ESTUDIO DE HEURÍSTICAS PARA LA IMPLEMENTACIÓN DE A* EN CTH BASADOS EN SELECCIÓN DE UNIDADES

Lluís Formiga y Francesc Alías

Departamento de Comunicaciones y Teoría de la Señal
Enginyeria i Arquitectura La Salle. Universitat Ramon Llull
Pg. Bonanova 8. 08022 Barcelona
$\{llformiga, falias\}$ @$salle.URL.edu$

RESUMEN

Los conversores texto-habla basados en selección de unidades (CTH-SU) tienen que realizar una búsqueda óptima de las unidades en un corpus de voz, para así obtener síntesis de alta calidad. Esta búsqueda hasta el momento se ha llevado a cabo mediante el algoritmo de Viterbi. Nuestro trabajo aporta un nuevo algoritmo (A*) al anteriormente utilizado para mejorar su complejidad computacional. Con este objetivo, se detalla una retrospectiva de los trabajos anteriores que han realizado dicha sustitución. Posteriormente, se define un banco de pruebas para valorar su eficacia y se analizan los resultados para validar, en un último paso, su justificación teórica.

1. INTRODUCCIÓN

Este trabajo se emmarca dentro del ámbito de los conversores de texto en habla basados en selección de unidades (CTH-SU) a partir de un corpus de voz con más de una realización por unidad fonética. La selección de unidades es un proceso que ejerce de cuello de botella en el contexto de la CTH-SU [1, 2, 3]. El proceso de la selección de unidades parte de una especificación fonética y prosódica de la unidad deseada indicada por el módulo de procesamiento del lenguaje natural del CTH-SU (PLN). Típicamente, a cada fonema se le asocia una prosodia, caracterizada por su frecuencia fundamental, duración entre otros. La selección se efectúa en una gran base de datos de unidades candidatas mediante unas funciones de coste [1] ponderadas adequadamente [4, 5, 6]. El proceso más utilizado para encontrar la mejor secuencia de unidades del corpus se realiza mediante una búsqueda *naive* de Viterbi que evalua los subcostes de *Target* y *Concatenación*. La señal de voz que conforman dichas unidades es concatenada y ajustada para satisfacer las especificaciones originales.

En [3] y [7] se menciona y argumenta el incanzable aumento complejidad computacional para la minimización de la función de coste. Se detallan heurísticas (conocimiento experto) en varios puntos para mejorar la eficacia: proporcionar un tratamiento especial para las pausas, o predisponer la concatenación de semifonemas específicos son algunos ejemplos.

Durante los últimos años se han dedicado esfuerzos importantes a investigar cómo reducir la complejidad computacional y la obtención de una aproximación de síntesis en tiempo real [3, 8]. Parte de la investigación [9] se ha concentrado en disminuir el espacio de búsqueda, clusterizandolo mediante árboles de decisión. Otra línea de investigación [10] se basa en optimizar el método de busqueda mediante técnicas basadas en poda o con *caché*, que se basan en evitar el cálculo de caminos que producen un coste no aceptable (vía estimación o comparativa de cálculo hecho).

En este trabajo se pretende dar una retrospectiva del estado del cuestión de los resultados obtenidos hasta el momento con el objetivo de minimizar la complejidad de los costes de computación. Se propone reemplazar el algoritmo A* justificando teóricamente su aplicación con trabajo previo; se acompaña de un estudio detallado de las heurísticas con las que debe trabajar éste para mejorar la eficiencia de la búsqueda. En la sección 2 se detalla el proceso de la selección de unidades. En la sección 3 se introduce el algoritmo A*, se compara con el algoritmo de Viterbi clásico y se justifica su utilización. En la sección 4 se detalla la implementación y se exponen las distintas heurísticas. La sección 5 detalla el conjunto de pruebas realizadas y los resultados obtenidos; éstos son discutidos en la sección 6. Finalmente se detallan las conclusiones y las líneas de futuro de este trabajo.

