Supporting Information

Open-pNovo: De Novo Peptide Sequencing with Thousands of Protein Modifications

Hao Yang[1,2], Hao Chi[1*], Wen-Jing Zhou[1,2], Wen-Feng Zeng[1,2], Kun He[1,2], Chao Liu[1], Rui-Xiang Sun[1], Si-Min He[1,2*]

[1]Key Lab of Intelligent Information Processing of Chinese Academy of Sciences (CAS), Institute of Computing Technology, CAS, Beijing 100190, China

[2]University of Chinese Academy of Sciences, Beijing 100049, China

1. The pseudo code of pDAG-I.

Algorithm: finding the *k* longest normal paths and modified paths in a directed acyclic graph.

Input: DAG(*V, E1, E2*), *V* is the set of vertices, *E1* is the set of normal edges, *E2* is the set of modified edges. $u, v, w \in V$, $edge1(u,w) \in E1, edge2(v,w) \in E2$. *s* is the source vertex and *t* is the destination vertex. The weight of $edge1(u,w)$ is $weight1(u,w)$ and the weight of $edge2(v,w)$ is $weight2(v,w)$.

Output: the *k* longest normal paths and the *k* longest modified paths from *s* to *t*.

Function: DPKlongestPath(*V, E1, E2*). It can find the *k* longest normal paths and the *k* longest modified paths from *s* to *t* and save these paths in $P1[t]$ and $P2[t]$ respectively.

$InvE1[v]$ records all preceding vertices of v whose edges (u,v) are normal edges.

$InvE2[v]$ records all preceding vertices of v whose edges (u,v) are modified edges.

$P1[v]$ records the *k* longest normal paths from *s* to *v*.

$P2[v]$ records the *k* longest modified paths from *s* to *v*.

Procedure DPKlongestPath(*V, E1, E2*)

```
1   sort all vertices including the source vertex and the destination vertex by mass in ascending
order
2   while (there are vertices which have not been computed)
3       get a vertex u which has the smallest mass in all vertices which have not been computed
4       for each v ∈ InvE1[u]
5           insert all P1[v]+ edge1(v,u) to P1[u]
6           insert all P2[v]+ edge1(v,u) to P2[u]
7       for each v ∈ InvE2[u]
8           insert all P1[v]+ edge2(v,u) to P2[u]
9       P1[u].sort();P2[u].sort();
10      only retain top K in P1[u]
11      only retain top K in P2[u]
12  return P1[t] and P2[t]
```

2. The pseudo code of pDAG-II.

Algorithm: Finding the *k* longest normal paths and modified paths in a directed acyclic graph.
Input: DAG(*V, E1, E2*), *V* is the set of vertices, *E1* is the set of normal edges, *E2* is the set of modified edges. $u, v, w \in V$, $edge1(u,w) \in E1, edge2(v,w) \in E2$. *s* is the source vertex and *t* is the destination vertex. The weight of $edge1(u,w)$ is $weight1(u,w)$, the weight of $edge2(v,w)$ is $weight2(v,w)$.
Output: the *k* longest normal paths and the *k* longest modified paths from *s* to *t*.
Function: DPKlongestPath(*V, E1, E2*). To find the *k* longest normal paths and the *k* longest modified paths from *s* to *t* and store these paths to $P1$ and $P2$ respectively.
$InvE1[v]$ records all preceding vertices whose edges (u, v) are normal edges.
$InvE2[v]$ records all preceding vertices whose edges (u, v) are modified edges.
$P1[v, i]$ is the structure that records the i-th longest normal path and its weight from *s* to *v*.
$P2[v, i]$ is the structure that records the i-th longest modified path and its weight from *s* to *v*.

Procedure DPKLongestPath(*V, E1, E2*)
1 **sort** all vertices by mass in ascending order
2 **while** (there are vertices which have not been computed)
3 get a vertex u which has the smallest mass in all vertices which have not been computed
4 insert each $P1[v_d, 0]$ ($\forall v_d \in InvE1[u]$) to a loser tree $T1$
5 **initialize** $i_d = 0$ for every $P1$
6 **while** (the size of $P1[u]$ is less than or equal to *k*)
7 $p1 = P1[v_j, i_j] = T1$.pop_up_max_weight()
8 update $i_j = i_j + 1$
9 insert $P1[v_j, i_j]$ to $T1$
10 **if** $p1$ doesn't contain the cognate vertex of u
11 **then** insert $p1 + edge1(v_j, u)$ to $P1[u]$
12 insert each $P2[v_d, 0]$($\forall v_d \in InvE1[u]$) and $P1[v'_d, 0]$($\forall v'_d \in InvE2[u]$) to a loser tree $T2$
13 **initialize** $i_d = 0$ for every $P2$ and $i'_d = 0$ for every $P1$
14 **while** (the size of $P2[u]$ is less than or equal to *k*)
15 $p2 = T2$.pop_up_max_weight()
16 **if** $p2 \in P2$, such that $p2 = P2[v_j, i_j]$
17 update $i_j = i_j + 1$
18 $edge = edge1(v_j, u)$
19 insert $P2[v_j, i_j]$ to $T2$
20 **else if** $p2 \in P1$, such that $p2 = P1[v'_j, i'_j]$
21 update $i'_j = i'_j + 1$
22 $edge = edge2(v'_j, u)$
23 insert $P1[v'_j, i'_j]$ to $T2$
24 **if** $p2$ doesn't contain the cognate vertex of u
25 **then** insert $p2 + edge$ to $P2[u]$
26 **return** $P1[t]$ and $P2[t]$

3. An example of the algorithm pDAG-I to find the *k* longest paths.

Take the DAG in Figure 2b as an example.

Step 1, vertex 0 is computed, it has only one normal path which contains only itself.

Step 2, excepting vertex 0, the mass of vertex 1 is smallest, so vertex 1 should be computed. It does not have paths.

Step 3, the vertex 2 should be computed, it has only one normal path 0→2, the weight is 24.

Step 4, then vertex 3 should be computed, it has two prefix vertices 0 and 2. So vertex 3 has two normal paths 0→2→3 and 0→3, the weights are 55 and 29, respectively.

Step 5, vertex 4 should be computed. It has no prefix vertices. So it does not have any paths.

Step 6, vertex 5 should be computed, it has one prefix vertex 3. So it has two modified paths 0→2→3→5 and 0→3→5, the weights are 81 and 55, respectively.

Step 7, vertex 6 should be computed, it does not have any paths.

Step 8, vertex 7 should be computed, it has two prefix vertices 5 and 3. So vertex 7 has two modified paths and two normal paths, the two modified paths are 0→2→3→5→7 and 0→3→5→7, the weights are 108 and 82, while the two normal paths are 0→2→3→7 and 0→3→7, the weights are 78 and 52.

