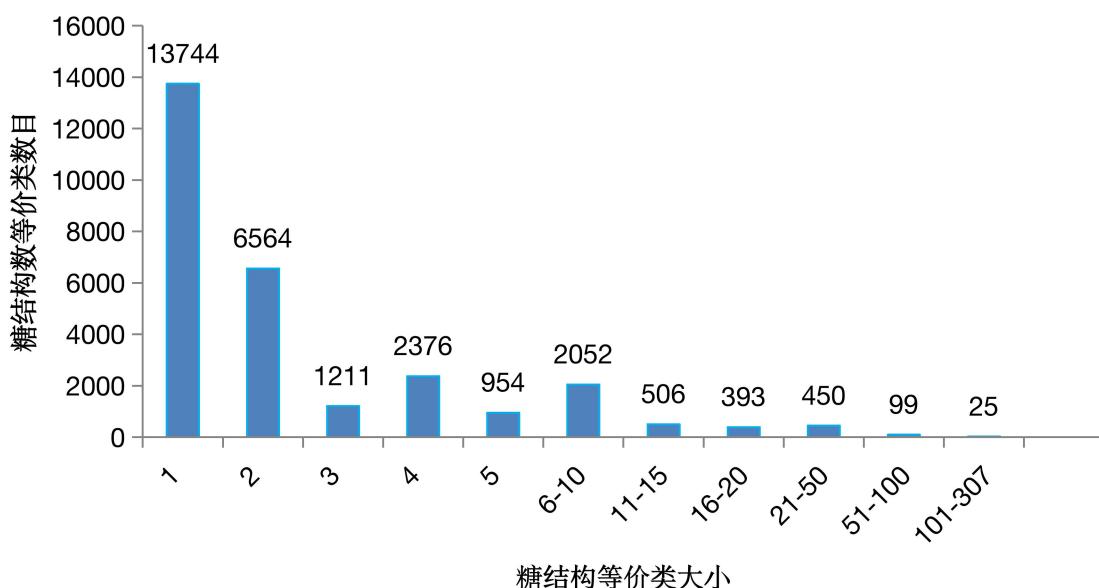


图 5.7 枚举库中节点数目为 11 的糖结构的等价类分布

枚举库中节点数目为 11 的糖结构中，包含超过 20 个糖结构的等价类有 574 个，包含超过 200 个糖结构的等价类有 4 个，最大的等价类包含 307 个糖结构，其对应的糖组成为[9,2,0,0,0]。考虑到质谱也无法区分等价类，我们可以在搜索枚举库之前，就把糖结构按照等价类合并。

此外，我们统计了合并库中 7,682 个由五种单糖构成的 N-糖结构等价类的分布情况，如图 5.8 所示。可以看出，合并库中只包含一个糖结构的等价类有 6,942 个，说明了合并库中大概有 90% 的糖结构对应的理论谱图都是唯一的。最大的等价类包含 11 个糖结构，一共有 3 个。虽然从这一分布情况可以看出，合并库中等价糖结构的存在对 pGlyco 2.0 软件的性能影响不是很大，但 pGlyco 2.0 根据糖结构的理论 Y 离子进行匹配打分还是无法区分属于同一等价类的糖结构。因此，我们在使用合并库进行鉴定之前，依然会预先对该库中的糖结构进行等价类归并。

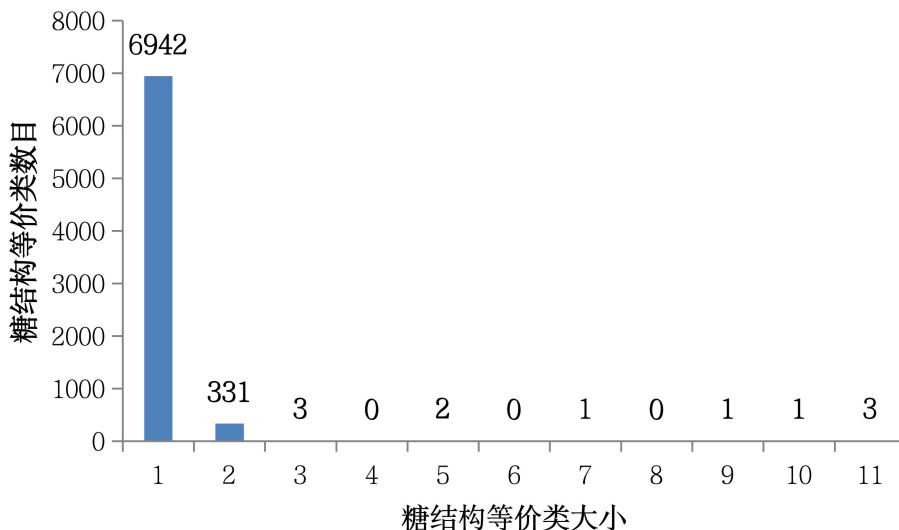


图 5.8 合并库中 7,682 个糖结构的等价类分布图

5.4 枚举库的应用实验

我们枚举了节点数目为 6~14 的由除了 NeuGc 之外的四种单糖构成的合理且互不同构的糖结构，并统计了每个节点数目下糖库的规模，如图 5.9 所示。可以看出，糖库的规模随着糖结构的节点数目呈指数增长趋势。经统计，枚举库中节点数目为 6~14 的合理且互不同构的糖结构共有 31,705,650 个。随着搜索空间的增大，谱图匹配过程中的干扰项也会增多，使得谱图鉴定的难度增大。然而，如果某张谱图在合并库和枚举库下都以 1% 的 FDR 被鉴定到，且鉴定结果相同，则可以认为该鉴定结果更加可信。另外，我们通过实验发现，枚举库单独鉴定到的结果有助于发现一些新的糖结构。

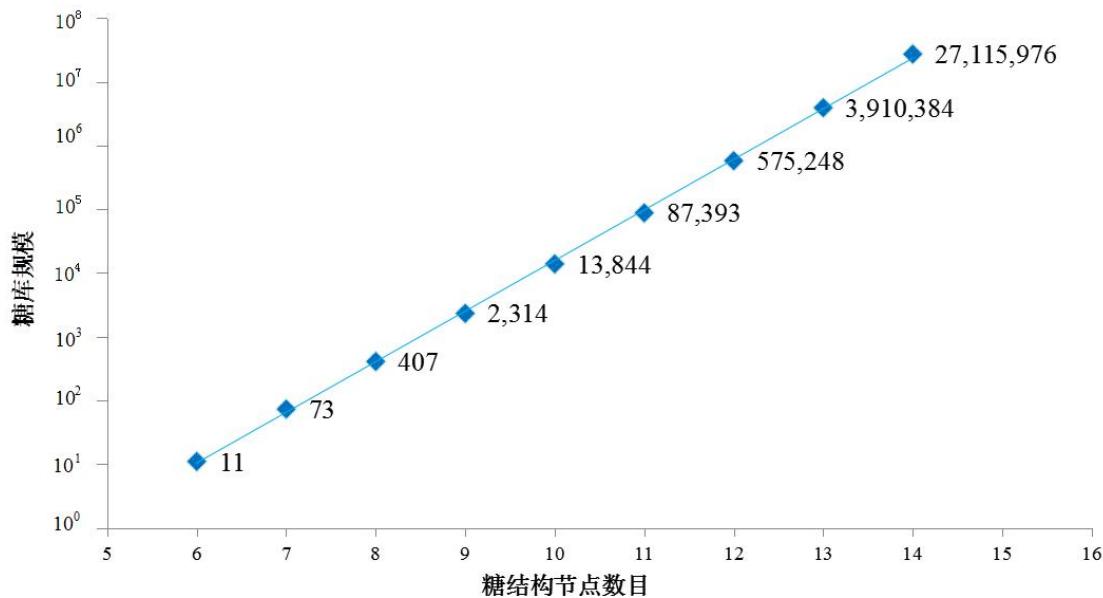


图 5.9 糖结构的节点数目及糖库规模

我们分别将枚举库和合并库中由五种单糖构成的 6~11 个节点的糖结构按照等价类合并，枚举库的规模由 116,623 个糖结构缩减到 33,966 个糖结构，合并库的规模由 724 个糖结构缩减到 513 个糖结构。然后用 pGlyco2.0 分别基于按照等价类归并后的枚举库和合并库对鼠肝 HCD-MS/MS 样品进行鉴定，搜索参数如下：母离子和碎片离子的质量偏差分别为 $\pm 5\text{ppm}$ 和 $\pm 20\text{ppm}$ ；酶切类型为胰蛋白酶的特异性酶切，且最大遗漏酶切位点数目为 2；固定修饰为 Cys 上的 Carbamidomethylation，可变修饰分别为 Met 上的 Oxidation 和蛋白质 N-端的 Acetylation。