2. PROCESO DE SELECCIÓN DE UNIDADES

A continuación se detallan las características fundamentales del módulo de selección de unidades del sistema de CTH utilizado en este trabajo de investigación.

Ajuste de los datos por fonema a nivel de unidad

La unidad de trabajo de predicción a partir del texto es la prosodia por fonema, pero en nuestro caso se establece por difonemas puesto que trabajamos con selección de unidades. Es necesario adaptar la prosodia del fonema a

Este trabajo ha sido financiado, en parte, por el proyecto SALERO IST-FP6-027122 de la Comisión Europea.

nivel de difonema. Considerando que las unidades (tanto difonemas como trifonemas) estan formadas por semifonemas (ver figura 1), el primer paso consiste en dividir los fonemas en semifonemas y en establecer los semifonemas como unidad mínima de representación del modelado prosódico. El parámetro prosódico más sencillo de adaptar es la duración: se distibuye equitativamente la duración del fonema entre los dos semifonemas.

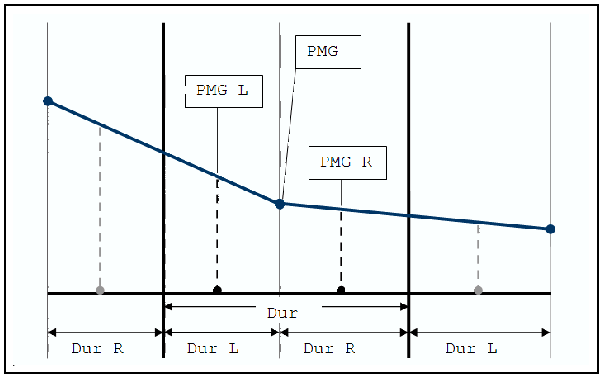

Figura 1. Recta de interpolación de la prosodia para pasar de fonema a semifonema

Siguiendo con los requisitos, transformar el *pitch* y la energía a semifonema es un proceso más complejo. Se consideran los requisitos presentados en el fonema como valores en el punto central (estable) del difonema. Cuando se divide el fonema en semifonemas los requisitos se conservan pero en el extremo de cada uno de los semifonemas: el extremo derecho en el semifonema izquierdo y extremo izquierdo en el semifonema derecho. Pero las características deseadas son las del punto central del semifonema, no las de los extremos. Para obtener estos datos se observará la tendencia contextual de la unidad teniendo en cuenta las prosodias del fonema anterior y del posterior. De esta forma se limitan los extremos de la variación prosódica más allá de la unidad. La recta de tendencia determinará el valor en los puntos requeridos. En el algoritmo 1 se dan más detalles del proceso.

Algorithm 1 Algoritmo de conversión de fonemas a semifonemas

procedure $splitFon(i)$

1: $Dur_L^i \Leftarrow Dur^i/2$
2: $Pitch_L^i \Leftarrow Pitch^{i-1} + (\frac{Dur_L^i}{2} * \frac{Pitch^i - Pitch^{i-1}}{Dur_R^{i-1} + Dur_L^i})$
3: $Energy_L^i \Leftarrow$
$Energy^{i-1} + (\frac{Dur_L^i}{2} * \frac{Energy^i - Energy^{i-1}}{Dur_R^{i-1} + Dur_L^i})$
4:
5: $Dur_R^i \Leftarrow Dur^i/2$
6: $Pitch_R^i \Leftarrow Pitch^i + (\frac{Dur_R^i}{2} * \frac{Pitch^{i+1} - Pitch^i}{Dur_R^i + Dur_L^{i+1}})$
7: $Energy_R^i \Leftarrow$
$Energy^i + (\frac{Dur_R^i}{2} * \frac{Energy^{i+1} - Energy^i}{Dur_R^i + Dur_L^{i+1}})$

Si se dispone la prosodia para semifonemas trabajar por selección de unidades es immediato: todas las unidades están formadas por semifonemas y se pueden aplicar las mismas funciones de interpolación para determinar la prosodia de cada unidad.

