Sistemas de Inteligencia Artificial
T.P. Búsqueda: Senku

CARLA BARRUFFALDI, TOMÁS CERDÁ, LUCIANO BIANCHI, MARCELO LYNCH

Instituto Tecnológico de Buenos Aires

INTRODUCCIÓN

Se implementó un sistema de producción con el fin de ser utilizado para la resolución del juego Senku. Para este sistema se implementaron distintas estrategias de búsqueda no informada tales como *depth first*, *breadth first*, *uniform cost*, *profundización iterativa* y estrategias de búsqueda informada como *A** y *greedy search*. Con el fin de optimizar la búsqueda informada se propusieron distintas heurísticas y se evaluó el rendimiento de cada una de ellas.

Acerca del senku

El Senku es un juego de tablero en el cual todos los casilleros comienzan ocupados por una ficha excepto uno. El jugador debe mover una ficha por vez, pero su único movimiento posible consta de *capturar* una ficha vecina (horizontal o verticalmente, nunca en diagonal) mediante un salto sobre ella (figura 2), cayendo a la posición siguiente en esa dirección (que debe estar vacía). La *captura* de una pieza implica su remoción del tablero. El objetivo del juego es lograr eliminar todas las fichas excepto una, finalizando así con una única ficha en la posición central del tablero.

Existen variantes al juego original, con diferentes formas de tableros y posiciones "objetivo" para que quede la última pieza. En este trabajo se considera la versión original (tambien llamado *tablero inglés*) de 31 piezas, con el objetivo de dejar la última pieza en la posición central (ver figura 1).

Figura 1. Configuraciones inicial (izquierda) y final (derecha) del juego.

Figura 2. Un movimiento válido: la pieza roja "salta" la verde y la verde es removida del tablero.

IMPLEMENTACIÓN

Motor de búsqueda

El motor de búsqueda fue implementado de la forma más genérica posible a partir de las interfaces provistas por la cátedra; siendo así independiente de la base de conocimiento de nuestro problema en particular. Se muestra el pseudocódigo en el algoritmo 1. Una versión TREE SEARCH también fue implementada, si bien todos los experimentos fueron realizados a partir de la metodología GRAPH SEARCH.

Considerando que lo único que varía entre las implementaciones de las búsquedas *depth first*, *breadth first*, *uniform cost*, *A** y *greedy search* es el criterio a partir del cual se eligen los nodos a expandir de la frontera (*frontier*), basta con modificar la cola (*Queue*) a utilizar en la línea 2 del algoritmo para implementar cada uno de ellos.

La búsqueda *profundización iterativa* fue implementada a partir de una estrategia recursiva tal que existen a lo sumo d nodos en la memoria, siendo d la profundidad mínima a la solución.

Características del problema

Un análisis sobre las particularidades del Senku como problema de búsqueda, previo a la confección de las heurísticas, permitió mayor eficiencia en la implementación y guió el análisis posterior.

Camino a la solución

Considerando el Senku en su forma original se puede notar que el camino a la solución siempre tiene exactamente $N - 1$ pasos, siendo N la cantidad inicial de fichas en el tablero. Esto es porque cada movimiento remueve exactamente una ficha del tablero. En este sentido, el problema está en *encontrar el camino a la solución* más que encontrar un *camino mínimo*, y la función de las heurísticas es optimizar esta búsqueda, minimizando el backtracking.

Así, no tiene sentido en hablar de un *camino óptimo* a la solución, y por esto mismo la admisibilidad de las heurísticas (que garantiza encontrar un camino óptimo) no es una condicion directamente deseable en este caso. Es más: se conoce la heurística $h*$, que indica exactamente el costo del camino restante al objetivo (será $N - 1$, con N la cantidad de piezas en ese estado). Sin embargo, esta heurística no distingue entre *mejores* o *peores* estados sucesores (pues, en cuanto a costo, no los hay), y así no es muy valiosa para la búsqueda.

Simetrías

El Senku es un problema altamente simétrico, y puede explotarse este hecho para reducir considerablemente el espacio de búsqueda: de hecho, los primeros cuatro movimientos del juego resultan en configuraciones simétricas entre sí lo cual implica una poda de un 75 % de los estados solo considerando esta simetría (figura 3).