在 $\text{FDR} \leq 1\%$ 的情况下，pGlyco 2.0 基于合并库鉴定到了 4,633 张谱图，基于枚举库鉴定到了 4,658 张谱图，用于对比分析基于这两个糖库的鉴定结果的韦恩图如图 5.10 所示。可以看出，合并库单独鉴定出了 85 张谱图，枚举库没有鉴定到这些谱图的因素包括：1) 枚举库相比合并库包含了更多数目的糖结构，这也意味着每张谱图的鉴定过程中面临着更多的干扰项，导致鉴定到的糖结构为假阳，2) 我们采用的 GP Finder 的过滤规则还不足以适用于所有的糖结构，导致合并库中的一些糖结构并没有包含在枚举库中，3) 枚举库鉴定到了其中一些谱图，但是被 FDR 卡掉了。另外，我们发现枚举库相比合并库单独鉴定出了 110 张谱图。为了避免出现合并库也鉴定到了相应的谱图但被 FDR 卡掉的情况，我们将合并库中所有的鉴定结果与枚举库中满足 FDR 值不超过 1% 的鉴定结果比较，发现枚举库单独鉴定出 10 张谱图，且这 10 张谱图对应的糖组成均为 [7,3,0,1,0]。图 5.11 展示了其中一张谱图的鉴定结果及其谱图的匹配情况。

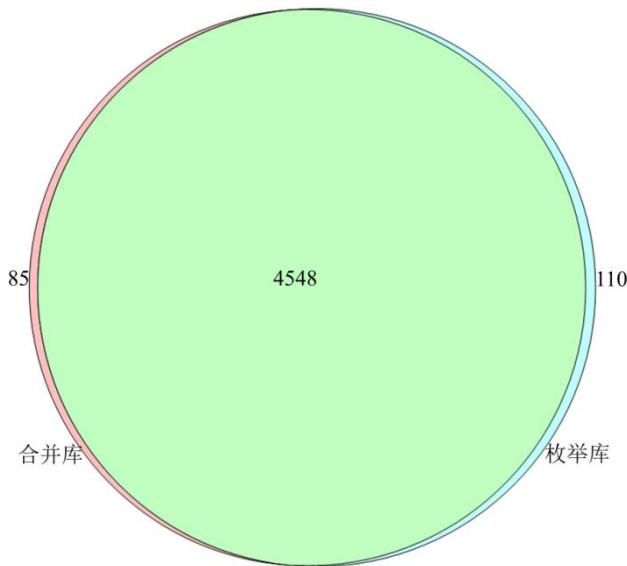


图 5.10 pGlyco 2.0 分别基于合并库和枚举库鉴定到的谱图数目的韦恩图

pGlyco 2.0 基于合并库单独鉴定了 85 张谱图，基于枚举库单独鉴定了 110 张谱图，二者共同鉴定了 4,548 张谱图，且这 4,548 张谱图分别基于枚举库和合并库鉴定到的肽段和糖组成完全相同。

我们进一步对该鉴定结果进行手工验证，可以看出高丰度的氧鎓离子匹配上了，说明这是一张糖肽谱图。另外， Y_1 离子对应于除了唾液酸的特征离子峰之外强度最高的谱峰，并且大部分高强度的谱峰都以较小的质量偏差匹配上了，说明这张谱图的鉴定结果相当可信。该鉴定结果中的糖结构对应于枚举库中 19 个组成为 [7,3,0,1,0] 且属于同一个等价类的糖结构（图中右上角展示了其中一个糖结构，其它 18 个糖结构如图 5.12 所示），而合并库中并不包含组成为 [7,3,0,1,0] 的糖结构。此外，我们在 GlyTouCan 1.0[35] 及 JCGGDB(http://jcgdb.jp/index_en.html) 这些公开的数据库中分别搜索糖结构及糖组成，均未发现相应的糖链。考虑到枚举库中包含一些自然界中还未被发现并记录的糖结构，我们可以基于枚举库鉴定到一些新的糖结构，从而在一定程度上克服传统糖库的局限性，但同时，更大的糖库可能导致匹配过程中的干扰项增多，这就不可避免地给糖肽的鉴定提出了更大的挑战。

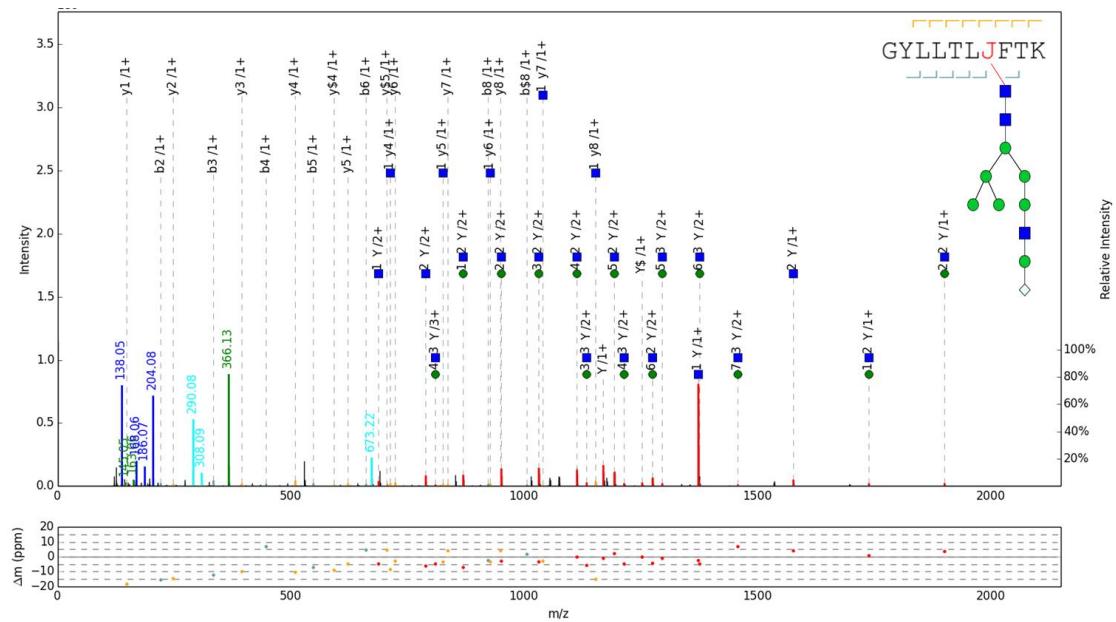


图 5.11 pGlyco 2.0 基于枚举库比基于合并库多鉴定到的一张谱图

这是一个母离子为三电荷的糖肽串联质谱图，红色的谱峰表示匹配到的糖肽 Y 离子，橘色和绿色的谱峰表示匹配到糖肽离子中对应的肽段分别为肽段 y 离子和 b 离子。pGlyco 2.0 基于枚举库将该谱图鉴定为 GYLLTLJFTK-[7,3,0,1,0]。其中，“J”表示糖基化位点 (N-X-S/T/C)，[7,3,0,1,0] 的每一维分别表示该糖结构中 Hex、HexNAc、NeuAc、NeuGc、dHex 这五种单糖的数目。枚举库中一共有 19 个糖结构的理论谱图包含该谱图中所有匹配到的理论谱峰，且这 19 个糖结构对应的理论谱图完全相同，即属于同一个等价类。

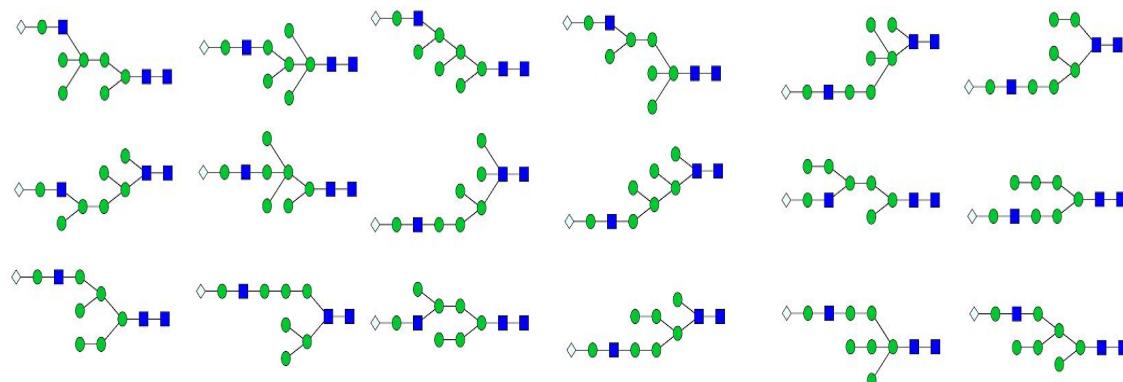


图 5.12 枚举库鉴定到的新的糖链对应的可能的结构

这些糖结构对应的 Y 离子的组成完全相同。

第六章 基于线性正则表达式的搜索引擎设计

在第一章中我们介绍了糖肽的微观不均一性，即相同的位点可能连接着多种不同的糖结构，而糖结构的多样性可能导致糖肽的碎裂情况有一定的差异。在糖肽对应的肽段序列相同的情况下，如果糖结构太大，则会一定程度上增大肽段碎裂的难度，肽段可能不能满足 FDR 阈值限制而被搜索引擎误判为错误的鉴定结果。因此，我们希望在常规的鉴定结果的基础上，再增加一轮筛选过程，通过谱谱之间的匹配召回更多可信的结果。本章首先通过具体的实例说明了 gMatch 的设计缘由，然后介绍了 gMatch 的具体实现流程，最后通过实验展示了 gMatch 的召回效果。