Selección de Unidades

Para determinar qué unidades del corpus se seleccionan, se usa un algoritmo de programación dinámica que escoje la mejor combinación de unidades basándose en minimizar una función de coste de selección. Esta función de coste está desglosada en un coste de unidad (*target*) C^t (ver ecuación 2), que evalúa la diferencia entre la unidad candidata y la unidad objetivo, y en un coste de concatenación C^c equación 3, que evalúa la bondad de la unión entre las unidades [1].

$$C(t_1^n, u_1^n) = \sum_{i=1}^{n} C^t(t_i, u_i) + \sum_{i=2}^{n} C^c(u_{i-1}, u_i) \quad (1)$$

$$C^t(t_i, u_i) = \sum_{j=1}^{p} w_j^t C_j^u(t_i, u_i) \quad (2)$$

$$C^c(u_{i-1}, u_i) = \sum_{j=1}^{q} w_j^c C_j^c(u_{i-1}, u_i) \quad (3)$$

Donde t_1^n representa la secuencia de unidades objetivo $\{t_1, t_2, \ldots, t_n\}$ y u_1^n representa la secuencia de unidades candidatas $\{u_1, u_2, \ldots, u_n\}$.

$$C_j^t(t_i, u_i) = \frac{\left|P_j(t_i) - \overline{P_j}(u_i)\right|}{\sigma_{P_j}} \quad (4)$$

$$C_j^c(u_{i-1}, u_i) = \frac{\left|P_j^R(u_{i-1}) - P_j^L(u_i)\right|}{\sigma_{P_j}} \quad (5)$$

Donde $\overline{P_j}(\cdot)$ indica el valor medio del subcoste para la unidad analizada.

La selección aplicando Viterbi al proceso se realiza en dos fases. En la primera fase se crea una matriz de distancias entre todas las unidades candidatas y se asigna el coste asociado a cada una de las realizaciones. Una vez se ha obtenido la matriz de distancias se acumulan los costes de cada camino posible y se escoge el camino con el menor coste acumulado, obteniendo ya las unidades seleccionadas. La ponderación de los pesos de la selección (w_j^i) de las unidades de síntesis se puede realizar con procesos automáticos como MLR[1] o GA[4] o mediante procesos interactivos como en [11] y en [12].

La implementación del algoritmo de Viterbi aplicada a grandes bases de datos dificulta la tarea de realizar síntesis de calidad en tiempo real. En [10] se detallan los factores clave para deteminar la velocidad de síntesis para sistemas basados en selección de unidades: la busqueda

que debe ser preprocesada y la eficiencia del coste computacional de la red de búsqueda.

En este artículo se propone reemplazar el algoritmo de programación dinámica de Viterbi, que devuelve los mejores caminos que presentan una distancia mínima, por un algoritmo que sólo encuentra la mejor distancia (A*).

3. ALGORITMO A* VS. VITERBI

Trabajo relacionado

Un primer enfoque de trabajo con los dos algoritmos para selección de unidades es empleado por [8] al usar transductores de estados finitos para hacer una busqueda probabilística por Viterbi primero, y seleccionar la mejor sequencia de unidades con un A*.

Principalmente, el marco de investigación donde se han comparado estos algoritmos es en procesado del lenguaje natural (PLN) para el estudio de gramáticas (gramáticas probabilisticas incontextuales - *PCFG*). En [13] se detalla la comparativa entre A* y Viterbi concluyendose que se obtiene la mejor solución en ambos casos y que en el peor caso de ordenamiento de heurísticas tienen el mismo coste computacional. Se destacan como ventaja la senzillez de construir el modelo A* reduciendo la complejidad computacional por transición.

En [7] se expone una comparativa entre los dos algoritmos trabajando en la decodificación dinámica posterior a la clasificación acústica para el reconocimiento del habla (ASR).

