Resolución
de problemas de búsqueda

Memoria de Prácticas de Inteligencia Artificial

Primera Entrega

11 de enero de 2009

Autor:

Daniel Fernández Núñez *danielfdezn@gmail.com*

Directorio de entrega: infdfn01

Resumen

En esta práctica se tratará de proponer soluciones para un simple juego recreativo formulándolo como un problema de búsqueda. El juego consiste en que a partir de un número de tres cifras y, mediante incrementos y decrementos en una unidad de sus dígitos, se debe llegar a un número meta teniendo en cuenta una serie de restricciones. Se considerarán distintos algoritmos de búsqueda ciega e informada, se evaluarán y se extraerán las conclusiones pertinentes.

Índice

1. Introducción

El problema que nos ocupa es un sencillo cuyo objetivo es el de a partir de un número de 3 dígitos y con una serie de modificaciones sobre estos llegar tendremos que llegar a un número destino. Las modificaciones consisten en que sólo podremos incrementar o disminuir en una unidad de cada vez al dígito que elijamos. Sin embargo, no podremos disminuir un dígito para pasar de 0 a 9 ni aumentarlo para ir de 9 a 0, por lo que tendremos que hacer 9 modificaciones si queremos realizar este caso extremo. Otra restricción es que no podremos modificar dos veces de forma consecutiva a un dígito, lo cual podría complicar las cosas una vez estemos acercándonos al número meta. Existen también un conjunto de números prohibidos definidos a priori por los que no podremos pasar al intentar llegar al resultado final, por lo que dependiendo de cómo esté configurado este conjunto es posible que nunca podamos llegar a una solución.

Por consiguiente un movimiento válido consiste en modificar un dígito en una unidad tal que no se salga del rango [0,9], que este dígito no se haya modificado en el movimiento anterior y que el número resultado no esté prohibido. De aquí deducimos que como máximo vamos a tener 6 movimientos en el caso inicial y 4 en los posteriores, y esta cifra se disminuirá dependiendo de si estamos en los límites o por culpa de los número prohibidos.

El objetivo del juego, formulado como un problema de búsqueda, es obtener la secuencia de movimientos o camino que se debe realizar para, partiendo de un estado inicial dado, llegar hasta el estado solución.

2. Definiciones formales

- **Estado inicial:** puede ser cualquier combinación de tres cifras, aunque si queremos mantener cierta coherencia, debemos evitar que sea un número prohibido.
- **Estado meta:** al igual que el estado inicial, una combinación de tres cifras cualquiera. Este número no debe estar contenido en el conjunto de números prohibidos si queremos que en un principio finalice.
- **Representación de estados:** el número se representará por un vector de tres posiciones de números enteros. Por ejemplo, para el número 748 tendremos que:

$$V(3) = 7; V(2) = 4; V(1) = 8;$$

- **Conjunto de operadores y restricciones:** se trata de definir operaciones tales que cumplan con las restricciones dadas en el enunciado. Los operadores y restricciones para cada dígito i son mostrados en la tabla 1, en la cual V es el estado actual, V'' es el estado antecesor, V' es el estado sucesor y B es el conjunto de estados prohibidos. Por lo tanto, como en este caso solo jugamos con un número de 3 dígitos, tendremos 6 operadores posibles con 3 restricciones a verificar por cada uno de ellos.
- **Prueba de meta:** comprobamos si el vector coincide dígito a dígito con el número meta y, de ser así, finalizamos la búsqueda.
- **Función de coste:** es 1 para todas las operaciones, por lo que el coste total del camino para una solución es el número de operaciones invertidas en el proceso.

Tabla 1: Operadores y restricciones

Operador	Precondiciones	Resultado	Comentario
INCi	$V(i) < 9$ $V(i) = V''(i)$ $V(3) \times 100 + V(2) \times 10 + V(1) + 10^{i-1} \notin B$	$V'(i) = V(i) + i$	Incrementar dígito i
DECi	$V(i) > 1$ $V(i) = V''(i)$ $V(3) \times 100 + V(2) \times 10 + V(1) - 10^{i-1} \notin B$	$V'(i) = V(i) - i$	Decrementar dígito i

3. Análisis de métodos de búsqueda ciega

En este apartado se analizaran los distintos factores que influirán en la decisión de elegir entre búsqueda ciega por anchura o por profundidad.