6.1 gMatch 的设计初表

对于糖肽鉴定软件比如 pGlyco 2.0 给出的鉴定结果，通常，人们会将 FDR 指标作为鉴定结果可靠性的衡量指标。我们按照 FDR 这一指标可以将 pGlyco 2.0 的输出结果分为四类。第一类是糖和肽段的 FDR 均不超过给定阈值（比如，0.01）。第二类是糖的 FDR 超过了给定阈值，而肽段的 FDR 没有超过给定阈值。第三类是糖的 FDR 没有超过给定阈值，而肽段的 FDR 超过了给定阈值。第四类是糖和肽段的 FDR 均超过了给定阈值。这里我们关注第三种情形，即糖的 FDR 没有超过给定阈值，而肽段的 FDR 超过了给定阈值。对于这一类输出结果，仅仅从 FDR 的角度来看，可以推测为糖结构鉴定正确，而肽段鉴定错误。然而，有些肽段可能是正确的鉴定结果，但是由于碎裂不是很充分，导致其没有通过 FDR 的阈值限制。比如，A 糖肽和 B 糖肽，这两个糖肽对应的肽段是相同的。但由于带的糖结构不同，A 糖肽的肽段碎裂情况比 B 糖肽略好一点，导致 A 糖肽刚好就过了 FDR 阈值，而 B 糖肽刚好就没有过 FDR 阈值。对于这种情形，由于 A 糖肽和 B 糖肽对应的肽段相同，且二者的碎裂情况相差不大，因此这两个糖肽的肽段谱峰的相似度还是较高的。于是，我们可以通过 A 糖肽谱图和 B 糖肽谱图中肽段对应的谱峰的高相似度来说明 B 糖肽鉴定结果的可靠性，从而提高糖肽鉴定的召回率。

我们从 pGlyco 2.0 软件基于合并库对鼠数据的鉴定结果（实验参数如表 6.1 所示）中挑选了两张谱图，表 6.2 展示了 pGlyco 2.0 给出的这两张谱图的部分输出结果。为了描述方便，我们称谱图 M-Brain-Z-T-1612-01.7596.7596.3.0.dta 的鉴

定结果为谱图 1, 如图 6.1 所示, 谱图 M-Brain-Z-T-1612-01.7398.7398.3.0.dta 的鉴定结果为谱图 2, 如图 6.2 所示。

表 6.1 实验数据的搜索参数

参数类型	参数值
pGlyco 版本	pGlyco 2.0
糖结构库	GlycomeDB 中的 N-糖+人类血清通用结构库, 共 7,884 个包含 NeuGc 的糖结构
蛋白质序列库	uniprot sprot 的鼠数据库, 将序列 N-X-S/T(X≠P)中的 N 替换为 J
酶切类型	胰蛋白酶的特异性酶切
最大遗漏酶切数目	2
固定修饰	Carbamidomethyl[C]
可变修饰	Oxidation[M], Acetyl[ProteinN-term]
母离子质量误差	10 ppm
碎片离子质量误差	20 ppm

表 6.2 pGlyco 2.0 软件基于合并库对鼠数据的鉴定结果

Spec	Peptide	Glycan	PepScore	GlyScore	TotalScore	GlyFDR	PepFDR	TotalFDR
M-Brain-Z-T-1612-01.7596.7596.3.0.dta	GPGIKPJQTSK	7 2 0 0 0	16.37235	61.69239	32.23437	1.05E-15	0.0101	0.0101
M-Brain-Z-T-1612-01.7398.7398.3.0.dta	GPGIKPJQTSK	9 2 0 0 0	18.69688	76.3237	38.86627	1.26E-18	0.0039	0.0039



图 6.1 谱图 1 的匹配情况



图 6.2 谱图 2 的匹配情况

这两张谱图对应的肽段都是 GPGIKPJQTSK, 而谱图 1 的肽段 FDR 大于 0.01, 谱图 2 的肽段 FDR 小于 0.01, 但其实谱图 2 只是比谱图 1 多匹配上了一根谱峰, 对应着谱图 2 中的 y4 离子 (图 6.2 中用粉色方框标记)。如果我们通过这两张实验谱图对应的肽段谱峰的高相似度指标, 则可以把谱图 1 的鉴定结果捞回来。具体来说, 谱图 2 中匹配上肽段 b/y 离子的实验谱峰有 10 根, 而谱图 1 的实验谱峰中有 9 根谱峰在谱图 2 的这 10 根实验肽段谱峰对应的误差窗口内。因此, 这两张谱图的肽段谱峰之间的相似度计算值为 $9/10 = 0.9$ 。

6.2 gMatch 的实现细节

给定 FDR 卡值阈值, 不妨设为 0.01, 我们将 pGlyco 2.0 的鉴定结果中 FDR 值小于 0.01 的谱图作为标注谱图, FDR 值大于等于 0.01 的谱图作为搜索谱图。对于标注谱图, 我们先对该谱图的谱峰的误差窗口建立哈希。即根据谱峰的质荷比及设置的误差计算出该谱峰对应的误差窗口, 然后再把计算误差窗口内的质荷比以一定的倍数 (其倒数值为步长) 哈希到其对应的谱峰 ID。如果某个质荷比在多根谱峰对应的误差窗口范围内, 则这些谱峰一定是连续的。因此, 我们只记录质荷比对应的第一根谱峰的 ID 及其对应的谱峰的总数目。根据标注谱图对应的肽段的序列及其修饰信息计算出理论碎片离子。这里考虑的理论碎片离子类型包括 b 离子, y 离子, 带糖基化位点的 b 离子加上半个单糖, 带糖基化位点的 y 离子加上半个单糖, Y 离子, Y 离子加半个单糖。然后推断理论的肽段碎片离子

与实验谱峰之间的匹配情况。理论碎片离子与实验谱峰匹配的过程中，我们分别计算实验谱峰在不同电荷状态下与理论碎片离子之间的匹配误差。如果某个电荷下的肽段碎片离子同时匹配到多根实验谱峰，我们最终选取使误差达到最小的实验谱峰作为匹配到的谱峰。对于匹配到肽段的理论碎片的实验谱峰，我们会对其对应的离子类型进行标注。

给定一张搜索谱图，我们需要判断搜索谱图是否是糖肽谱图。在删除掉搜索谱图中的一些糖中性离子之后，如果搜索谱图中特征峰 138 为 m/z 为 [100, 200] 范围内强度最高的谱峰，且特征峰 204 为 m/z 为 [170, 270] 范围内强度最高的谱峰，那么该搜索谱图判定为糖肽谱图，如图 6.3 所示。否则，如果上述条件中任意一条不满足，则认为该谱图为非糖肽谱图。如果搜索谱图满足上述两条筛选条件，那么我们进一步根据糖肽谱图中单糖 NeuAc 与单糖 NeuGc 对应的特征峰的相对强度来判断谱图对应的糖结构是否不包含 NeuAc 和 NeuGc。具体来说，特征峰 274/谱峰 204 的值如果小于 0.1，则不考虑包含 NeuAc 的谱峰；谱峰 290/谱峰 204 的值如果小于 0.1，则不考虑包含 NeuGc 的谱峰。

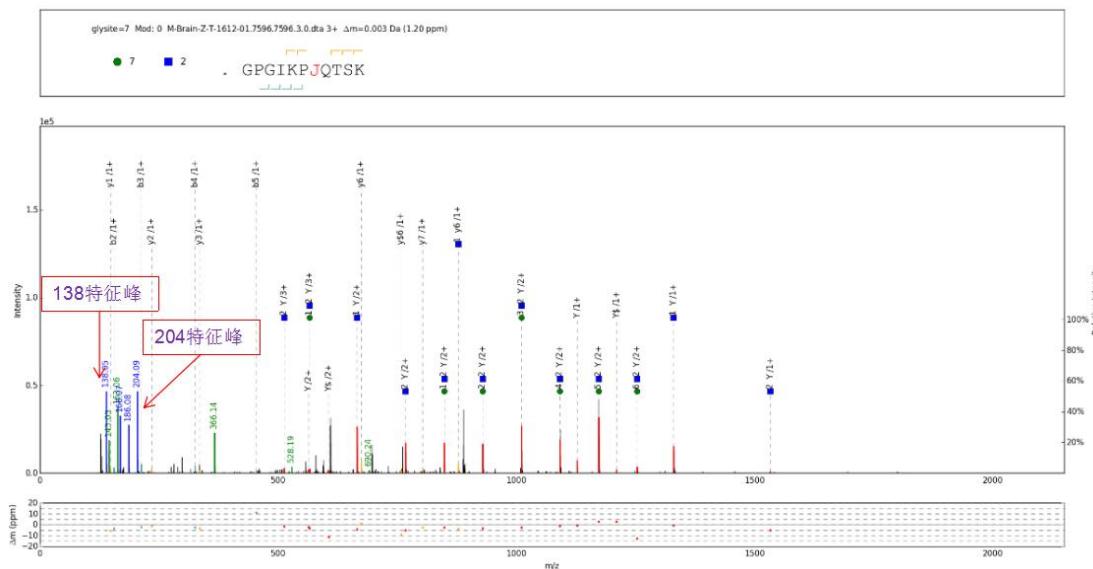


图 6.3 判断一张谱图是否为糖肽谱图的特征峰

给定一张满足糖肽谱图筛选条件的搜索谱图，我们根据特征峰判断这张谱图为糖肽谱图，且根据标注谱图中已标注的肽段谱峰找到与给定的搜索谱图有较高相似度的标注谱图之后，我们还需进一步根据 Y₁~Y₅ 离子（完整肽段+五糖核心的 Y 离子）来筛选可信的标注谱图，其中，五糖核心对应的 Y 离子如图 6.4 所示。具体来说，我们先假设搜索谱图与标注谱图对应着相同的肽段序列，然后根据该肽段序列得到糖肽理论 Y₁~Y₅ 离子，并判断这些离子与搜索谱图实验谱峰