Siguiendo el razonamiento de [14] se justifica la utilización de Viterbi en selección de unidades por la tendencia y buenos resultados en ASR.

En base a estos trabajos consideramos utilizar A* en lugar de Viterbi por un cambio de paradigma de búsqueda en el problema. En la selección de unidades el hecho de obtener la función de coste es computacionalmente complejo y, a la vez, resulta factible estimar heurísticas por la similitud de los costes y el conocimiento previo de sus estadísticas.

El algoritmo de Viterbi

El algoritmo de Viterbi se divide en la parte de *forward* y una parte *backward*. Las principales aplicaciones del algoritmo de Viterbi son en problemas de estimación y detección en comunicaciones digitales y procesado de la señal.

En procesado de la señal en reconocimiento del habla donde se modela la señal de voz con modelos ocultos de Markov (HMM). Dado un estado inicial y un estado final definidos, devuelve el orden de los caminos que tienen menor probabilidad de error en los estados probabilísticos intermedios que los conforman[15]. En la fase de *forward* se acumula el error probabilístico para cada camino posible (sucesión de posibles estados con sus transiciones), sobreviviendo solo los caminos con menor error para cada estado. Una vez llegado al estado final, se procede en un proceso *backward* para reconstruir el camino de menor error hasta el estado inicial.

El algoritmo A*

A* es un algoritmo clásico de búsqueda *best-first seach* estructurado en árbol, para la resolución de problemas en grafos. Fue formulado por Hart, Nilsson y Raphael (1968), revisado en 1969 y resumido por Nilsson en 1971 [16]. A diferencia del paradigma de la programación dinámica, el A* se basa en la decodificación de una pila de estados de búsqueda, donde el modelo de búsqueda es un arbol con las hipótesis parciales representadas por las distintas ramas hasta las hojas. Un concepto nuevo en el paradigma A* es la búsqueda heurística, ya que las heurísticas son un factor clave del éxito del algoritmo A*. Una heurísticas pobres determinan un comportamiento similar a Viterbi o *backtracking* en terminos de coste computacional (y con más consumo de memoria).

A diferencia del algoritmo de Greedy, considerado como algoritmo heurístico puro, A* mantiene las propiedades de realizar una búsqueda óptima y completa [16], pero en cambio, prioriza el orden de búsqueda en función de heurísticas inherentes del problema. De este modo, se reduce el coste computacional de un algoritmo exaustivo de búsqueda como el *backtracking* aunque la reducción tiene relación con la calidad de las heurísticas. La principal ventaja del algoritmo A* es la computación del coste de un vértice y viene determinado por dos factores: sólo se computan los costes según la recomendación de la función heurística, evitando el cálculo de la totalidad de ellos, y no es necesario computarlos más de una vez. Por consiguiente es recomendable para problemas donde la evaluación de un estado es computacionalmente complejo, por ejemplo la función de coste en selección de unidades. En cambio, el principal inconveniente del A* es la memoria de la pila de búsqueda: para limitar el tamaño de la pila de memoria con problemas complejos existen las variantes IDA* (Iterative Deeping A*) y SMA* (Simplified Memory-bounded A*) [16] aunque no garantizan la mejor solución al limitar la búsqueda.

4. IMPLEMENTACIÓN

Consideraciones Iniciales

El algoritmo principal que rige el A* se detalla en el algoritmo 2. La filosofía del algoritmo es memorizar estados (nodos) de la búsqueda en cada iteración, asociandoles un coste a cada uno. Para proceder así almacena una lista de nodos para visitar y una lista de nodos visitados. A medida que va progresando la búsqueda actualiza los costes de los nodos ya visitados anteriormente. Dicho coste se calcula mediante la ecuación 6 dónde el coste f del nodo n es el acumulado hasta el nodo por el camino que se está iterando, más la estimación del coste de dicho