Step 9, vertex 8 should be computed, it does not have any paths.

Step 10, vertex 9 should be computed, it has two prefix vertices 7 and 5. So similarly it has four modified paths and two normal paths. The four modified paths are 0→2→3→5→7→9, 0→3→5→7→9, 0→2→3→5→9 and 0→3→5→9, the weights are 108, 82, 81 and 55. The two normal paths are 0→2→3→7→9 and 0→3→7→9, the weights are 78 and 52.

Step 11, at last, because no more vertices should be computed, the algorithm ends.

4. Algorithm pDAG-II is always better than or at least equal to pDAG-I. In other words, a correct path is sure to be found by pDAG-II if it can by found by pDAG-I.

In some conditions, pDAG-II might not recall the correct path at a very low probability. For the example in Figure 2b, assuming that vertices 2 and 7 belong to a cognate vertex pair and vertex 7 is the correct vertex. *k* is set as 1, which means that only the best normal path and the best modified path can be stored in the iterative process. As a result, the normal path 0→2→3 is stored at vertex 3, the modified path 0→2→3→5 is stored at vertex 5 and there are no paths stored at vertex 7 because it is cognate with vertex 2. At last, the best path to vertex 9 is from vertex 5, but this path 0→2→3→5→9 is not correct (the correct path is 0→3→5→7→9). However, we will prove that pDAG-II is always better than or at least equal to pDAG-I. In other words, there is no such a situation that the correct path can be recalled by pDAG-I but not by pDAG-II.

Proof: Assume that there is no such a case that one peak in the spectrum is matched more than one type of ion in the real data. In other word, all the paths containing the cognate vertex are wrong. (Note in real datasets, there are only 7.0% of the spectra containing a peak that matches more than one types of ions)

G(*V*, *E*) is a directed acyclic graph. The set of the vertices is $\{s, x_1, x_2, \dots, x_k, y_k, \dots, y_2, y_1, t\}$, s and t are the source vertex and the destination vertex respectively. All vertices is sorted by mass in ascending order. Because the cognate vertex pair is converted by the same peak, one for b ion and another for y ion, then the mass m_1 and m_2 of these two vertices follow a formula: $m_1 + m_2 = M + 1$ (M is the precusor ion mass). Then we can suppose x_i and y_i for every i are the cognate vertex pair. In computing the paths from s to t, we should consider the antisymmetry restriction. In pDAG-II, when computing the paths from s to y_i, it will consider all paths of the prefix vertices. If there is one path containing x_i, then we shouldn't consider this path. While in pDAG-I, all paths of the prefix vertices are considered. We should prove that pDAG-II is at least not worse than pDAG-I. That is to say there is no such a

case that pDAG-I can recall the right path while pDAG-II can not.

In order to simplify the analysis, we assume that the right path is $s \rightarrow y_k \rightarrow \ldots \rightarrow y_1 \rightarrow t$. The right path doesn't contain any x_i. Then all paths from s to x_k computed by pDAG-II are same with pDAG-I because all vertices before x_k have no the cognate vertex. We give a proof by contradiction. The right path can be found by pDAG-I but not be found by pDAG-II. That means there is at least one path from s to y_k found by pDAG-I which doesn't contain any x_i in the *k* longest paths. But all paths before computing y_k are the same between these two algorithms. Therefore, there must be at least one path from s to y_k found by pDAG-II which doesn't contain any x_i in the *k* longest paths.

In the example above, these two algorithm are same. Let us see a more general case, the right path is $s \rightarrow x_j \rightarrow y_k \rightarrow \ldots \rightarrow y_{j+1} \rightarrow y_{j-1} \rightarrow \ldots \rightarrow y_1 \rightarrow t$. In this case, pDAG-II is better than pDAG-I with a very large probability. Similar to the discussion above, we can assume that pDAG-I can find the right path. In other word, there is at least one path from s to y_k found by pDAG-I which contains x_j but doesn't contain any other x_i. Before computing the paths of y_k, these two algorithms are the same. So pDAG-II can also get the sub right path s$\rightarrow x_j \rightarrow y_k$. In the next steps, these two algorithms will get different paths because there are many cognate vertices after y_k. When computing the paths of y_j, to the sub right path s$\rightarrow x_j \rightarrow y_k$, pDAG-II will ignore all paths of y_j because the sub right path contains x_j. While pDAG-I will still consider y_j, and if the weight of y_j is very high, then the paths containing y_j will be easily found in the *k* longest paths of y_p (for every $p > j$). Therefore, all paths containing y_j found by pDAG-I are wrong.

5. Theorem 1. The time complexity of algorithm pDAG-II is $O(k|V| + |E| + k|V|\log\bar{d})$.

Proof: Assuming that G(V, E) is a directed acyclic graph. The in-degree of each vertex v_i is d_i and the average in-degree of all vertices is $\bar{d} = \frac{\sum_i d_i}{|V|}$. In finding the k longest paths of the vertex v_i, every prefix vertex of v_i should be considered. Because the k longest paths of every prefix vertex is already sorted by the weight. We can use the loser tree to record these paths. Firstly, each top path of all prefix vertices need to be recorded by the loser tree (it takes $O(d_i)$). Then, we can get the first path of v_i, push the second path to the loser tree. Inserting each new path into the loser tree, we should take $O(\log d_i)$ to maintain this tree. Lastly, popping up all k longest paths will take $O(k)$. So the whole time of finding the k longest paths of every vertex should be $O(d_i + k\log d_i + k)$. That is to say finding the k longest paths of all vertices should take $O(\sum_i(d_i + k\log d_i + k)) = O(k|V| + |E| + k\sum_i \log d_i)$ $(\sum_i d_i = |E|)$. In the formula, $\sum_i \log d_i = \log(\prod_i d_i) \leq \log \bar{d}^{|V|} = |V|\log\bar{d}$. At last, the time complexity of finding the k longest paths is $O(k|V| + |E| + k|V|\log\bar{d})$.

While the time complexity of the algorithm in pNovo+ is $O(|E|\log d + k|V|\log\bar{d})$ where d is the maximum in-degree of the graph. Normally, k is 150 shown in Table S10, $|V| = 300$, $\bar{d}$ and d are respectively 14.7 and 50 shown in Figure S2, $|E| = |V|\,\bar{d} \approx 15|V|$. So the time complexity of the algorithm in Open-pNovo and pNovo+ are respectively $k|V| + |E| + k|V|\log\bar{d} = 150|V| + 15|V| + 150|V|\log 14.7 \approx 747|V|$ and $|E|\log d + k|V|\log\bar{d} = 15|V|\log 50 + 150|V|\log 14.7 \approx 666|V|$. Noted that the last item in time complexity formula of these two algorithms are the same. So if we want the first algorithm is faster than the second one, we only need to let $k|V| + |E| < |E|\log d$ where $k|V| + |E| = k|V| + \bar{d}|V|$ and $|E|\log d = \bar{d}|V|\log d$. In other word, we need to let $k < \bar{d}(\log d - 1)$ so that the time complexity of Open-pNovo is smaller than that of pNovo+.