的匹配情况。如果 Y1 离子 (完整肽段+HexNAc) 没有匹配上或者匹配到 Y1~Y5 对应的理论谱峰的数目 (多种电荷状态叠加计算) 小于等于设定的阈值, 比如 2, 那么就将这个标注谱图筛除。需要注意的是, 如果糖库中所有的糖结构都包含五糖核心, 那么给定一张搜索谱图的肽段序列之后, Y1~Y5 这五根谱峰的匹配情况对于糖库中的糖结构来说没有任何区分度。

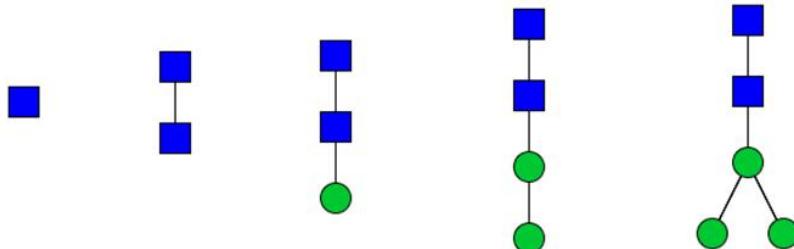


图 6.4 五糖核心对应的 Y 离子

对于满足糖肽谱图筛选条件的搜索谱图与标注谱图, 我们计算搜索谱图与标注谱图的肽段谱峰之间的相似度。如果搜索谱图与标注谱图的肽段谱峰的相似度相当高, 那么我们认为该搜索谱图对应的肽段与标注谱图对应的肽段序列一致, 包括肽段序列上的修饰位点及修饰类型。实际上, 标注谱图的谱峰标注可以采用两种策略。策略一是从实验谱图中删掉 Y 离子匹配的谱峰, 策略二是选择实验谱图中与 pGlyco 2.0 输出结果的肽段匹配上的谱峰, 即 b/y 离子对应的谱峰。如果采取策略一, 那么在进行相似度计算的时候可以将一些由于误差窗口开得太小而没有匹配到的肽段谱峰, 从而加强搜索谱图与标注谱图之间相似度的计算。然而, 剩下的谱峰中包含一些由于质量误差窗口太小而没有匹配上的糖肽谱峰, 这样又会对相似度的计算带来一定的干扰, 因此我们暂时先考虑 b/y 离子对应的谱峰的相似度计算。

进一步, 我们将搜索谱图的谱峰与理论糖组成表进行匹配, 并从中选出搜索谱图匹配到的谱峰的数目较高的糖结构。我们首先基于糖结构库构造了理论糖组成表。我们读取糖结构库中糖结构对应的糖组成, 记录糖结构库中每类单糖对应的最大数目, 记为 $[m_1, m_2, m_3, m_4, m_5]$ 。则理论糖组成表可以看成是集合 $C = \{[n_1, n_2, n_3, n_4, n_5] \mid 0 \leq n_1 \leq m_1, 0 \leq n_2 \leq m_2, 0 \leq n_3 \leq m_3, 0 \leq n_4 \leq m_4, 0 \leq n_5 \leq m_5\}$ 。然后, 我们从糖结构库中读取糖结构对应的线性正则字符串码, 具体的编码方法在 2.3.1 节中有详细的介绍。基于 4.2 节中介绍的基于糖结构的字符串码生成糖结构的子结构的组成的方法, 我们构造了糖结构的理论谱图。接着, 我们根据糖结构对应的理论谱图建立了理论糖组成表到糖结构集合之间的映射关系, 称之为糖结构理论谱峰表, 如图 6.5 所示。

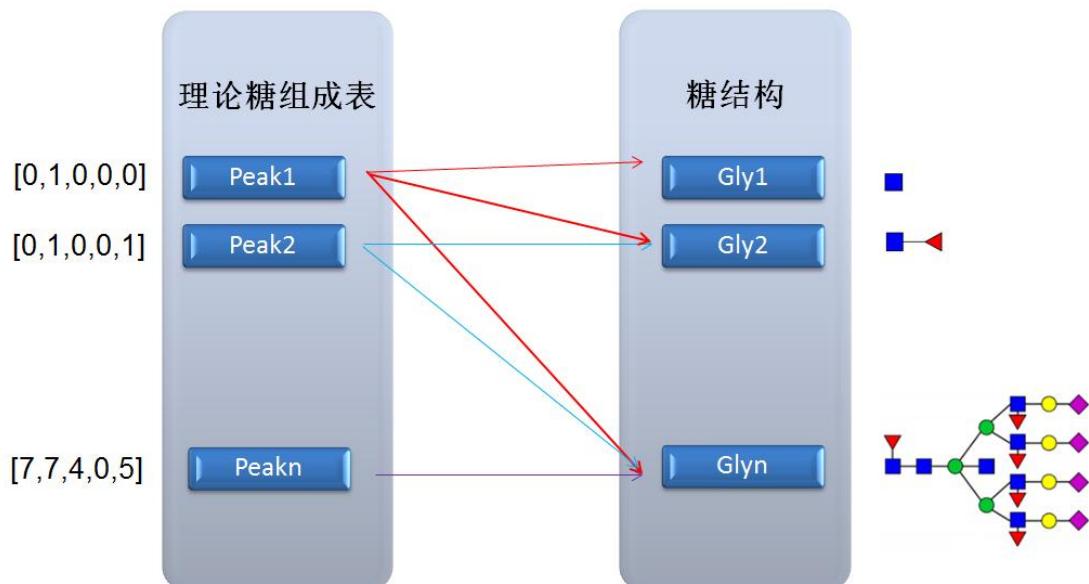


图 6.5 糖结构理论谱峰表

建立好理论糖组成表及糖结构理论谱峰表之后，给定搜索谱图的一根谱峰，我们通过已经匹配好的肽段的质量推测出该谱峰对应的实验糖谱峰的质荷比，然后从理论糖组成表中找到与该谱峰最接近的理论糖谱峰，判断该理论糖谱峰是否在搜索谱图的这根实验谱峰的误差窗口内，最终选出与搜索谱图中的谱峰匹配情况最好的糖结构。

6.3 gMatch 应用实验

gMatch 将 pGlyco 2.0 的搜索结果按照 FDR 值分为搜索谱图和标注谱图，通过搜索谱图与标注谱图的肽段谱峰的相似度计算得到肽段，与糖结构理论谱图匹配得到糖结构。gMatch 的优势在于我们可以不用局限于传统的只与理论肽段离子峰匹配的方法，还可以根据标注谱图进行谱谱匹配。下面我们通过具体的实验展示 gMatch 对鉴定结果的召回效果。测试实验相关参数与 6.1 节中的表 6.1 中的设置一致。

pGlyco 2.0 的搜索结果一共有 15,061 个。其中，标注谱图的数目为 3,602 个，搜索谱图的数目为 11,459 个，gMatch 召回了 7,337 张谱图。我们将 gMatch 的输出结果用 gLabel 软件画图，图 6.6~图 6.8 展示了 gMatch 通过搜索谱图与谱图库中谱图的相似度计算召回的一些鉴定结果，表 6.3 为第一轮中这些谱图通过 pGlyco 2.0 软件给出的鉴定结果以及对应的打分值和 FDR 值。

