Pseudocode for AutoIncrementalClustering

Pseudocode

Algoritmo 1 AgrupamientoAutoIncremental

Require: *vocabulario*: Vocabulario del corpus, *matriz-coocurrencia*: Matriz de coocurrencia del corpus, *word_embeddings*: *Word embeddings* de las palabras del vocabulario, *min_sim*: Umbral de similitud mínima intra-clúster, *min_coh*: Umbral de coherencia mínima intra-clúster.
Ensure: Grupos de palabras semánticamente similares del vocabulario.

```
function AUTO_INCREMENTAL CLUSTERING(vocabulario, matriz_coocurrencia, word_embeddings, min_sim, min_coh)
    grupos ← {}        ▷ Lista de grupos de la forma (centroide, word_embeddings, palabras).
    for cada word embedding w_e de las palabras del vocabulario do
        if no existen grupos then
            crear primer grupo con w_e
        else
            grupos_similares ← {C | C en grupos and sim(w_e, C) ≥ min_sim}
            grupos_coherentes ← {C | C en grupos_similares and coh(w, {w_C}) ≥ min_coh}
            if no existen grupos coherentes para w then
                crear un nuevo grupo para w
            else
                añadir w_e a los grupos coherentes
                actualizar centroides de los grupos modificados
            end if
        end if
    end for
      return C
end function
```

Algoritmo 2 PalabrasRepresentativasTópico

Require: *tópico*: Número del tópico, k: Número inicial de palabras principales a recuperar
Ensure: Lista de las palabras más relevantes para el tópico especificado

```
function OBTENER_PALABRAS_PRINCIPALES(topico, k)
    palabras_del_topico ← Obtener las k palabras más probables para el tópico, que sean sustantivos, y sus probabilidades
    Ordenar palabras_del_topico en orden descendente por probabilidad
    cambios_de_probabilidad ← Calcular cambios en la probabilidad entre palabras consecutivas
    cambio_promedio ← Calcular el promedio de cambios_de_probabilidad
    desviacion_estandar ← Calcular la desviación estándar de cambios_de_probabilidad
    indice_caida_significativa ← Encontrar el primer índice donde el cambio es mayor que cambio_promedio + desviacion_estandar
    palabras_filtradas ← Filtrar las palabras hasta el punto de caída significativa
    return palabras_filtradas
end function
```

Algoritmo 3 LeskExtendido

Require: *palabra*: La palabra a desambiguar, *contexto*: Contexto de la palabra, *synsets*: Lista de *synsets* de la palabra objetivo. *modelo*: Modelo preentrenado de *word embeddings*.
Ensure: *Synset* que mejor se ajusta al contexto.

```
1:  function EXTENDED_LESK(palabra, contexto, synsets, modelo)
2:      context_embedding ← media aritmética de los word embeddings de las palabras del contexto.
3:      for cada synset de la palabra objetivo do
4:          if el synset no es un verbo then
5:              definition_embedding ← media aritmética de las palabras que conforman la definición del synset.
6:              similitud ← sim(context_embedding, definition_embedding)
7:              if similitud > maxima_similitud then
8:                  maxima_similitud ← similitud
9:                  mejor_synset ← synset
10:             end if
11:         end if
12:     end for
          return mejor_synset
13: end function
```

Algoritmo 4 AlgoritmoGenéticoWSD

Require: *synsets*: Conjunto de synsets, *generaciones*: Número de generaciones, *tamaño_pob*: Tamaño de la población.
Ensure: Conjunto de *synsets* elegidos y su valor de fitness.

function GENETIC_ALGORITHM(*synsets*, *generaciones*, *tamaño_pob*)
    población ← INIT_POPULATION(*synsets*, tamaño_pob)
    mejor_solución ← ([], $-\infty$) ▷ Inicializar la mejor solución encontrada
    for cada generación **do**
        for cada individuo de la población **do**
            fitness ← EVALUATE(individuo)
            if fitness > *mejor_solución*[*fitness*] **then**
                mejor_solución ← (individuo, fitness)
            end if
        end for
        mejores_individuos ← SELECT_PARENTS(*población*, *valores_fitness*)
        población ← NEW_POPULATION(*mejores_individuos*)
    end for
    return *mejor_solucion*
end function
function EVALUATE(*synsets*)
    Calcular la similitud media entre los *synsets* seleccionados utilizando la medida de similitud según el camino (path) que ofrece wordnet
    Las combinaciones que contengan algún synset que actúe como verbo serán penalizadas.
    return Valor de similitud media
end function
function INIT_POPULATION(*context_synsets*, tamaño_pob)
    Inicializar *población* como lista vacía
    for cada iteración hasta *tamaño_pob* **do**
        Crear *individuo* como una lista de listas binarias, donde cada lista representa los synsets de una palabra. Asigna un '1' en una posición aleatoria de cada lista, correspondiendo al synset activo de la palabra que representa.
        Añadir *individuo* a *población*
    end for
    return *población*
end function
function SELECT_PARENTS(*población*, *valores_fitness*, *selection_proportion*)
    Seleccionar los mejores individuos como padres usando selección por ruleta
    return Lista de los mejores padres
end function
function NEW_POPULATION(*mejores_individuos*, *razón_cruce, razón_mutación*)
    Generar una nueva población usando operaciones de cruce y mutación
    return Nueva población
end function
function CROSSOVER(*padre_1*, *padre_2*)
    Seleccionar un punto de cruce aleatorio
    Crear dos hijos intercambiando los conjuntos de *synsets* en el punto de cruce
    return Lista con ambos hijos
end function
function MUTATE(*individuo*)
    Seleccionar un número aleatorio de conjuntos de *synsets* para mutar
    for cada conjunto seleccionado **do**
        Cambiar el *synset* activo por otro aleatorio
    end for
    return *individuo* mutado
end function