Supplemental Figures (9 figures)

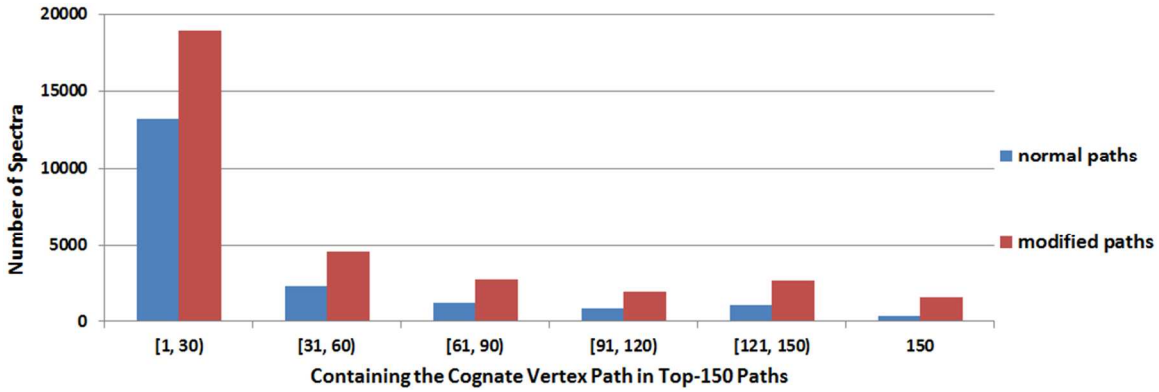
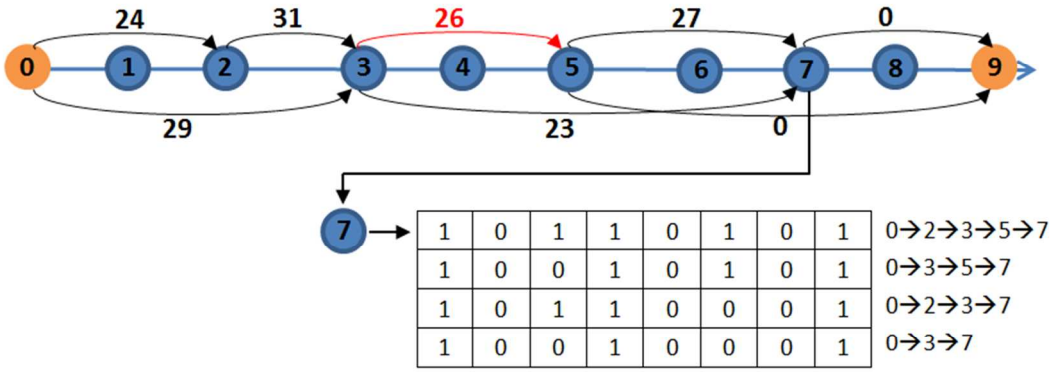


Figure S1. The frequency of the 150 longest normal paths and the 150 longest modified paths containing at least one cognate vertex pair. For example, the leftmost blue column means the number of spectra in which 1~30 normal paths in 150 longest paths containing the cognate vertex is 13,232. The statistics are from the three real datasets (60,777 PSMs).

a)



1	0	1	1	0	1	0	1	0→2→3→5→7
1	0	0	1	0	1	0	1	0→3→5→7
1	0	1	1	0	0	0	1	0→2→3→7
1	0	0	1	0	0	0	1	0→3→7

b)

A V-bit vector of one path from the source to any one vertex:

1	**0**	**1**	**1**	**0**	**1**	**0**	**1**	**0**	**1**	...

	Memory space (bits)
One path to one vertex	V
All paths of a DAG	$k \times V^2$

V denotes the number of vertices in one DAG and k denotes the number of paths to each vertex.

Figure S2. a) Vertex 7 has four paths (two normal paths and two modified paths) and we store its bit vector in memory. In a bit vector, each row denotes one path and each value denotes whether the corresponding vertex is visited, 0 denotes it is not visited while 1 denotes it is visited. b) The memory space of a bit vector. Generally, the number of vertices and paths are 300 and 150, so the bit vector only needs $150 \times 300^2 = 1.35 \times 10^7$ bits ≈ 13 MB. The time complexity of judging whether a cognate vertex has been visited by the bit vector approach is O (1).