Algorithm 2 Algoritmo de busqueda A*

procedure doAStarSearch($start$,$goal$)

```
1:  openList ⇐ {start}
2:  closedList ⇐ ∅
3:  while openList ≠ ∅ do
4:    cur ⇐ openList.popFirst()
5:    if cur = goal then
6:      return cur
7:    else
8:      for all successor ∈ cur.getSuccessors() do
9:        cost = cur.Cost(successor)
10:       cost = cost + succesor.goalHeuristic()
11:       if succesor ∈ openList then
12:         if cost < openList.cost(successor) then
13:           openList.replaceNode(successor, cur, cost)
14:         end if
15:       else
16:         if succesor ∈ closedList then
17:           if cost < closedList.cost(successor)
              then
18:             closedList.removeNode(successor)
19:           end if
20:         end if
21:         openList.addNode(successor, cur, cost)
22:       end if
23:     end for
24:     closedList.addNode(cur)
25:   end if
26:   openList.sortCost()
27: end while
```

nodo hasta el nodo objetivo.

$$f(n) = g(n) + h'(n) \tag{6}$$

En el caso de la selección de unidades el nodo inicial es una sequencia vacía de unidades seleccionadas y el estado objetivo es la secuencia con todas las unidades seleccionadas. El coste $g(n) = C(t_1^n, u_1^n)$ se calcula como se define en la ecuación 1. Para mejorar la eficiencia del algoritmo se implementan optimizaciones basadas en memoria caché para los costes de *target* y concatenación tal como se propone en [3]

Estudio de las Heurísticas

Las heurísticas son un factor clave para la eficacia del algoritmo. Para encontrar una solución rápidamente se necesita una heurística que no sobreestime el coste de pasos hacia la meta.

El factor efectivo de branqueo (b*) es un método propuesto [16] para evaluar la calidad de una heurística. Su fórmula se define en la ecuación 7, donde N es el número de nodos expandidos para la búsqueda y d es la profundidad de la solución en el árbol.

$$N = 1 + b^* + (b^*)^2 + \cdots + (b^*)^d \tag{7}$$

Por ejemplo, si para seleccionar 12 unidades se exploran 320 nodos, entonces $b^* = 1{,}47$. Este factor se suele mantener constante aunque aumenten las unidades a seleccionar. Una buena heurística debe tener un factor cercano a 1, permitiendo así soluciones eficaces en problemas grandes. Al probar varias heurísticas para el mismo problema pero con distintas d, si una heurística h_1 es en todos los casos mejor que una heurística h_2, entonces se puede aplicar el concepto de dominancia pues h_1 domina a h_2 ($h_1 \geq h_2$): eso conlleva que la heurística h_1, en media, produzca menos nodos que h_2. En el mismo trabajo se concluye que siempre es "mejor utilizar una función heurística con valores altos mientras no sobreestime la d". Otra conclusion del trabajo es la definición de múltiples heurísticas para finalmente el valor máximo (*worst-case*) proporcionado en cada una de ellas. Finalmente, se recomienda trabajar con datos estadísticos si hay experiencia previa del mismo.

5. PRUEBAS Y RESULTADOS

Para implementar las heurísticas se define un benchmark compuesto de 3 frases, 48 configuraciones de pesos diferentes y un barrido de 4 heurísticas globales para todo el corpus. Seguidamente se procede a explicarlo en detalle:

1. *Frases*: Se han seleccionado para sintetizar 3 frases dist ("mantenen discretes","reunit amb els consellers","tenen amb els"). Dichas frase son fruto de un proceso de previo para determinar aquellas frases que seleccione un mayor numero de frases que presenten mayor variabilidad.

2. *Pesos*: Se han iniciado unos 144 conjuntos de pesos (48 para cada frase) que aportan un buen barrido de combinaciones para así proporcionar distintos costes en cada estado.