3.1. Gasto en memoria

El número de estados total a priori sería el rango de enteros de entre 0 y 999, es decir, 1000 estados posibles. Sin embargo, debemos tener en cuenta la restricción de que no podemos modificar un dígito dos veces consecutivas, por lo que en cada estado tendremos que tener un método de saber cual ha sido el dígito modificado anteriormente.

En una primera aproximación podemos pensar en añadir una variable adicional al estado en la que se indica el dígito modificado. Así pues el número de estados a considerar sería de 4×1000, donde el el estado estaría representado por un vector de cuatro enteros donde los tres primeros son el número y el cuarto es otro entero que se corresponde con el dígito modificado.

Otra forma de implementar esto sería asegurarse de que el estado actual tenga un puntero al estado anterior y así poder obtener su número y saber cual es el dígito que se ha modificado. Con esto sólo necesitamos un vector de tres enteros en vez de los cuatro de la anterior solución, ya que en búsqueda ciega se suele mantener un puntero de un estado hacia su padre ya en el algoritmo y no se necesitaría guardarlo de forma explícita. Este ahorro de un entero en memoria puede parecer pequeño a priori pero puede ser vital para algoritmos exponenciales como anchura.

3.2. Optimalidad

Esta característica tiene que ver no tanto con este caso particular sino más por la implementación de los algoritmos de búsqueda que nos ocupan. En anchura se explorarán todos los posibles caminos desde el comienzo del algoritmo, por lo que en el momento en el que encuentre una primera solución ese camino será óptimo.

Sin embargo en profundidad si se encuentra una solución no podemos afirmar que será óptima ya que por la implementación del algoritmo es posible que se diera un rodeo

para llegar a esa solución.

3.3. Completitud

Antes de analizar esta característica sobre los algoritmos de búsqueda que nos ocupan, debemos analizar que si dados un estado inicial y otro meta cualquiera podremos siempre obtener una solución. Sin las restricciones es obvio que podremos llegar siempre al estado meta desde cualquier estado inicial con las operaciones que se nos permiten. La restricción de no modificar un dígito dos veces consecutivas tampoco es un impedimento. Un ejemplo:

> Estamos en el estado 345, el meta es el 346 y acabamos de modificar las unidades. Una de las posibles soluciones sería 345 ->335 ->336 ->346. Intuitivamente vemos que esta técnica se puede aplicar en cualquier situación.

La única forma en la que no podríamos llegar a un estado meta es que éste pertenezca al conjunto de los números prohibidos y que la ruta al numero destino esté bloqueada por números prohibidos (por ejemplo, la meta es 999 y {998, 989,899} es el conjunto de números prohibidos).

Usando la búsqueda en anchura y suponiendo que el estado final no está en el conjunto de números prohibidos ni bloqueado por estos, siempre vamos a encontrar una solución y va a ser óptima. Con busqueda en profundidad también encontraremos siempre una solución si garantizamos que el algoritmo no va a entrar en un bucle, pero en este caso la solución no tiene porqué ser óptima.

3.4. Factor de ramificación

El factor de ramificación es por definición el número medio de estados posibles que pueden derivar de cada una de las configuraciones. Esto es importante ya que nos ayuda a saber el espacio que ocupara la búsqueda en anchura. Suponiendo que todas las posibles configuraciones son equiprobables, que no tenemos en cuenta números prohibidos y que descartaremos los movimientos pertinentes por la restricción de no modificar un dígito dos veces consecutivas, distinguiremos los siguientes casos:

1. **Los tres dígitos son** 0 **o** 9**:**
   $000, 999, 900, 909, 990, 099, 090, 009$
   (Variaciones con repeticion de 2 elementos tomados de 3 en 3)
   $2 \text{ mov} \times 2^3 = 2 \times 8 = 16 \text{ mov}$

2. **Dos dígitos son** 0 **o** 9 **y uno es** $0 < x < 9$**:**
   $00x, 0x0, x00, 99x, 9x9, x99, 90x, 09x, 9x0, 0x9, x90, x09$
   (Variaciones con repeticion de 2 elementos tomados de 2 en 2 por 3 casos)
   Supongamos que la probabilidad de que un dígito sea el que se haya modificado en el estado anterior es equiprobable:
   $((2/3 \times 3 \text{ mov}) + (1/3 \times 2 \text{ mov})) \times (2^2 \times 3)) \times 8 = (2{,}66) \times 96 = 256 \text{ mov}$