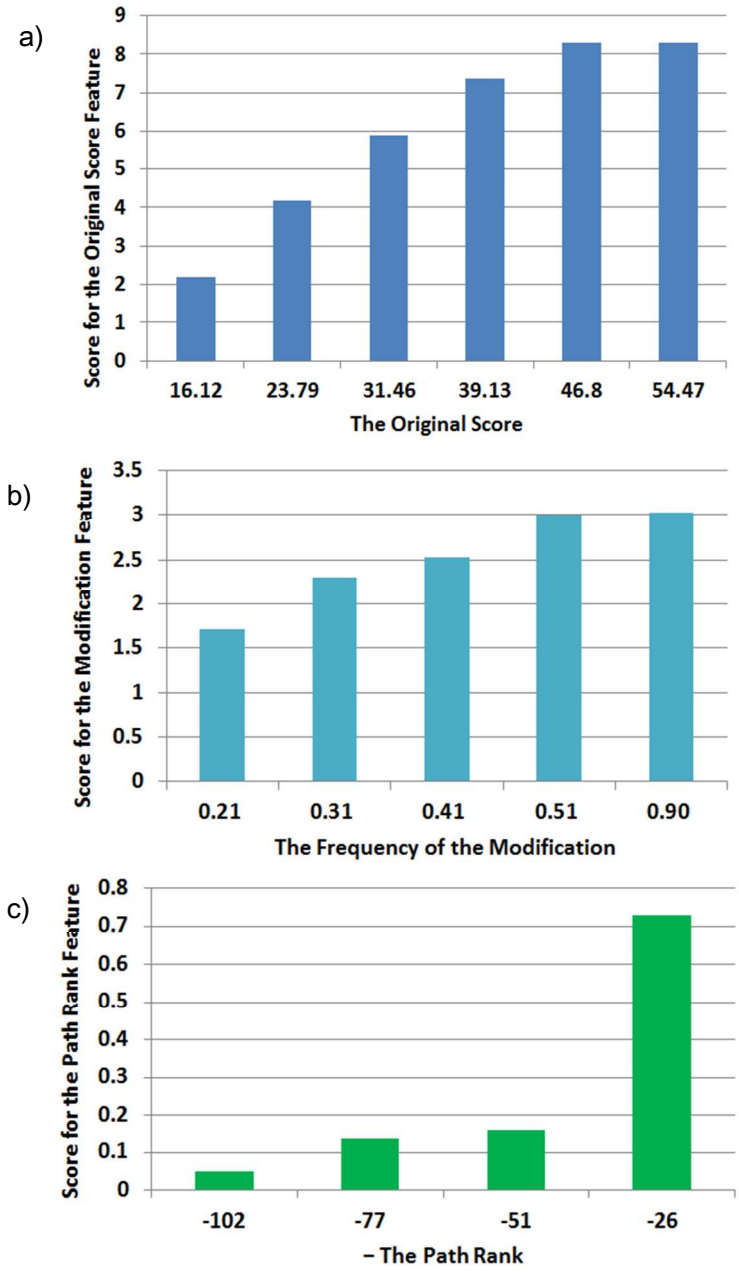


Figure S3. The score function of three most important features trained by RankBoost model. a) The score function of the original score. b) The score function of the frequency of the modification. The frequency of the modification is the second most important feature after the original score. For example, when the frequency of one modification is bigger than or equal to 0.9, the score is 3.0; when the frequency of another modification is smaller than 0.21, then the score is 0.0. c) The score function of the path rank. When the path rank is less than or equal to 26, the score is 0.7. At last, the final score for RankBoost model is summed over all scores determined by each feature.

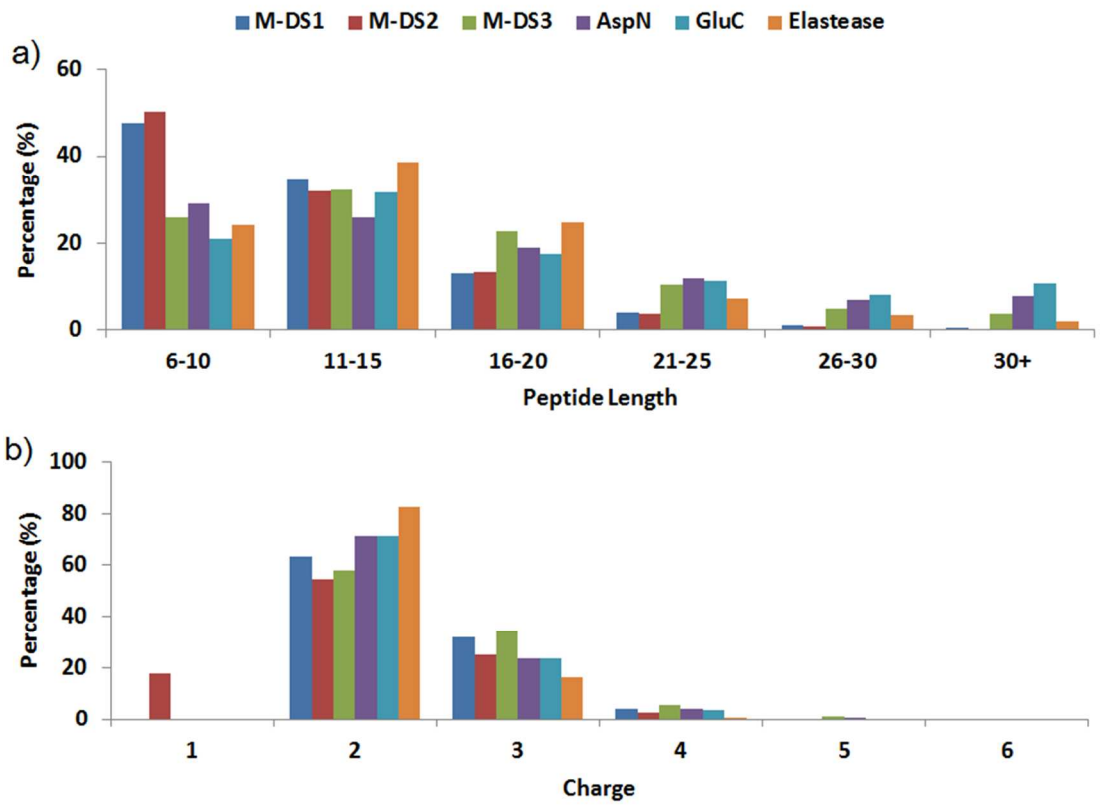


Figure S4. The distributions of a) peptide lengths and b) charge states on the whole data sets of "M-DS1", "M-DS2", "M-DS3", AspN, GluC and Elastease. The other three alternative enzymes (AspN, GluC, Elastease) published in the pNovo+ paper in 2013 are analyzed. The number of spectra on these six data sets are 64,112, 136,560, 452,182, 24,372, 24,006 and 19,846, respectively. The peptide length is approximatly computed by the precursor mass divided by the average mass (110 Da) of 20 amino acids. The percentages of peptides whose lengths are between 6-20 on these six data sets are 94.8%, 95.3%, 81.0%, 73.8%, 70.2% and 87.4%, respectively. Although the peptides on AspN and GluC data sets are slightly longer than other data sets, these peptides (6-20 aa) are still on the vast majority of the whole data sets. The charge states are also analyzed. The percentages of charge states (+2 and +3) on all six data sets are 95.1%, 79.4%, 92.7%, 95.0%, 95.5% and 99.2%. Furthermore, on the "M-DS2" data set, there are 17.7% of precursors whose charge states are +1.