3. *Heurísticas* Las heurísticas se han implementado a nivel global por lo tanto aplica la misma en cada estado de la búsqueda. El resumen de las heurísticas efectuadas se presenta en las equaciones 9 donde N_u es el nombre de unidades totales por seleccionar y N_{us} el numero de unidades ya seleccionados en ese estado.. La estimación de profundidad se realiza en función de las unidades que quedan por seleccionar. Como los costes normalizados se mueven en el intervalo [0,1] se prueban distintus valores en su intervalo de más freqüencia.

Para analizar los resultados se hace en función de tres parametros: se analiza el factor de branqueo eficaz (*b**) ya que según [17] es el método ideal para evaluar una heurística independientemente del tamaño del problema. Se analizan, a la vez, los parámetros clásicos de tiempo de ejecución y número de nodos explorados. Aun así, el

Heurística	Frase	Search Steps	B*	Tiempo Ejecución
VITERBI	Tenen amb els	$59774 \pm 0,00$	$2,39 \pm 0,00$	$29,65 \pm 0,56$
VITERBI	Mantenen discretes	$45843 \pm 0,00$	$1,79 \pm 0,00$	$23,06 \pm 56$
VITERBI	Reunit amb els consellers	$45508 \pm 0,00$	$1,59 \pm 0,00$	$21,98 \pm 0,67$
0,666	Tenen amb els	$640,12 \pm 262,85$	$1,55 \pm 0,11$	$26,60 \pm 11,71$
0,666	Mantenen discretes	$787,10 \pm 588,69$	$1,31 \pm 0,12$	$27,50 \pm 20,38$
0,666	Reunit amb els consellers	$890,68 \pm 519,99$	$1,25 \pm 0,10$	$20,14 \pm 10,69$
0,75	Tenen amb els	$640,12 \pm 262,85$	$1,55 \pm 0,11$	$26,95 \pm 11,69$
0,75	Mantenen discretes	$787,10 \pm 588,69$	$1,32 \pm 0,12$	$28,02 \pm 20,71$
0,75	Reunit amb els consellers	$890,68 \pm 519,99$	$1,25 \pm 0,10$	$20,52 \pm 10,83$
0,85	Tenen amb els	$427,66 \pm 481,60$	$1,39 \pm 0,22$	$16,31 \pm 19,17$
0,85	Mantenen discretes	$691,12 \pm 823,86$	$1,24 \pm 0,17$	$24,14 \pm 28,62$
0,85	Reunit amb els consellers	$473,54 \pm 772,12$	$1,14 \pm 0,12$	$10,79 \pm 16,56$
0,92	Tenen amb els	$427,66 \pm 481,60$	$1,39 \pm 0,22$	$16,25 \pm 19,03$
0,92	Mantenen discretes	$691,12 \pm 823,86$	$1,24 \pm 0,17$	$23,97 \pm 28,21$
0,92	Reunit amb els consellers	$473,54 \pm 772,12$	$1,14 \pm 0,12$	$10,75 \pm 16,32$
1	Tenen amb els	12 ± 0	$0,98 \pm 0$	$1,39 \pm 0,49$
1	Mantenen discretes	17 ± 0	$0,99 \pm 0$	$1,54 \pm 0,50$
1	Reunit amb els consellers	21 ± 0	$0,99 \pm 0$	$1,75 \pm 0,43$

Tabla 1. Resultados obtenidos con los tres parámetros de evaluación para todo el *benchmark* de pruebas

tiempo de ejecución no es una medida determinante debido a que puede variar dependiendo de la gestión de procesos del sistema operativo. Los resultados obtenidos se muestran en la tabla 5 y en las gráficas comparativas de tiempo 2, de factor de branqueo 3 y de nodos visitados 4 (observar que se ha trazado con escala logarítmica).