3. **Un dígito es** 0 **o** 9 **y dos son** $0 < x < 9$**:**
   $0xx, 9xx, x0x, x9x, xx0, xx9$
   (Variaciones con repeticion de 2 elementos tomados de 1 en 1 por 3 casos)
   $((2/3 \times 3 \text{ mov}) + (1/3 \times 4 \text{ mov})) \times (2 * 3) \times 8^2 = (3{,}333) \times 384 = 1280 \text{ mov}$

4. **Los tres dígitos son** $0 < x < 9$**:**
   (Variaciones con repetición de 8 elementos tomados de 3 en 3)
   $4 \text{ mov} \times 8^3 = 4 \times 512 = 2048 \text{ mov}$

Teniendo en cuenta que tenemos 1000 casos totales, entonces el factor de ramificación b es:

$$b = \frac{16 + 256 + 1280 + 2048}{1000} = 3{,}6 \tag{1}$$

Este resultado resulta lógico puesto que lo más probable es que el estado actual se encuentre en el "interior"del rango de números, donde el número de movimientos es 4, así que es normal que la media de estados se situe cerca de este.

Ahora intentemos aproximar el número medio de profundidades que le llevará al algoritmo de busqueda en anchura para encontrar una solución. Ya que son todos casos equiprobables calcularemos el caso más desfavorable y dividiremos esa profundidad entre 2. No es una aproximación muy exhaustiva pero valdrá como ejemplo.

> Supongamos estado inicial 000 y estado meta 999. Uno de los caminos óptimos sería: 000 ->001 ->011 ->111 ->112 ... que son 9 + 9 + 9 = 27 movimientos o niveles de profundidad.

Si tomamos como que media de pasos es el caso más desfavorable entre 2, tenemos que $27/2 \approx 14$ niveles de media, por lo que en memoria tendriamos almacenados aproximadamente $3{,}6^{14} \approx 6{,}14 \times 10^7$ estados.

3.5. Conclusiones

Lo más óptimo sería utilizar busqueda en anchura, ya que nos garantiza encontrar siempre que sea posible un camino óptimo y en un tiempo razonablemente corto. Sin embargo, estamos sujetos a la disponibilidad de recursos de memoria para usar este algoritmo:

> Cada uno de los dígitos es un número de 0 a 9 y se puede expresar con 4 bits. Por lo tanto cada estado ocuparía en memoria $3 \times 4 = 12$ bits = 1,5 Bytes. En el caso medio de 14 profundidades con ramificación efectiva de 3,6 esto se traduciría en un tamaño medio en memoria de $3{,}6^{14} \times 1{,}5 =$ 87,84 MB, y eso sin contar el tamaño ocupado por los punteros y demás variables del programa. Esto aún podría ser asumible, pero en el peor caso de 27 profundidades ocuparía $3{,}6^{27} \times 1{,}5 = 1{,}395$ PetaBytes, una cantidad de memoria inasumible en la actualidad.

Por lo tanto tendremos que estar sujetos a la disponibilidad de memoria, limitando el número de profundidades de búsqueda y además controlar también el consumo de recursos. Si el algoritmo de búsqueda en anchura no consigue un solución antes de llegar al límite de consumo de recursos, entonces es cuando se debe acudir al algoritmo de busqueda en profundidad.

4. Heurísticas

En este apartado propondremos dos heurísticas para usar con los algoritmos de búsqueda informada y analizaremos sus características.

4.1. Heurística de los dígitos incorrectos

Consiste en contar el número de dígitos que no son iguales a los del número meta.

Expresión matemática. Sean a_1, a_2, a_3 los tres dígitos que componen el número del estado actual n y sean f_1, f_2, f_3 los del estado meta. Entonces se define la heurística como:

$$h_1(n) = \sum_{i=1}^{3} S_i \tag{2}$$

donde S_i es una función que se define como:

$$S_i = \begin{cases} 0 & a_i = f_i \\ 1 & \text{otro caso} \end{cases} \tag{3}$$

Idoneidad. Esta heurística resulta de relajar las restricciones del problema original. Se ha simplificado de forma que podríamos suponer que podemos llegar al dígito meta en un sólo movimiento, es decir, esta heurística nos dice el mínimo número de movimientos para llegar al estado final. Es la mayor abstracción del problema original que podríamos conseguir, por lo que es una heurística admisible.