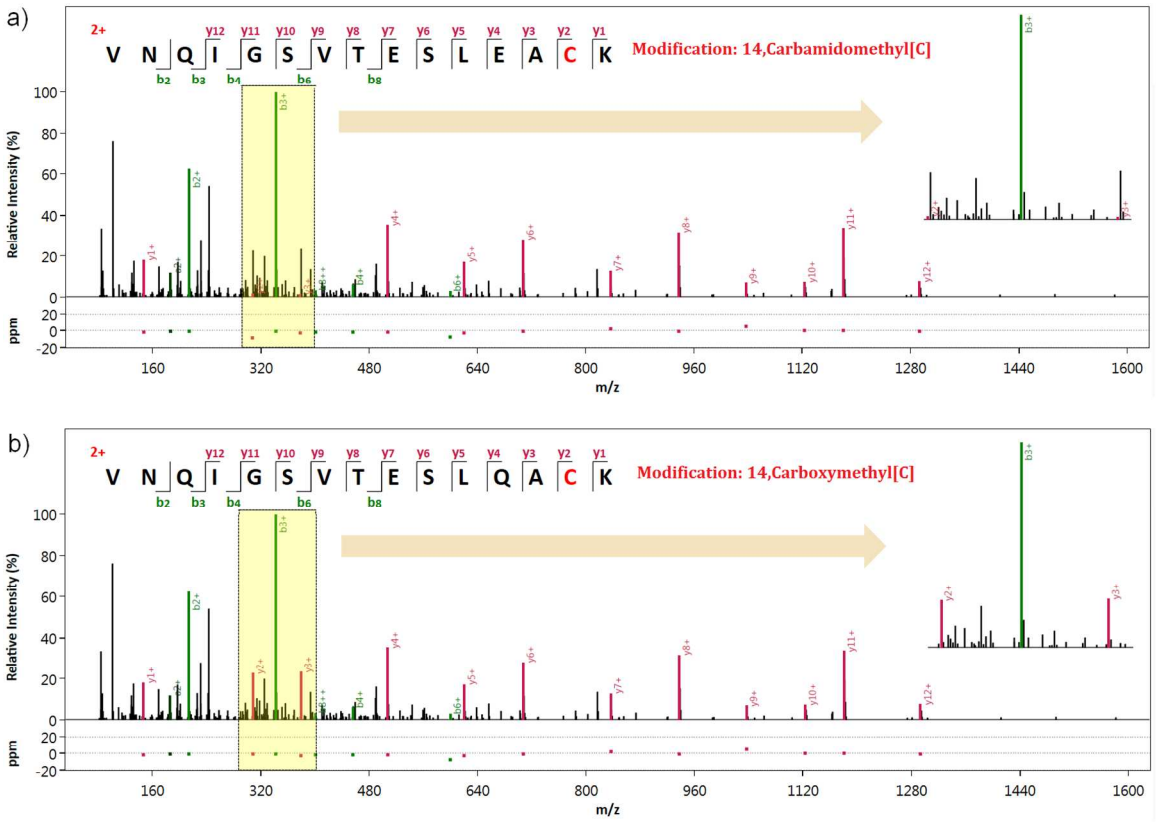


Figure S5. An example of two identifications of one spectrum with very similar peptide sequences but different modifications. a) The wrong result with carbamidomethylation, which is identified by both pNovo+ and PEAKS in no-modification mode. b) The right result with carboxymethylation, which is identified by both pNovo+ and PEAKS in modification mode and by Open-pNovo. (Note: the MS/MS scan of this spectrum is 11,394 from 20100825_Velos2_AnMi_QC_wt_HCD_iso4_swG_2.raw.)

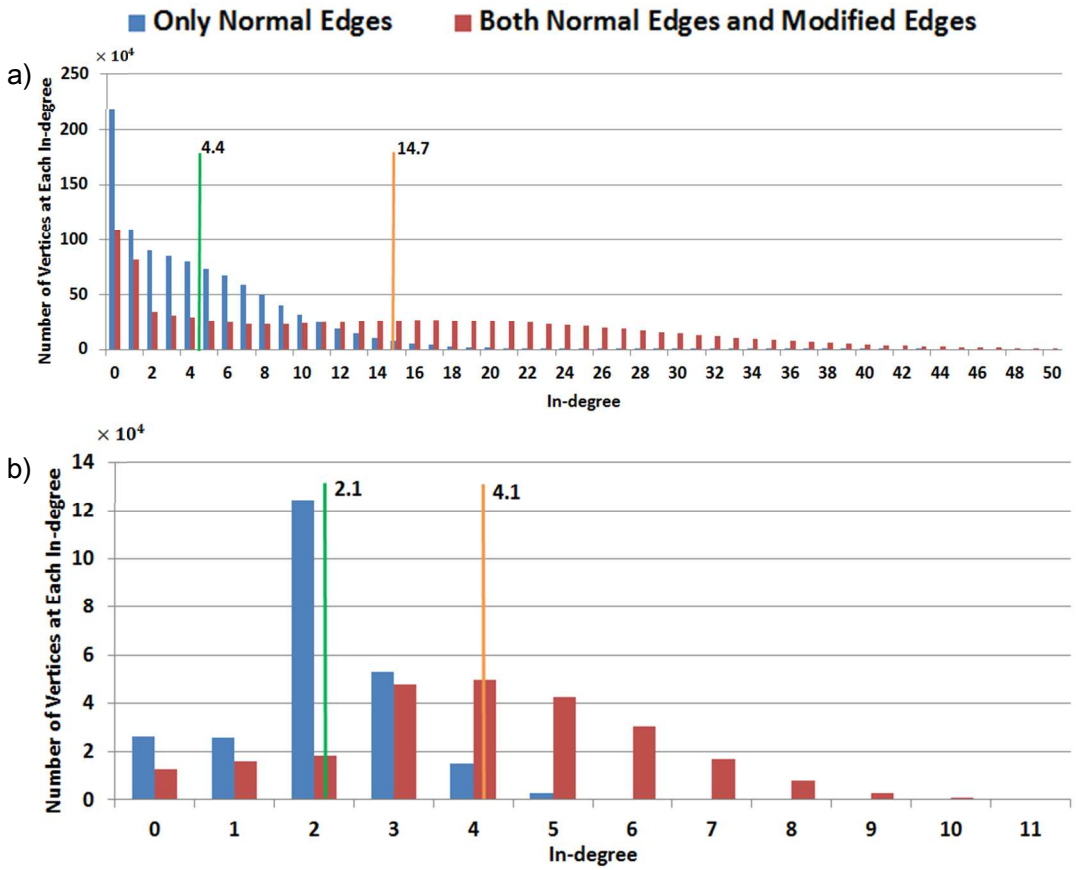


Figure S6. Number of vertices distributed at each in-degree. The green line and the orange line respectively mean the average in-degree of only normal edges and both two types of edges. a) The average in-degree of normal edges and both two types of edges are 4.4 and 14.7 on the three real dataset. b) The average in-degree of normal edges and both two types of edges are 2.1 and 4.1 on the three simulated dataset. The spectrum graph is much more complex on the real dataset than the simulated one, and when considering the modified edges, the spectrum graph became more complex on both the two types of datasets.

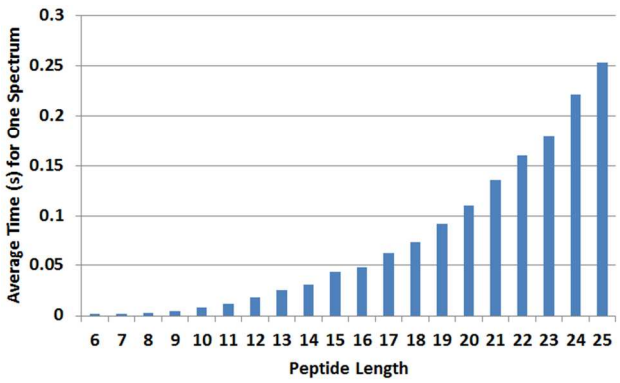


Figure S7. The relationship between peptide length and average running time (s) of Open-pNovo for one spectrum. The average running time grows exponentially with the peptide length (especially for the peptides with lengths greater than 20).