$$d_{left} = N_u - N_{us} \quad (8)$$

$$h'(n) = d_{left} * 0{,}666 \quad (9)$$

$$h'(n) = d_{left} * 0{,}750 \quad (10)$$

$$h'(n) = d_{left} * 0{,}850 \quad (11)$$

$$h'(n) = d_{left} * 0{,}920 \quad (12)$$

$$h'(n) = d_{left} * 1{,}000 \quad (13)$$

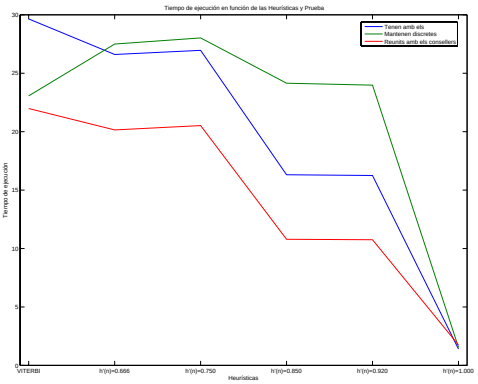

Figura 2. Tiempo de ejecución detallado para cada heurística y prueba

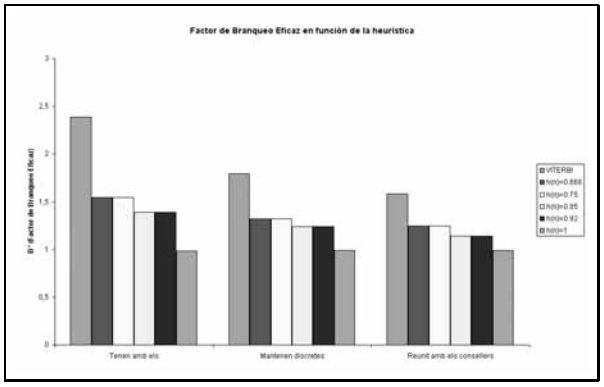

Figura 3. Factor de branqueo eficaz en función de las pruebas y heurísticas

6. DISCUSIÓN

Una vez planteados los resultados se puede observar que el A*, si se tiene en cuenta la exploración de nodos y el factor de branqueo eficaz, es drásticamente mejor que el algoritmo de Viterbi. También se observa, como ya se ha mencionado en trabajos anteriores, que el tiempo de ejecución en sus peores heurísticas (ver ecuaciones 9 y 10), se comporta de forma equivalente al algoritmo de Viterbi; en cambio, una buena heurística (ver ecuacion 13) puede acelerar el proceso. A la vez, los resultados validan la teoría que la heurística más pesimista (ver ecuacion 13) es la que se comporta mejor siempre y cuando no sobreestime la profundidad del árbol de búsqueda.

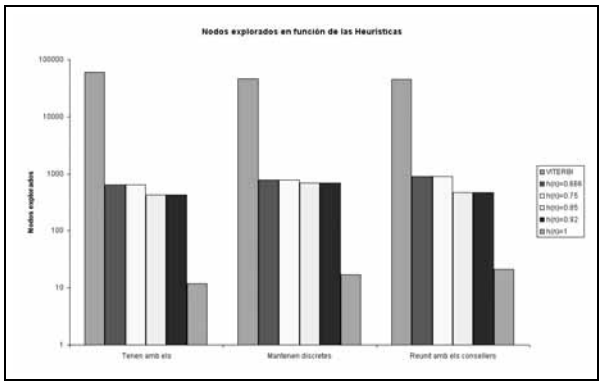

Figura 4. Nodos explorados en función de las pruebas y heurísticas

7. CONCLUSIONES Y LÍNEAS DE FUTURO

En este trabajo se aporta un nuevo algoritmo para la selección de unidades, justificando su elección en los trabajos anteriores y en los resultados obtenidos en nuestro caso. Se abre un nuevo frente de investigación sobre heurísticas de este algoritmo, puesto que este trabajo no deja de ser un primer enfoque.