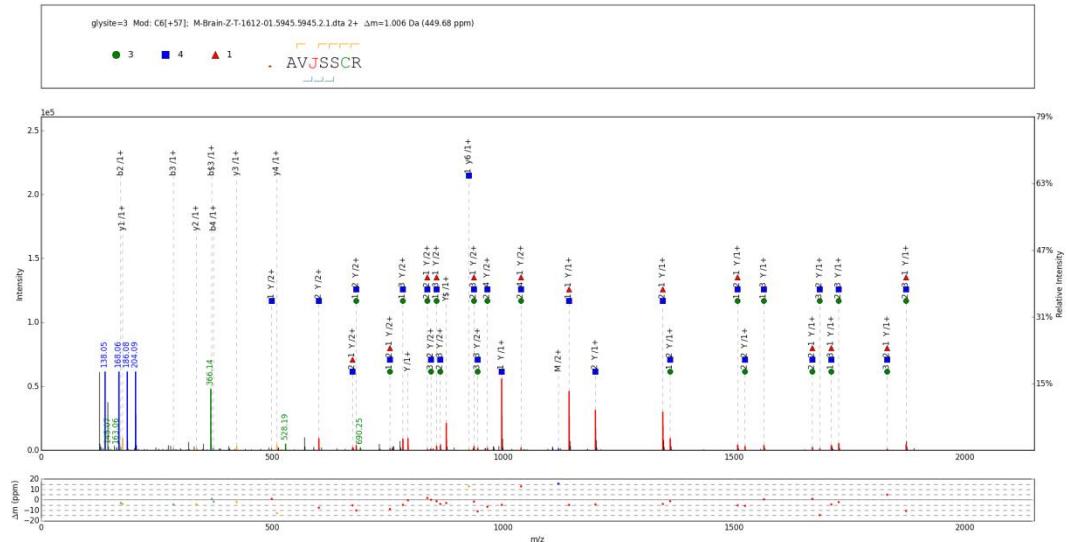


图 6.6 gMatch 召回的谱图 M-Brain-Z-T-1612-01.5945.5945.2.1.dta 对应的鉴定结果

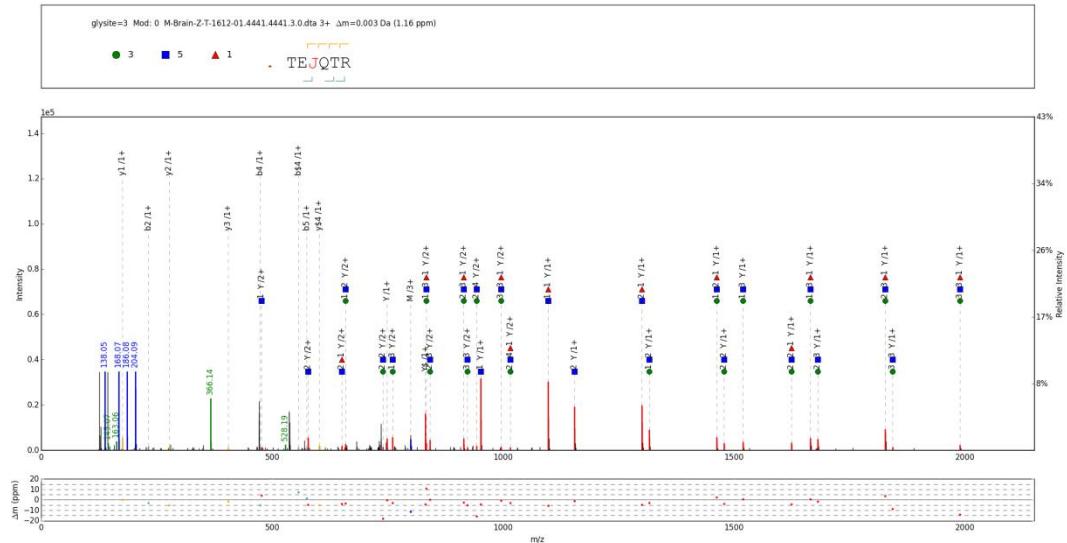


图 6.7 gMatch 召回的谱图 M-Brain-Z-T-1612-01.4441.4441.3.0.dta 对应的鉴定结果

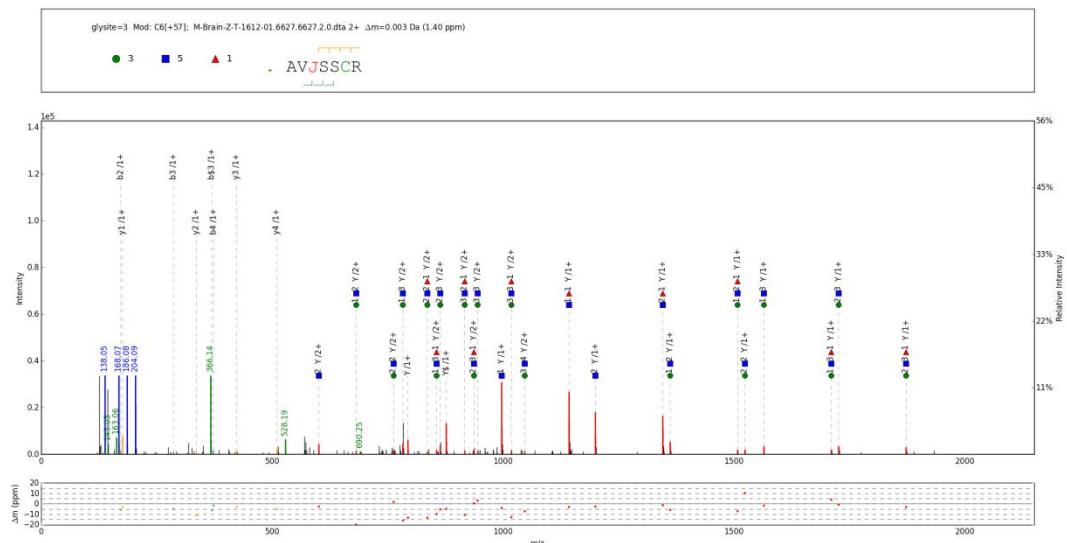


图 6.8 gMatch 召回的谱图 M-Brain-Z-T-1612-01.6627.6627.2.0.dta 对应的鉴定结果

表 6.3 pGlyco 给出的鉴定结果及相应指标值

Spec	Peptide	Glycan	PepScore	GlyScore	TotalScore	GlyFDR	PepFDR	TotalFDR
M-Brain-Z-T-1612-01.5945.5945.2.1.dta	JCSEPYCPLG CSSR	0 2 0 0 1	1.42513	1.90811	1.59417	0.0121	0.5408	0.5408
M-Brain-Z-T-1612-01.4441.4441.3.0.dta	TEJQTR	3 5 0 0 1	11.73586	79.55166	35.47139	3.14E-19	0.0644	0.0644
M-Brain-Z-T-1612-01.6627.6627.2.0.dta	AVJSSCR	3 5 0 0 1	13.46242	66.67638	32.08731	1.04E-16	0.0304	0.0304

假如给定搜索谱图，然后将该搜索谱图与一些标注谱图进行比较。搜索谱图与标注谱图比较的目的就是如果搜索谱图与标注谱图的相似度比较高，那么就判定该搜索谱图对应的肽段与该标注谱图对应的肽段一致。但是，不是所有肽段谱峰相似度高的搜索谱图和标注谱图对应的肽段谱峰的相似度比较高，那么这两个谱图对应的肽段序列就是相同的。我们还需要判断这两张谱图的保留时间。接下来，我们还可以分析如果糖肽谱图对应的肽段序列是相同的，那么这些谱图对应的保留时间服从的规律。由于标注谱图中给出了包含修饰的搜索结果，因此gMatch的鉴定结果中肽段的修饰已经考虑到了。如果允许一定的质量偏移，或许可以用来分析糖结构的修饰鉴定。

第七章 结束语

7.1 全文总结

本文的第一章中我们首先介绍了蛋白质的 N-糖基化修饰和 O-糖基化修饰，然后介绍了基于串联质谱技术的蛋白质鉴定及糖肽鉴定，其中，基于质谱技术的蛋白质鉴定主要分为数据库搜索、从头测序和肽序列标签查询这三类方法，基于质谱技术的糖肽鉴定主要分为对完整糖肽的鉴定和对糖和肽段依次进行鉴定两种方法。随着串联质谱技术的发展，人们对蛋白质糖基化的鉴定也有了可观的进展。MIT 创新杂志于 2003 年指出糖组学是能够改变未来的十大技术之一[77]。糖组学的发展过程中，依赖于数据库搜索的质谱串联技术还是当前糖肽鉴定的主流方法，因此，糖结构数据库的建立尤为重要。建立糖结构数据库首先需要确定的就是糖结构的存储格式。基因和蛋白质是线性结构，在计算机中可以直接用线性字符串表示，而计算机很难从糖的化学表示中直接提取出单糖和连接信息，因此我们不能直接把糖的化学表示作为一种存储格式。于是，数据库构建者纷纷提出了新的糖结构的存储格式，我们对糖结构的表示方法进行了调研，并介绍了当前公开的糖结构。