Demostración de no sobreestimación. Sabemos que no va a sobreestimar por lo dicho en el apartado anterior: la heurística se ha obtenido a partir de la abstracción del problema original. Por lo tanto, el número de movimientos que necesitaríamos para llegar a la meta será mayor con todas las restricciones que con la heurística, por lo que:

$$\forall n, h_1(n) \leq h^*(n)$$

donde $h(n)$ es la función que devuelve el coste óptimo para llegar a meta.

Mínimos locales. Esta heurística tiene un grave problema que ilustraremos en el siguiente ejemplo:

> Supongamos que estamos en el estado 321 y que el meta es 765. Aquí $h_1(321) = 3$ puesto que ninguna de los dígitos está en meta. Sin embargo, hagamos el movimiento que hagamos, la función seguirá siendo 3 durante varios movimientos aunque vayamos en la dirección correcta.

Esto se conoce como una *meseta*, un área plana donde todos los estados tienen el mismo valor de la función heurística y es imposible determinar la dirección correcta. Una de las soluciones más usuales sería realizar un salto en el espacio de estados para tratar de encontrar una región diferente, aunque en este caso particular lo más probable es que nos encontrásemos de nuevo en una meseta. Este problema hace que esta heurística sea poco recomendable para este problema.

4.2. Heurística de las distancias euclideas

Esta heurística se define como la suma de las distancias euclideas de cada dígito a su correspondiente del número meta.

Expresión matemática. Sean a_1, a_2, a_3 los tres dígitos que componen el número del estado actual n y sean f_1, f_2, f_3 los del estado meta. Entonces se define la heurística como:

$$h_2(n) = \sum_{i=1}^{3} |a_i - f_i| \tag{4}$$

Idoneidad. Esta heurística se obtiene mediante una abstracción de las restricciones del problema original suponiendo que podemos modificar un dígito en tantos movimientos como indica la distancia euclidea para llegar al dígito meta, sin tener en cuenta las restricciones de los números prohibidos ni de no podemos modificar un dígito dos veces consecutivamente. Es decir, es una cota inferior de la función de coste del problema original, por lo que es un heurística admisible de este.

Demostración de no sobreestimación. Como ya se dijo, la heurística es una abstracción del problema original y no va a sobreestimar ya que, aunque la distancia euclidea de el número exacto de pasos para ir de un dígito hasta al meta, siempre podemos tener problemas con las restricciones que nos harán incrementar el número de pasos, por lo que:

$$\forall n, h_2(n) \leq h^*(n)$$

donde $h^*(n)$ es la función que devuelve el coste óptimo para llegar a meta.

Mínimos locales. Esta heurística no tiene el grave problema de las mesetas de la heurística anterior, pero si tiene otro tipo que se pondrá de manifiesto en el siguiente ejemplo:

> Supongamos que nuestro estado actual es 654 y que el meta es 754 y que el último dígito modificado es el de las centenas. Aquí tenemos que $h_2(654) = 1$,es decir, que nos falta un solo movimiento para llegar a la meta. Sin embargo, no podemos modificar las centenas por culpa de las restricciones iniciales, por lo que al modificar alguno de los otros dos el valor de la función heurística se nos va a incrementar irremediablemente a 2.

Este problema es conocido como de *máximos locales*, estados puntuales mejores que cualquiera de sus vecinos posibles pero peores que otros estados más lejanos. Cuando aparecen cerca de la solución, como es nuestro caso, se denominan *estribaciones*. Una de las soluciones que se suelen proponer para estos casos (y que en este problema funcionaría bien) es el de regresar a un nodo previo e intentar ir en una dirección diferente. Este problema no es tan grave como el de las mesetas, ya que una vez nos encontremos en la estribación no es necesario más que unos poco movimientos más para llegar al número final, lo que hace que esta heurística sea muy recomendable para este problema.

Referencias

[1] V. Moret, A. Alonso, M. Cabrero, B. Guijarro, E. Mosqueira. *Fundamentos de Inteligencia Artificial (2a Ed)*, capitulo Resolución de problemas pp. 17–60. Servicio de Publicaciones, UDC, 2000.