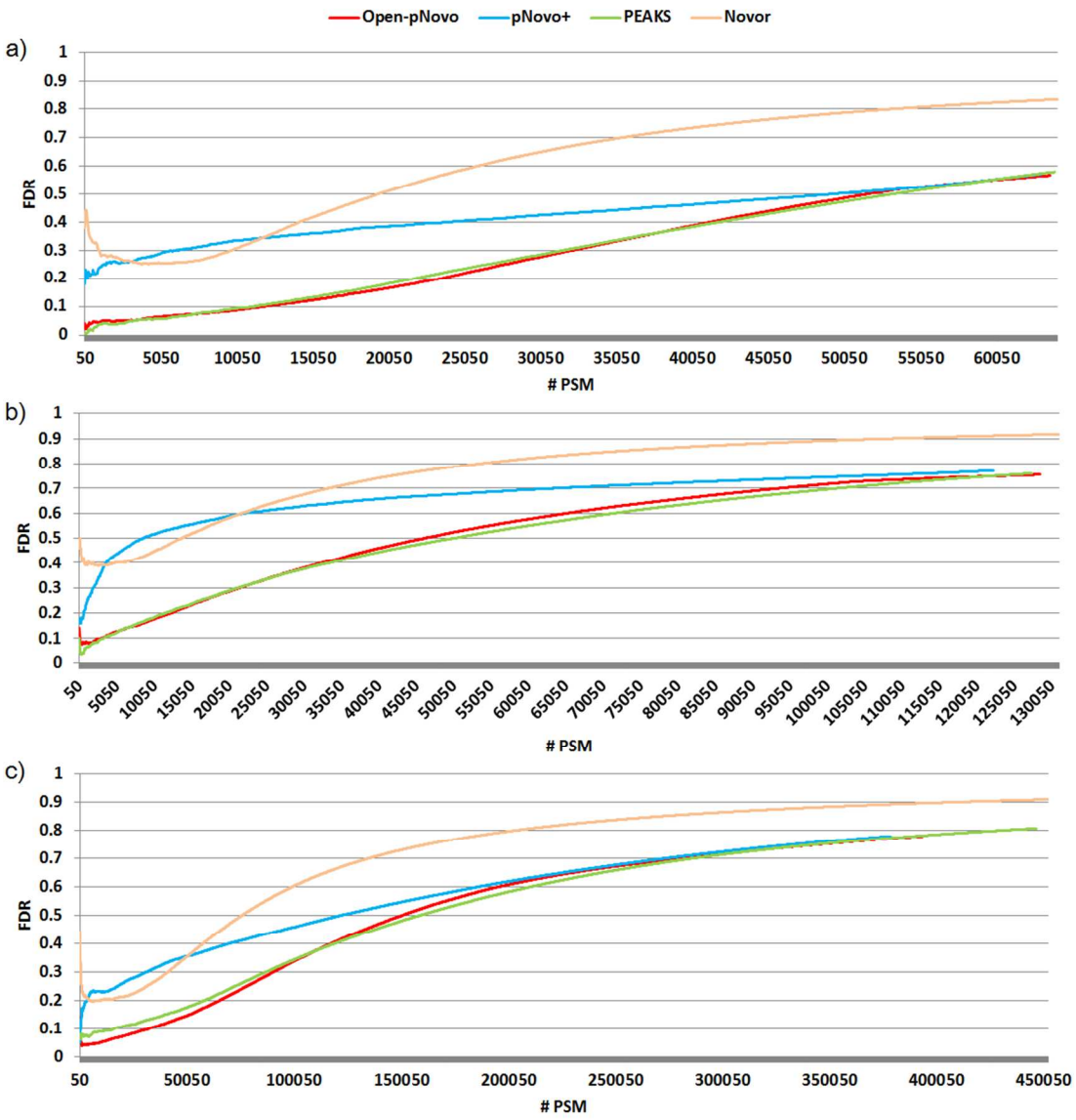


Figure S8. The FDR curves of Open-pNovo, pNovo+, PEAKS and Novor on the complete a) M-DS1, b) M-DS2 and c) M-DS3 data sets. Open-pNovo and PEAKS perform the best.

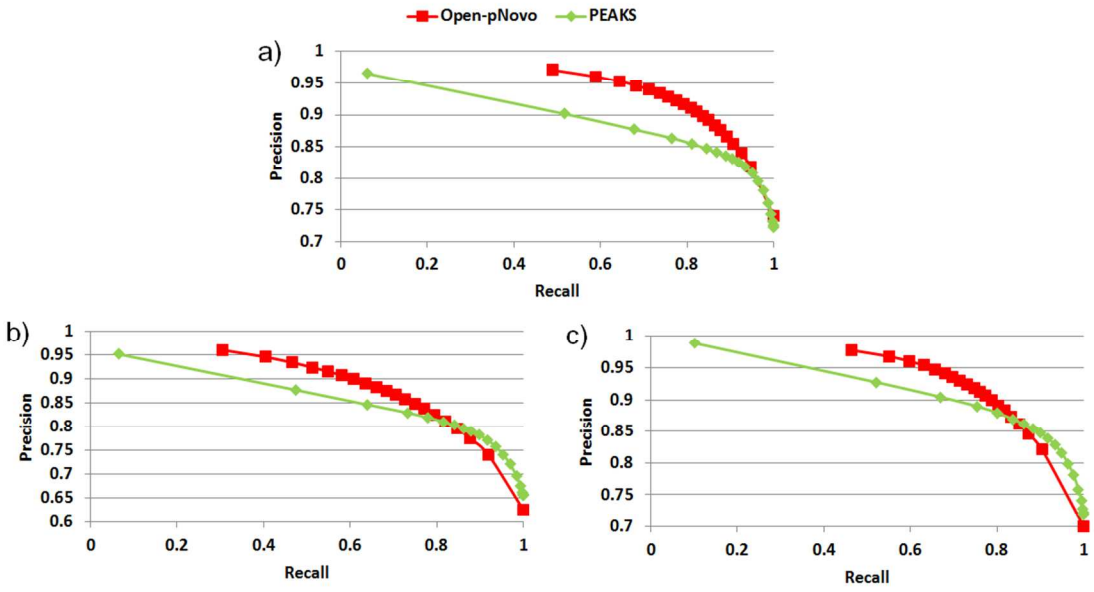


Figure S9. The precision and recall rates of amino acids identified by Open-pNovo and PEAKS on the complete a) M-DS1, b) M-DS2 and c) M-DS3 data sets. When the recall rate is ~50%, the precision rates of Open-pNovo and PEAKS are ~95% and ~90%, respectively.