本文的第二章介绍了糖结构同构判断的方法。枚举理论糖结构的过程中，可能会生成一些同构的糖结构，考虑到同构的糖结构对应着完全相同的拓扑结构，我们需要对枚举出的糖结构去掉同构冗余。给定 n 个糖结构，一种直观的判断方法是对这些糖结构进行两两比较。然而，这种方法会对同一个糖结构进行多次遍历，导致糖结构同构判定的效率较低。为了加快糖结构的同构判定速度，我们将糖结构的拓扑信息从叶节点往上汇总到根节点，并记录相应的哈希值。哈希方法保证同构的糖结构哈希对应着相同的哈希值，也就是说，哈希值不同的糖结构一定不同构，因此，对于哈希值不同的糖结构我们可以直接判定这些糖结构互不同构，从而一定程度上减少了糖结构的遍历次数。哈希方法虽然提高了糖结构判断的效率，但是哈希值相等的糖结构可能并不同构。为了实现同构糖结构到哈希值之间的一一映射，我们进一步调研了树结构的同构判定方法，并基于这些方法提出了一种新的糖结构的编码方法。本章的创新之处包括提出了一种新的且既方便计算机存储又易于人工解码的糖结构编码方法，将树形糖结构线性化，使得糖结构之间的去冗余如同肽段之间的去冗余一样方便。

本文的第三章和第四章中，我们旨在构造更加全面合理的理论 N-糖库，从而为依赖于数据库搜索的 N-糖肽谱图的鉴定提供更好的保障。主要研究内容包括枚举给定节点数目的糖结构、糖结构去同构冗余、生成糖结构的 Y 离子、糖结构中 NeuAc 与 NeuGc 的替换以及利用败者树归并解决由于内存不足而使得枚举规模受限的问题等。具体包括：1) 给定 n 个糖结构，找出这 n 个糖结构中互为同构的糖结构，并对于互为同构的糖结构，只保留其中一个；2) 实现糖结构的扩展算法，即将所有合理且非冗余的糖结构中的 NeuAc 替换为 NeuGc；3) 按照一定的生物规则 (GP Finder 规则[75]) 及单糖的分支数目限制过滤掉不合理的糖结构，有效地缩减糖库的规模；4) 给定一个糖结构，生成这个糖结构的所有包含糖结构的根节点的子结构的组成，一方面可以用来判断糖结构是否符合 GP Finder 规则，另一方面可以为 N-糖肽鉴定提供理论谱图库以便于匹配打分；5) 枚举生成的糖结构的数目达到一定阈值之后，我们就把糖结构按照其字符串码对应的哥德尔码值非降序输出到磁盘文件中，如果磁盘文件的数目大于 1，则利用败者树将不同磁盘文件中的糖结构归并，对于哥德尔码值相同的糖结构，我们只保留其中一个，最终将互不同构的糖结构按照其对应的哥德尔码值非降序输出到一个磁盘文件中。

第三章中我们是基于糖结构的邻接表构建理论糖库，而第四章中我们基于新提出的糖结构的线性正则编码方法，实现了根据糖结构的字符串码往糖结构中加入新的节点、调整糖结构的正则顺序从而构造新的正则糖结构、计算糖结构中每个节点的分支数目以及生成糖结构的子结构等算法。我们分别用基于糖结构的字符串码和基于糖结构的邻接表构造理论糖结构的方法枚举出 6~10 个节点的糖结构，并对齐了这两种不同枚举算法的枚举结果。第四章中的创新之处包括设计了一种新的根据糖结构的字符串码生成糖结构的子结构的组成的算法，把原始的需要 600+ 秒的程序通过算法方面的优化改进为只需要 5 秒，又通过实现方面的优化把 5 秒加速到 2 秒等。

本文的第五章中，我们通过对枚举库中等价类的分析，发现了 pGlyco 2.0 直接用于鉴定枚举库的问题所在，通过对枚举库的糖结构进行归并使得 pGlyco 2.0 能够基于枚举库实现有效的糖肽鉴定，且验证了基于枚举库可以有助于发现潜在的新的 N-糖结构。我们将理论 N-糖结构库用于 N-糖肽鉴定时，如果糖库规模太大将不可避免地引入更多的假阳。因此，我们提供了两种缩减理论 N-糖结构库的策略，策略一是根据对 GlycomeDB 中 N-糖结构的分支数目的统计，设置了不同单糖类型的最大分支数目阈值，筛除掉单糖分支数目超过给定阈值的糖结构，策略二是筛除掉不符合 GP Finder 规则的糖结构，从而使枚举库更加适用于依赖于数据库搜索的糖肽鉴定。

本文的第六章介绍了我们基于糖结构的线性正则编码方法设计了搜索引擎 gMatch, 该搜索引擎可以有效地提高糖肽谱图鉴定结果的召回率。我们通过分析 pGlyco 2.0 的鉴定结果, 发现一些糖肽谱图的鉴定结果对应着相同的肽段序列, 这些谱图的肽段谱峰与肽段序列的匹配情况都非常好, 但其中一部分谱图过了 FDR 阈值, 而另一些谱图由于少匹配了几根实验谱峰而没有通过 FDR。如果我们比较这些实验谱图能匹配上肽段序列的谱峰, 会发现这些谱峰的相似度很高。基于上述观察及分析, 我们在 pGlyco 2.0 给出的第一轮的鉴定结果的基础上进行了第二轮的搜索匹配, 通过实验谱图之间的相似度召回了更多可信的鉴定结果。

7.2 研究展望

对于本文中介绍的一些工作, 目前想到的一些完善方向如下:

(1) 本文中基于糖结构的线性正则字符串码提出了一种新的构建理论 N-糖结构库的方法, 虽然该方法相比于基于糖结构的邻接表构建理论 N-糖结构库的方法来说时间效率和空间效率两方面都有一定的提升, 但还是有改进的空间, 进一步可能的改进方向包括: (a) 或许可以尝试用(D,L)糖结构编码方法来枚举糖结构, 其中, D 表示节点的深度, L 表示节点对应的标号值; (b) 为了尽量减少垂直冗余的生成, 可以尝试在枚举的过程中加入的叶子节点的标号值一定不小于当前糖结构中叶子节点的最大标号值; (c) 可以尝试采用其它正则顺序进行枚举, 也可以尝试采用其它外包方法。

(2) 给定实验糖肽谱图, 我们可以直接将线性表示的糖结构粘贴到序列框中, 并根据糖结构的线性表示高效地生成该糖结构的碎片离子, 从而方便地构造糖结构对应的理论谱图以与实验谱图进行匹配。将来可以考虑基于糖结构的线性正则字符串码开发面向于糖结构的类似于 pLabel[72]的谱图标注软件。

(3) gMatch 由于时间有限, 还有不少可以完善的地方。包括: (a) 当前 gMatch 只是在一定的阈值设置下召回了一定量的鉴定结果, 但并不是所有召回的鉴定结果都是可信的, 因此, 我们需要进一步对召回的谱图进行评价, 评价方法可以考虑 FDR 指标或者机器学习模型; (b) 分析相同肽段序列不同糖型对应的糖肽谱图的保留时间服从的规律。给定一张搜索谱图, 可以通过保留时间的相关规律来筛选出一些标注谱图进行匹配; (c) 可以分析一下糖结构核心 Y 离子的强度分布规律。

参考文献

1. Shental-Bechor, D. and Y. Levy, *Effect of glycosylation on protein folding: a close look at thermodynamic stabilization*. Proceedings of the National Academy of Sciences, 2008. **105**(24): p. 8256-8261.
2. Solá, R.J. and K. Griebenow, *Glycosylation of therapeutic proteins*. BioDrugs, 2010. **24**(1): p. 9-21.
3. Kim, Y.J. and A. Varki, *Perspectives on the significance of altered glycosylation of glycoproteins in cancer*. Glycoconjugate journal, 1997. **14**(5): p. 569-576.
4. Brockhausen, I., J. Schutzbach, and W. Kuhns, *Glycoproteins and their relationship to human disease*. Cells Tissues Organs, 1998. **161**(1-4): p. 36-78.
5. Buskas, T., P. Thompson, and G.-J. Boons, *Immunotherapy for cancer: synthetic carbohydrate-based vaccines*. Chemical Communications, 2009(36): p. 5335-5349.
6. Hakomori, S., *Glycosylation defining cancer malignancy: new wine in an old bottle*. Proceedings of the National Academy of Sciences, 2002. **99**(16): p. 10231-10233.
7. Apweiler, R., H. Hermjakob, and N. Sharon, *On the frequency of protein glycosylation, as deduced from analysis of the SWISS-PROT database*. Biochimica et Biophysica Acta (BBA)-General Subjects, 1999. **1473**(1): p. 4-8.
8. Gao, H.Y., *Generation of asparagine-linked glycan structure databases and their use*. Journal of the American Society for Mass Spectrometry, 2009. **20**(9): p. 1739-1742.
9. Pan, S., et al., *Mass spectrometry based glycoproteomics—from a proteomics perspective*. Molecular & Cellular Proteomics, 2011. **10**(1): p. R110. 003251.
10. 吴建强. 基于质谱数据的N糖蛋白鉴定算法研究与软件开发. 中国科学院大学:硕士学位论文, 2015.
11. 曾文锋. 基于生物质谱技术的规模化完整糖肽鉴定方法研究. 中国科学院大学:博士学位论文, 2016.
12. 钱小红 and 贺福初, *蛋白质组学: 理论与方法*. M]. 北京: 科学出版社, 2003.
13. Gavel, Y. and G. von Heijne, *Sequence differences between glycosylated and non-glycosylated Asn-X-Thr/Ser acceptor sites: implications for protein engineering*. Protein engineering, 1990. **3**(5): p. 433-442.
14. Chen, M.M., K.J. Glover, and B. Imperiali, *From peptide to protein: comparative analysis of the substrate specificity of N-linked glycosylation in C. jejuni*. Biochemistry, 2007. **46**(18): p. 5579-5585.
15. Joshi, H.J., et al., *GlycoViewer: a tool for visual summary and comparative*