En un trabajo futuro se plantea efectuar la selección A* con clusterización del espacio de búsqueda con árboles de decisión, y aplicar el trabajo hecho con estados finitos de transducción. Aún así, los resultados obtenidos son prometedores y representan un avance cualitativo y cuantitativo en el estudio de la eficacia de la selección de unidades en CTH-SU en tiempo real.

8. BIBLIOGRAFÍA

[1] A. Hunt y A. W. Black, "Unit selection in a concatenative speech synthesis system using a large speech database," in *Proceedings of ICASSP*, Atlanta, USA, 1996, vol. 1, pp. 373–376.

[2] A. W. Black y P. Taylor, "Automatically clustering similar units for unit selection in speech synthesis," in *Proceedings of EuroSpeech*, Rodas, Grecia, 1997, pp. 601–604.

[3] M. Beutnagel, M. Mohri, y M. Riley, "Rapid unit selection from a large speech corpus for concatenative speech synthesis," in *Proceedings of EuroSpeech*, Budapest, Hungría, 1999, vol. 2, pp. 607–610.

[4] F. Alías y X. Llorà, "Evolutionary weight tuning based on diphone pairs for unit selection speech synthesis," in *Proceedings of the 8th European Conference on Speech Communication and Technology (EuroSpeech)*, 2003.

[5] F. Alías, X. Llorà, I. Iriondo, y L. Formiga, "Ajuste subjetivo de pesos para selección de unidades a través de algoritmos genéticos interactivos," *Procesamiento del Lenguaje Natural*, vol. 31, pp. 75–82, Septiembre 2003.

[6] Y. Meron y K. Hirose, "Efficient weight training for selection based synthesis," in *Proceedings of EuroSpeech*, Budapest, Hungría, 1999, vol. 5, pp. 2319–2322.

[7] Nikko Ström, "Automatic continuous speech recognition with rapid speaker adaptation for human/machine interaction - PhD Thesis," 1997.

[8] J Yi y J. Glass, "Natural-sounding speech synthesis using variable-length units," 1998.

[9] M Macon, A. Cronk, y J. Wouters", "Generalization and discrimination in tree-structured unit selection," 1998, note: "Proc. 3 rd Synthesis Workshop, Nov. 1998.

[10] A. Conkie, M. C. Beutnagel, A. K. Sydal, y P. E. Brown, "Preselection of candidate units in a unit selection-based text-to-speech synthesis system," in *Proceedings of the 6th International Conference on Spoken Language Processing (ICSLP)*, Beijing, China, 2000, vol. 3, pp. 314–317.

[11] F. Alías, X. Llorà, L. Formiga, K. Sastry, y D. E. Goldberg, "Efficient interactive weight tuning for tts synthesis: reducing user fatigue by improving user consistency," in *Proceedings of ICASSP*, Toulouse, Francia, 2006, vol. I, pp. 865–868.

[12] A. K. Syrdal y A. D. Conkie, "Perceptually-based data-driven join costs: Comparing join types," in *Proceedings of the 9th International Conference on Speech Communication and Technology (InterSpeech)*, Lisboa, Portugal, 2005, pp. 2813–2816.

[13] Dan Klein y Christopher D. Manning, "A* parsing: Fast exact viterbi parse selection," in *Proceedings of HLT-NAACL 03*, 2003.

[14] M. Ostendorf y I. Bulyko, "The impact of speech recognition on speech synthesis," in *Proceedings of 2002 IEEE Workshop on Speech Synthesis*, Santa Monica, USA, 2002.

[15] H. L Lou, "Implementing the viterbi algorithm," *Signal Processing Magazine, IEEE*, vol. 12, no. 5, pp. 42–52, 1995.

[16] Stuart Russell y Peter Norvig, *Artificial Intelligence: A modern approach*, Prentice Hall, 1995.

[17] T Dutoit y H. Leich, "Mbr-psola: Text-to-speech synthesis based on an mbe re-synthesis of the segments database," *Speech Communication*, vol. 13, pp. 435–440, Junio 1996, note:.