Supplemental Tables (10 tables)

Table S1. The database search parameters of pFind and PEAKS DB.

Item	pFind	PEAKS DB
Enzyme	Trypsin	Trypsin
Maximum missed cleavage sites	3	3
Precursor tolerance	±20 ppm	±20 ppm
Fragment tolerance	±20 ppm	±0.02Da

Table S2. The search parameters of pNovo+ in the second searching mode on the three simulated datasets.

S-DS1	S-DS2	S-DS3
Label_13C(5)15N(1)[V]	PyruvicAcidIminyl[ProteinN-termV]	Label_15N(1)[V]
PyruvicAcidIminyl[ProteinN-termV]	Label_15N(1)[V]	Label_13C(5)15N(1)[V]
Label_15N(1)[I]	Label_13C(5)15N(1)[V]	Label_15N(1)[I]
Label_15N(1)[V]	Methyl[I]	PyruvicAcidIminyl[ProteinN-termV]
Label_13C(6)15N(1)[I]	Label_13C(6)[I]	Label_13C(6)[I]
Methyl[I]	Label_13C(6)15N(1)[I]	Label_13C(6)15N(1)[I]
Label_13C(6)[I]	Label_15N(1)[I]	Methyl[I]
Myristoleyl[ProteinN-termG]	Myristoleyl[ProteinN-termG]	Label_15N(1)[G]
Label_15N(1)[G]	Label_15N(1)[A]	Myristoyl[AnyN-termG]
Dap-DSP[A]	Label_13C(3)15N(1)[A]	Label_13C(3)15N(1)[A]

Table S3. The effect of the RankBoost algorithm on three real MS/MS datasets.

	No RankBoost		RankBoost	
	Top-1	Top-10	Top-1	Top-10
All spectra (60,777)	55.5% (33,702)	90.7% (55,150)	76.3% (46,382)	93.7% (56,972)
Spectra with modified peptides (15,103)	49.8% (7,517)	82.2% (12,415)	57.2% (8,642)	85.0% (12,842)

Table S4. The top ten modifications recalled from Open-pNovo and two modes of other algorithms on M-DS1 when considering top-10 results.

Modification name	Open-pNovo	pNovo+	pNovo+ (Mods)	PEAKS	PEAKS (Mods)	Novor[b]	Novor (Mods)
Deamidated[Q]	92%	92%	91%	87%	85%	26%	20%
(947)	(870)	(871)	(861)	(826)	(804)	(243)	(190)
Gln->pyro-Glu[AnyN-termQ]	92%	0%	93%	0%	89%	0%	15%
(580)	(532)	(0)	(538)	(0)	(518)	(0)	(85)
Deamidated[N]	84%	85%	84%	80%	77%	20%	15%
(288)	(243)	(246)	(243)	(229)	(223)	(57)	(42)
Carboxymethyl[C]	91%	0%	91%	0%	40%	0%	0%
(183)	(166)	(0)	(167)	(0)	(74)	(0)	(0)
Glu->pyro-Glu[AnyN-termE]	94%	0%	96%	0%	90%	0%	8%
(157)	(147)	(0)	(150)	(0)	(142)	(0)	(12)
Oxidation[M]	77%	85%	84%	84%	84%	20%	17%
(122)	(94)	(104)	(103)	(103)	(102)	(25)	(21)
Acetyl[ProteinN-term]	90%	0%	89%	0%	95%	0%	0%
(87)	(78)	(0)	(77)	(0)	(83)	(0)	(0)
Carbamyl[AnyN-term]	100%	0%	95%	0%	100%	0%	0%
(41)	(41)	(0)	(39)	(0)	(41)	(0)	(0)
Dioxidation[W]	0%[a]	0%	0%	0%	0%	0%	0%
(21)	(0)	(0)	(0)	(0)	(0)	(0)	(0)
Acetyl[S]	100%	0%	0%	0%	0%	0%	0%
(14)	(14)	(0)	(0)	(0)	(0)	(0)	(0)

[a]The mass of the dihydroxy of Tryptophan residue is about equal to the mass of Methionine and Serine.

[b]Novor and Novor (mods) only report the top-1 result, so these two columns only consider the top-1 recall of each modification. In Table S4-S9, the recall of Novor and Novor (mods) are all the top-1 recall.

Table S5. The top ten modifications recalled from Open-pNovo and two modes of other algorithms on M-DS2 when considering top-10 results.

Modification name	Open-pNovo	pNovo+	pNovo+ (Mods)	PEAKS	PEAKS (Mods)	Novor	Novor (Mods)
Gln->pyro-Glu[AnyN-termQ]	91%	0%	92%	0%	87%	0%	13%
(1,083)	(988)	(0)	(1,000)	(0)	(945)	(0)	(141)
Deamidated[Q]	88%	86%	85%	81%	78%	27%	21%
(648)	(567)	(558)	(552)	(527)	(504)	(177)	(135)
Deamidated[N]	79%	78%	77%	81%	80%	27%	20%
(226)	(179)	(177)	(175)	(183)	(180)	(62)	(46)
Oxidation[M]	67%	80%	75%	76%	79%	8%	7%
(156)	(104)	(125)	(117)	(119)	(123)	(12)	(11)
Glu->pyro-Glu[AnyN-termE]	89%	0%	92%	0%	92%	0%	8%
(145)	(129)	(0)	(133)	(0)	(133)	(0)	(12)
Acetyl[ProteinN-term]	89%	0%	89%	0%	89%	0%	0%
(140)	(124)	(0)	(125)	(0)	(124)	(0)	(0)
Carboxymethyl[C]	80%	0%	84%	0%	46%	0%	0%
(99)	(79)	(0)	(83)	(0)	(46)	(0)	(0)
Carbamyl[AnyN-term]	82%	0%	94%	0%	96%	0%	0%
(96)	(79)	(0)	(90)	(0)	(92)	(0)	(0)
Acetyl[S]	85%	0%	0%	0%	0%	0%	0%
(13)	(11)	(0)	(0)	(0)	(0)	(0)	(0)
Acetyl[T]	90%	0%	0%	0%	0%	0%	0%
(10)	(9)	(0)	(0)	(0)	(0)	(0)	(0)

Table S6. The top ten modifications recalled from Open-pNovo and two modes of other algorithms on M-DS3 when considering top-10 results.