- analysis of the glycome.* Nucleic acids research, 2010. **38**(suppl 2): p. W667-W670.
16. He, L., *Algorithms for Characterizing Peptides and Glycopeptides with Mass Spectrometry*. 2013.
17. Zeng, W.F., et al., *Trends in Mass Spectrometry-Based Large-scale N-Glycopeptides Analysis*. Progress in Biochemistry and Biophysics, 2016. **43**(6): p. 550-562.
18. Zhu, Z., et al., *GlycoPep Detector: a tool for assigning mass spectrometry data of N-linked glycopeptides on the basis of their electron transfer dissociation spectra*. Analytical chemistry, 2013. **85**(10): p. 5023-5032.
19. He, L., et al., *GlycoMaster DB: software to assist the automated identification of N-linked glycopeptides by tandem mass spectrometry*. Journal of proteome research, 2014. **13**(9): p. 3881-3895.
20. Doubet, S. and P. Albersheim, *Letter to the glyco-forum CarbBank*. Glycobiology, 1992. **2**(6): p. 505-505.
21. Lütteke, T., et al., *GLYCOSCIENCES. de: an Internet portal to support glycomics and glycobiology research*. Glycobiology, 2006. **16**(5): p. 71R-81R.
22. Ogata, H., et al., *KEGG: Kyoto encyclopedia of genes and genomes*. Nucleic acids research, 1999. **27**(1): p. 29-34.
23. Kanehisa, M. and S. Goto, *KEGG: kyoto encyclopedia of genes and genomes*. Nucleic acids research, 2000. **28**(1): p. 27-30.
24. Hashimoto, K., et al., *KEGG as a glycome informatics resource*. Glycobiology, 2006. **16**(5): p. 63R-70R.
25. Raman, R., et al., *Advancing glycomics: implementation strategies at the consortium for functional glycomics*. Glycobiology, 2006. **16**(5): p. 82R-90R.
26. Toukach, P., *Bacterial carbohydrate structure database version 3*. GLYCOCONJUGATE JOURNAL, 2009. **26**(7): p. 154-154.
27. Maeda, M., et al., *JCGGDB: Japan Consortium for Glycobiology and Glycotechnology Database*. Glycobiology, 2015: p. 161-179.
28. von der Lieth, C.-W., et al., *EUROCarbDB: an open-access platform for glycoinformatics*. Glycobiology, 2011. **21**(4): p. 493-502.
29. Campbell, M.P., et al., *UniCarbKB: building a knowledge platform for glycoproteomics*. Nucleic Acids Res, 2014. **42**(Database issue): p. D215-21.
30. Cooper, C.A., et al., *GlycoSuiteDB: a new curated relational database of glycoprotein glycan structures and their biological sources*. Nucleic Acids Research, 2001. **29**(1): p. 332-335.
31. Cooper, C.A., et al., *GlycoSuiteDB: a curated relational database of glycoprotein glycan structures and their biological sources. 2003 update*. Nucleic acids research, 2003. **31**(1): p. 511-513.
32. Ranzinger, R., et al., *GlycomeDB—integration of open-access carbohydrate structure databases*. BMC bioinformatics, 2008. **9**(1): p. 1.
33. Ranzinger, R., et al., *Glycome-DB.org: a portal for querying across the*

- digital world of carbohydrate sequences.* Glycobiology, 2009. **19**(12): p. 1563-1567.
34. Ranzinger, R., et al., *GlycomeDB—a unified database for carbohydrate structures.* Nucleic acids research, 2011. **39**(suppl 1): p. D373-D376.
35. Aoki-Kinoshita, K., et al., *GlyTouCan 1.0—The international glycan structure repository.* Nucleic acids research, 2016. **44**(D1): p. D1237-D1242.
36. Ranzinger, R., et al., *GlycoRDF: an ontology to standardize glycomics data in RDF.* Bioinformatics, 2015. **31**(6): p. 919-925.
37. 杨钊, 姜伟业, and 陈闻天, 糖生物信息学数据库. 中国生物化学与分子生物学报, 2015. **31**(3): p. 257-263.
38. Campbell, M.P., et al., *Toolboxes for a standardised and systematic study of glycans.* BMC bioinformatics, 2014. **15**(1): p. 1.
39. Frank, M. and S. Schloissnig, *Bioinformatics and molecular modeling in glycobiology.* Cellular and molecular life sciences, 2010. **67**(16): p. 2749-2772.
40. Lütteke, T., *The use of glycoinformatics in glycochemistry.* Beilstein journal of organic chemistry, 2012. **8**(1): p. 915-929.
41. Ranzinger, R. and W.S. York. *Glyco-Bioinformatics today (August 2011)—Solutions and Problems.* in *2nd Beilstein Symposium on Glyco-Bioinformatics, Cracking the Sugar Code by Navigating the Glycospace. Potsdam, Germany.* 2012.
42. Hang, L., et al., *Bioinformatic Resources and Methods for Glycoproteomics.* PROGRESS IN BIOCHEMISTRY AND BIOPHYSICS, 2016. **43**(9): p. 910-918.
43. 王继峰, et al., 完整糖肽质谱解析的研究进展. 质谱学报, 2014. **35**(2): p. 108-117.
44. Eavenson, M., et al., *Qrator: A web-based curation tool for glycan structures.* Glycobiology, 2015. **25**(1): p. 66-73.
45. Hizal, D.B., et al., *Glycoproteomic and glycomic databases.* Clinical proteomics, 2014. **11**(1): p. 1.
46. Bohne-Lang, A., et al., *LINUCS: linear notation for unique description of carbohydrate sequences.* Carbohydrate research, 2001. **336**(1): p. 1-11.
47. Aoki, K.F., et al., *KCaM (KEGG Carbohydrate Matcher): a software tool for analyzing the structures of carbohydrate sugar chains.* Nucleic acids research, 2004. **32**(suppl 2): p. W267-W272.
48. Herget, S., et al., *GlycoCT—a unifying sequence format for carbohydrates.* Carbohydrate research, 2008. **343**(12): p. 2162-2171.
49. Doubet, S., et al., *The complex carbohydrate structure database.* Trends in biochemical sciences, 1989. **14**(12): p. 475-477.
50. Toukach, P.V., *Bacterial carbohydrate structure database 3: principles and realization.* Journal of chemical information and modeling, 2010. **51**(1): p. 159-170.

51. Banin, E., et al., *A novel linear code nomenclature for complex carbohydrates*. Trends in Glycoscience and Glycotechnology, 2002. **14**(77): p. 127-137.
52. Aoki-Kinoshita, K.F., *Glycome Informatics: Methods and Applications*. 2009: CRC Press.
53. Zeng, W.-F., et al., *pGlyco: a pipeline for the identification of intact N-glycopeptides by using HCD-and CID-MS/MS and MS3*. Scientific reports, 2016. **6**.
54. Liu, M., et al., *Efficient and accurate glycopeptide identification pipeline for high-throughput site-specific N-glycosylation analysis*. Journal of proteome research, 2014. **13**(6): p. 3121-3129.
55. Cheng, K., et al., *Large-scale characterization of intact N-glycopeptides using an automated glycoproteomic method*. Journal of proteomics, 2014. **110**: p. 145-154.
56. Chandler, K.B., et al., *Exploring site-specific N-glycosylation microheterogeneity of haptoglobin using glycopeptide CID tandem mass spectra and glycan database search*. Journal of proteome research, 2013. **12**(8): p. 3652-3666.
57. Toghi Eshghi, S., et al., *GPQuest: a spectral library matching algorithm for site-specific assignment of tandem mass spectra to intact N-glycopeptides*. Analytical chemistry, 2015. **87**(10): p. 5181-5188.
58. Park, G.W., et al., *Integrated GlycoProteome Analyzer (I-GPA) for Automated Identification and Quantitation of Site-Specific N-Glycosylation*. Scientific reports, 2016. **6**.
59. Hopcroft, J.E., J.D. Ullman, and A. Aho, *The design and analysis of computer algorithms*. 1975, Addison-Wesley.
60. Chi, Y., Y. Yang, and R.R. Muntz, *Canonical forms for labelled trees and their applications in frequent subtree mining*. Knowledge and Information Systems, 2005. **8**(2): p. 203-234.
61. Luccio, F., et al., *Bottom-up subtree isomorphism for unordered labeled trees*. International Journal of Pure and Applied Mathematics, 2004. **38**(3): p. 325-343.
62. Kreher, D.L. and D.R. Stinson, *Combinatorial algorithms: generation, enumeration, and search*. CRC press, 1999. **3**: p. 5.
63. Lindell, S. *A logspace algorithm for tree canonization*. in *Proceedings of the twenty-fourth annual ACM symposium on Theory of computing*. 1992. ACM.
64. Zaki, M.J., *Efficiently mining frequent trees in a forest: Algorithms and applications*. IEEE transactions on knowledge and data engineering, 2005. **17**(8): p. 1021-1035.
65. Chi, Y., et al., *Frequent subtree mining—an overview*. Fundamenta Informaticae, 2005. **66**(1-2): p. 161-198.
66. Jiménez, A.d., F. Berzal, and J.-C. Cubero, *Frequent tree pattern mining: A survey*. Intelligent Data Analysis, 2010. **14**(6): p. 603-622.

