Algorithm 1: Edge weight calculation.

Data: KB (knowledge base); property weights; f_l; f_g; c_{max}

Result: Weighted edges for graph traversal

foreach *node in KB* **do**

`/* Exclude triples with literals, only count traversable connections */`

$l_{in} \leftarrow$ **count** (s, p, *node*) **where** s is not a literal;

$l_{out} \leftarrow$ **count** (*node*, p, o) **where** o is not a literal;

$l_{total} \leftarrow l_{in} + l_{out}$;

insert (*node*, *links*, l_{total}) **into** KB;

end

$E \leftarrow$ **select** (s, p, o, l_s, l_o) **from** KB **where** (s, p, o) and (s, *links*, l_s) and (o, *links*, l_o);

$E \leftarrow$ **join** E **with** property weights **on** p;

`/* Enable traversing edges in both directions */`

$E_{spo} \leftarrow$ **select** ($s : s$, $p : p$, $o : o$, $w_p : w_p$, $l : l_o$, dir : 'spo') **from** E;

$E_{ops} \leftarrow$ **select** ($s : o$, $p : p$, $o : s$, $w_p : w_p$, $l : l_s$, dir : 'ops') **from** E;

$E \leftarrow$ concatenate E_{spo}, E_{ops};

`/* Calculate the weights */`

$E \leftarrow$ sort and repartition E by s;

$G \leftarrow$ **group** E **by** s, p, dir;

$result \leftarrow \emptyset$;

foreach *group in* G **do**

$g_{size} \leftarrow$ count items in *group*;

foreach *item in group* **do**

$item.w_{rel} \leftarrow item.w_p \cdot (item.l)^{f_l} \cdot (g_{size})^{f_g}$;

end

$group \leftarrow$ **order** *group* **by** w_{rel} ascending **limit** c_{max};

foreach *item in group* **do**

add (($item.s$, genericRelation, $item.o$), weight, $item.w_{rel}$) to *result*;

end

end

return *result*;

Algorithm 2: Graph neighborhood expansion.

Data: KB (knowledge base); *candidate*; $depth_{max}$; $dist_{max}$

Result: List of entities in the neighborhood of *candidate*

```
1  candidate.dist ← 0;
2  active ← {candidate};
3  result ← ∅;
4  for depth ← 1 to depth_max do
5      active_new ← ∅;
6      foreach v in active do
7          if v.dist > dist_max then
8              continue;
9          end
10         foreach (u, weight) in neighbors(v) do
11             dist ← v.dist + weight;
12             mutex lock u;
13             if u not visited or u.dist > dist then
14                 u.dist ← dist;
15                 add u to active_new;
16                 add u to result;
17             end
18             mutex release u;
19         end
20     end
21     active ← active_new;
22 end
23 result ← select u from result where (u, isEntityLinkingTarget, true);
24 return result;
```

Algorithm 3: Entity-entity coherence algorithm.

Data: $max\_eliminate\_factor$; $data$ – array of ($mention$, $candidates$); where $candidates$ in an array of ($candidate$, $score$, $neighbors$); where $neighbors$ is an array of ($neighbor$, $distance$)

Result: Mapping from $candidate$ to $boost$

$all\_c \leftarrow$ flatten $data.candidates$;
$max\_candidates \leftarrow \min(0.75 \cdot |all\_c|, 5 \cdot |data|)$;
$all\_c \leftarrow$ **select** distinct ($candidate$, $neighbors$) **from** all_c **order by** $score$ descending **limit** $max\_candidates$;
$related\_ents \leftarrow$ flatten all_c to ($neighbor$, $distance$, $candidate$);
$S \leftarrow 0_{|all\_c| \times |all\_c|}$;
foreach *c1 in related_ents* **do**
    foreach *c2 in related_ents* **do**
        if *c1 and c2 are linked to the same mention* **then continue end**;
        $similarity \leftarrow 1 \ / \ (1 + c1.distance + c2.distance)$;
        if $similarity > S[c1.candidate, c2.candidate]$ **then**
            $S[c1.candidate, c2.candidate] \leftarrow similarity$;
        end
    end
end
$result \leftarrow \emptyset$;
for $i \leftarrow 0$ **to** $max\_eliminate\_factor \cdot |all\_c|$ **do**
    $c_r \leftarrow$ find a non-taboo row in S with smallest sum;
    if c_r *corresponds to the last candidate of a mention* **then**
        add (candidate of c_r, sum of c_r row in S) to $result$;
        mark row c_r as taboo;
    else
        set column c_r in S to zeros;
        mark row c_r as taboo;
    end
end
add remaining non-taboo rows to $result$;
return $result$;

Algorithm 4: Candidate selection and enhancement.

Data: *mentions* – an array of (*mention*, *candidates*); β – 0.3 by default

Result: final set of relevant entities

`/* Collect all identified mentions and their parents */`

entities ← ∅;
foreach *(mention, candidates) in mentions* **do**
    (*candidate*, *score*) ← find top-scoring candidate in *candidates*;
    add (*mention.text*, *mention.lemma*, *candidate*, *score*) to *entities*;
    foreach *parent in parent entities of candidate* **do**
        add (*mention.text*, *mention.lemma*, *parent*, $parent.weight \cdot score$) to *entities*;
    end
end

`/* Group by candidate, aggregate scores */`

grouped ← **group** *entities* **by** *candidate*;
results ← ∅;
foreach *g in grouped* **do**
    $score \leftarrow |\text{unique}(g.lemma)| \cdot |g.lemma|^{\beta} \cdot \text{mean}(g.score)$;
    add (*g.candidate*, *score*) to *results*;
end

`/* Narrow down the result set to n most relevant entities */`

results ← **order** *results* **by** *score* descending;
knee ← find knee in *results.score*;
results ← take first *knee* rows in *results*;
return *results*;