Modification name	Open-pNovo	pNovo+	pNovo+ (Mods)	PEAKS	PEAKS (Mods)	Novor	Novor (Mods)
Oxidation[M]	86%	88%	86%	84%	84%	27%	22%
(4,225)	(3,615)	(3,724)	(3,616)	(3,550)	(3,530)	(1,150)	(938)
Deamidated[N]	86%	86%	84%	82%	79%	30%	24%
(2,332)	(2,003)	(2,013)	(1,968)	(1,903)	(1,833)	(691)	(555)
Gln->pyro-Glu[AnyN-termQ]	90%	0%	95%	0%	91%	0%	30%
(1,014)	(908)	(0)	(962)	(0)	(925)	(0)	(308)
Carbamidomethyl[AnyN-term]	79%	0%	0%	0%	0%	0%	0%
(958)	(760)	(0)	(0)	(0)	(0)	(0)	(0)
Carbamyl[AnyN-term]	70%	0%	88%	0%	87%	0%	0%
(538)	(374)	(0)	(473)	(0)	(467)	(0)	(0)
Cation_Na[D]	74%	0%	0%	0%	80%	0%	0%
(256)	(190)	(0)	(0)	(0)	(205)	(0)	(0)
Acetyl[ProteinN-term]	87%	0%	90%	0%	0%	0%	0%
(228)	(198)	(0)	(205)	(0)	(0)	(0)	(0)
Cation_Na[E]	71%	0%	0%	0%	0%	0%	0%
(216)	(154)	(0)	(0)	(0)	(0)	(0)	(0)
Carbamidomethyl[H]	68%	0%	0%	0%	0%	0%	0%
(157)	(107)	(0)	(0)	(0)	(0)	(0)	(0)
Dehydrated[D]	64%	0%	0%	0%	0%	0%	0%
(123)	(79)	(0)	(0)	(0)	(0)	(0)	(0)

Table S7. The top ten modifications recalled from Open-pNovo and two modes of other algorithms on S-DS1 when considering top-10 results.

Modification name	Open-pNovo	pNovo+	pNovo+ (Mods)	PEAKS	Novor
Label_13C(5)15N(1)[V]	100%	0%	95%	0%	0%
(109)	(109)	(0)	(104)	(0)	(0)
PyruvicAcidIminyl[ProteinN-termV]	100%	0%	92%	0%	0%
(99)	(99)	(0)	(91)	(0)	(0)
Label_15N(1)[I]	95%	0%	100%	0%	0%
(83)	(79)	(0)	(83)	(0)	(0)
Label_15N(1)[V]	99%	0%	98%	0%	0%
(80)	(79)	(0)	(78)	(0)	(0)
Label_13C(6)15N(1)[I]	100%	0%	97%	0%	0%
(78)	(78)	(0)	(76)	(0)	(0)
Methyl[I]	97%	0%	96%	0%	0%
(77)	(75)	(0)	(74)	(0)	(0)
Label_13C(6)[I]	99%	0%	96%	0%	0%
(75)	(74)	(0)	(72)	(0)	(0)
Myristoleyl[ProteinN-termG]	100%	0%	95%	0%	0%
(56)	(56)	(0)	(53)	(0)	(0)
Label_15N(1)[G]	100%	0%	98%	0%	0%
(47)	(47)	(0)	(46)	(0)	(0)
Dap-DSP[A]	93%	0%	98%	0%	0%
(46)	(43)	(0)	(45)	(0)	(0)

Table S8. The top ten modifications recalled from Open-pNovo and two modes of other algorithms on S-DS2 when considering top-10 results.

Modification name	Open-pNovo	pNovo+	pNovo+ (Mods)	PEAKS	Novor
PyruvicAcidIminyl[ProteinN-termV] (114)	100% (114)	0% (0)	82% (94)	0% (0)	0% (0)
Label_15N(1)[V] (104)	97% (101)	0% (0)	93% (97)	0% (0)	0% (0)
Label_13C(5)15N(1)[V] (103)	97% (100)	0% (0)	95% (98)	0% (0)	0% (0)
Methyl[I] (77)	91% (70)	0% (0)	92% (71)	0% (0)	0% (0)
Label_13C(6)[I] (75)	99% (74)	0% (0)	97% (73)	0% (0)	0% (0)
Label_13C(6)15N(1)[I] (71)	100% (71)	0% (0)	97% (69)	0% (0)	0% (0)
Label_15N(1)[I] (66)	100% (66)	0% (0)	92% (61)	0% (0)	0% (0)
Myristoleyl[ProteinN-termG] (53)	98% (52)	0% (0)	74% (39)	0% (0)	0% (0)
Label_15N(1)[A] (51)	96% (49)	0% (0)	94% (48)	0% (0)	0% (0)
Label_13C(3)15N(1)[A] (43)	95% (41)	0% (0)	93% (40)	0% (0)	0% (0)

Table S9. The top ten modifications recalled from Open-pNovo and two modes of other algorithms on S-DS3 when considering top-10 results.

Modification name	Open-pNovo	pNovo+	pNovo+ (Mods)	PEAKS	Novor
PyruvicAcidIminyl[ProteinN-termV] (125)	94% (118)	0% (0)	73% (91)	0% (0)	0% (0)
Label_15N(1)[V] (111)	95% (105)	0% (0)	96% (107)	0% (0)	0% (0)
Label_13C(5)15N(1)[V] (107)	91% (97)	0% (0)	87% (93)	0% (0)	0% (0)
Label_15N(1)[I] (93)	92% (86)	0% (0)	92% (86)	0% (0)	0% (0)
Label_13C(6)[I] (78)	94% (73)	0% (0)	95% (74)	0% (0)	0% (0)
Label_13C(6)15N(1)[I] (76)	100% (76)	0% (0)	96% (73)	0% (0)	0% (0)
Methyl[I] (75)	87% (65)	0% (0)	87% (65)	0% (0)	0% (0)
Label_15N(1)[G] (58)	93% (54)	0% (0)	93% (54)	0% (0)	0% (0)
Myristoyl[AnyN-termG] (54)	96% (52)	0% (0)	76% (41)	0% (0)	0% (0)
Label_13C(3)15N(1)[A] (48)	92% (44)	0% (0)	88% (42)	0% (0)	0% (0)

Table S10. The influence of the number of temporary paths *k* in Open-pNovo.

k	M-DS1 (8,600)		M-DS2 (6,727)		M-DS3 (45,450)	
	Time (s)	Recall	Time (s)	Recall	Time (s)	Recall
50	47.0	8,098	36.7	6,041	188.7	41,998
100	56.1	8,172	45.4	6,161	238.9	42,391
150	62.9	8,214	52.4	6,211	260.8	42,547
200	73.8	8,219	60.5	6,225	300.3	42,639
250	80.9	8,223	66.3	6,245	325.7	42,700
300	89.6	8,219	74.3	6,265	371.1	42,730
350	98.3	8,214	81.3	6,268	409.5	42,753
400	111.9	8,210	92.9	6,265	451.3	42,765