-
67. Tan, H., et al. *IMB3-Miner: mining induced/embedded subtrees by constraining the level of embedding.* in *Pacific-Asia Conference on Knowledge Discovery and Data Mining*. 2006. Springer.
 68. Chi, Y., Y. Yang, and R.R. Muntz. *HybridTreeMiner: An efficient algorithm for mining frequent rooted trees and free trees using canonical forms.* in *Scientific and Statistical Database Management, 2004. Proceedings. 16th International Conference on*. 2004. IEEE.
 69. Jiménez, A., F. Berzal, and J.-C. Cubero, *POTMiner: mining ordered, unordered, and partially-ordered trees.* Knowledge and information systems, 2010. **23**(2): p. 199-224.
 70. Hadzic, F., H. Tan, and T.S. Dillon. *UNI3-efficient algorithm for mining unordered induced subtrees using TMG candidate generation.* in *Computational Intelligence and Data Mining, 2007. CIDM 2007. IEEE Symposium on*. 2007. IEEE.
 71. Li, Y., et al., *Speeding up tandem mass spectrometry based database searching by peptide and spectrum indexing.* Rapid Communications in Mass Spectrometry, 2010. **24**(6): p. 807-814.
 72. Wang, L.h., et al., *pFind 2.0: a software package for peptide and protein identification via tandem mass spectrometry.* Rapid Communications in Mass Spectrometry, 2007. **21**(18): p. 2985-2991.
 73. Pompach, P., et al., *Semi-automated identification of N-Glycopeptides by hydrophilic interaction chromatography, nano-reverse-phase LC-MS/MS, and glycan database search.* Journal of proteome research, 2012. **11**(3): p. 1728-1740.
 74. Wu, S.-W., et al., *Sweet-Heart—an integrated suite of enabling computational tools for automated MS2/MS3 sequencing and identification of glycopeptides.* Journal of proteomics, 2013. **84**: p. 1-16.
 75. Strum, J.S., et al., *Automated assignments of N-and O-site specific glycosylation with extensive glycan heterogeneity of glycoprotein mixtures.* Analytical chemistry, 2013. **85**(12): p. 5666-5675.
 76. Kronewitter, S.R., et al., *The development of retrosynthetic glycan libraries to profile and classify the human serum N - linked glycome.* Proteomics, 2009. **9**(11): p. 2986-2994.
 77. GLYCOSCIENCES.de. <http://www.glycosciences.de/>.

致 谢

感谢三年前贺老师在我的执念下收留了我，才让后续的故事成为了可能。记得瑜伽课结束时，我们会双手合十，一起诵读三句话“感谢瑜伽，感谢自己的坚持，更要感谢周围陪你一起练瑜伽的人”。这里借以改编为“感谢科研，感谢自己的坚持，更要感谢周围陪你一起搞科研的人”。

感谢贺老师在春节期间还在帮我查找糖结构同构判定相关资料，思考并分享了这个问题的新进展。感谢贺思敏老师给予我们真诚的关怀与温暖。感谢贺老师在我们取得进步时给予的真诚的鼓励，在发现我们的小毛病时及时指出，让我们每个人都越来越好。

感谢建强师兄留下的宝贵的糖相关的工作，在繁忙的工作期间还抽出时间批注我的报告。感谢文锋师兄带我进入糖领域，耐心地给我普及糖相关的背景知识，带着我读代码，提出了一个又一个宝贵的研究思路，跟我讨论很多有意思的问题。

感谢孙瑞祥老师的帮助。孙老师博学健谈，乐观积极。还记得在大连开会期间，孙老师每天早上 5 点都会去附近的大海游泳，闭幕式的发言巧妙地将大会开闭幕时间与 FDR 联系起来，用专业知识道出了生活中的见闻，让人印象深刻。感谢邬龙师兄的耐心答疑，在邬龙师兄的帮助下，我的入门训练顺利了许多。感谢刘寒化老师和汪洋老师。刘寒化老师在生活方面给予了我们及时周到的关怀，汪洋老师像个活泼的大姐姐，所里的肚皮舞课、瑜伽课上总能看见她，主动请缨当我们的游泳教练，牺牲掉陪伴孩子的时间来教我们游泳，非常感动。感谢超哥在我们进组之前就一直给予着兄长般的关怀。感谢浩哥树立了很好的学术榜样，感谢在瑞敏的帮助下，我这个刚进组时连 zoj 里最简单的加法题都过不了的小白开始了解更多计算机相关知识。感谢小皓哥和文婧带来的欢乐与温暖，感谢钊伟分享的一个又一个精彩的故事。感谢各位师弟师妹的陪伴。秀南乐观活泼，镇霖沉着踏实，润乾学识广博，振振聪明执着，郑好甜美文静，王曦博学多才，心仪勤奋可爱。

感谢在雁栖湖我们五姐妹——钊伟（大姐）、瑞敏（二姐）、我（三姐）、蓝青（四姐）、文婧（小妹）——一起度过一个个难忘的生日，一起从所里坐火车回到雁栖湖，在村里的小路上拍星星、看兵哥哥。感谢在雁栖湖遇到的一起跳舞的小伙伴李林玥、荆丽华、李超、艾娟、付晶晶、刘月等。感谢带着我为班级排舞，给予我莫大信任与支持的赵浩钧。感谢温暖可爱陪伴我三年带给我无数次感动的赵连鹤。感谢跟我一起玩电钢辨音能力很强的石翔，一起玩尤克里里弹唱俱

佳的李瑜，还有大笔一挥给我一张乐谱、唱歌好听专业、走嘻哈风、一年读上百本课外书的杨帆。感谢教我大提琴的杨子清、王凤，从持琴姿势、持弓技巧到音阶指法，都非常耐心地指导纠正着。你们让我的生活更加多姿多彩，也留下了更多美好的回忆。

感谢我的爸爸妈妈，感谢你们给了我足够多的自主权，让我能够从心所欲地做自己喜欢的事情。感谢你们对我的呵护与宠爱，却也不忘在适当的时候敲打我。感谢你们督促我改掉坏毛病时的果断与严厉，让我完成一次又一次的蜕变。感谢你们在我遇到较大坎坷想打退堂鼓的时候坚定地支持着我鼓励着我，给予我无限的力量与勇气。

作者简介

姓名：张晓今 性别：女 出生日期：1992.04.21 籍贯：湖北襄阳

教育经历

2014.9 – 2017.7	中国科学院计算技术研究所	硕士	计算机应用技术
2010.9 – 2014.7	山东大学（威海）	本科	统计、金融

【攻读硕士学位期间发表的论文】

- [1] 曾文锋，张扬，刘铭琪，吴建强，张晓今，杨皓，刘超，迟浩，张昆，孙瑞祥，杨茺原，贺思敏. N-糖肽的规模化质谱解析方法进展. 生物化学与生物物理进展, 2016, 43(6), 550-562.
- [2] Xiao-Jin Zhang, Wen-Feng Zeng, Jian-Qiang Wu, Yang Zhang, Ming-Qi Liu, Pan Fang, Chao Peng, Catherine C. L. Wong, Rui-Xiang Sun, Peng-Yuan Yang, Si-Min He. Construction of N-glycan databases based on a linear canonical representation of N-glycans.(under review)

【攻读硕士学位期间参加的科研项目】

- [1] 自然科学基金仪器专项“糖蛋白结构解析串联质谱分析装置”(21227805),
2013年1月 – 2017年12月
- [2] 国家重点研发计划重点专项“蛋白质组精准鉴定搜索引擎及技术体系”
(2016YFA0501300), 2016年7月 – 2021年6月

【攻读硕士学位期间申请的专利】

- [1] 张晓今, 曾文锋, 吴建强, 孙瑞祥, 贺思敏, 基于糖结构的正则表达式的N-糖结构库构建方法, 申请日: 2017年5月5日

【攻读硕士学位期间申请的软件著作权】

- [1] 张晓今, 曾文锋, 吴建强, 孙瑞祥, 贺思敏, 蛋白质质谱分析的N-糖结构库构建软件, 申请日: 2017年5月5日