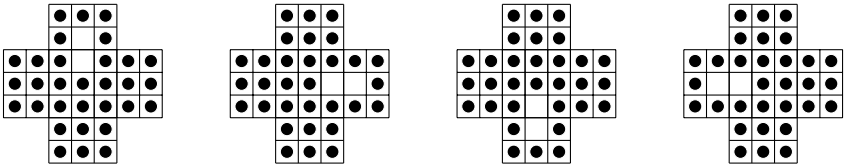

Figura 3. Posiciones simétricas tras el primer movimiento

Se considera el grupo de las 8 simetrías del cuadrado (figura 4) para no multiplicar la búsqueda en configuraciones simétricas. Esto es, consideramos a dos estados como idénticos si uno representa alguna transformación de simetría del otro.

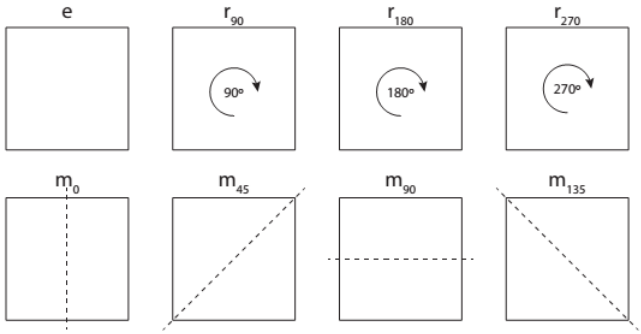

Figura 4. Simetrías del cuadrado: identidad y rotaciones (fila superior)
y reflexiones según ejes longitudinales, transversales y en diagonal (fila inferior)

Estados incorregibles

A medida que se avanza en el juego, ciertas jugadas pueden llevar a un estado del tablero *incorregible*, esto es, que dejan al tablero en una configuración a partir del cual es imposible ganar. Esto puede ocurrir en etapas relativamente tempranas del juego, cuando aún quedan muchas posibilidades por explorar (la figura 5 muestra un ejemplo con 16 piezas restantes). Se puede intentar de podar el árbol de búsqueda identificando estos estados incorregibles y (desde las heurísticas).

Este problema no es trivial, y no se halló una manera de identificar con precisión absoluta los estados incorregibles. Existe una manera de identificar *algunos* de estos estados. Para esto se define una asignación de valores v a las posiciones del tablero con la siguiente característica: si x, y, z son posiciones contiguas en el tablero (con y entre x y z, horizontalmente o verticalmente), entonces $v(x) + v(y) \geq v(z)$. Si definimos S como la suma de los valores ocupados por fichas, se cumple que tras cualquier movimiento válido, el valor de S no puede crecer. Así, si se encuentra un estado con un valor de S menor al de la configuración objetivo (una ficha solitaria en la posición central), se puede determinar que ese estado es incorregible y no explorar sus descendientes.

Se muestra una posible asignación de valores con estas características, que fue la utilizada en este trabajo, en la figura 4. Notemos que esta no es la única posible, y diferentes asignaciones resultarían en podas distintas, y además que pueden existir estados incorregibles que no resultan podados por una asignación en particular.

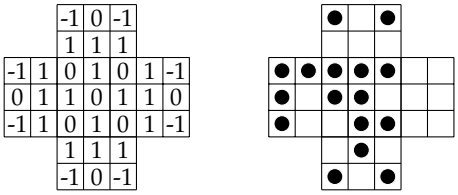

Figura 5. Una asignación de poda (izquierda) y ejemplo de un estado incorregible (derecha)

Heurísticas

Las heurísticas que se implementaron intentan penalizar distribuciones de piezas consideradas desventajosas, teniendo en cuenta de que existen configurariones "incorregibles" que se busca evitar.

Una de las estrategias es asignar valores a cada posición del tablero, y sumar los valores de las posiciones ocupadas. Esta asignación estática de valores se basa en asumir que hay posiciones que se quieren desocupar lo antes posible. La segunda estrategia consiste en considerar las posiciones relativas de las piezas para calcular el valor de la heurística. Se describen a continuación las heurísticas en detalle.

Cantidad de fichas restantes

La heurística devuelve simplemente $N-1$, donde N es la cantidad de piezas en el tablero en ese estado. Esta heurística es admisible y de hecho estima con exactitud el costo del camino restante (en los estados que son parte de un camino a la solución).

Fichas aisladas

Una ficha en la posición del tablero se considera aislada si no contiene fichas vecinas horizontal o verticalmente. En otras palabras, si la ficha esta en la posición (i, j), esta aislada si el tablero no tiene fichas en las posiciones $(i-1, j)$, $(i+1, j)$, $(i, j-1)$, $(i, j+1)$. La figura 6 muestra ejemplos de fichas aisladas.

Esta heurística busca minimizar la cantidad de piezas aisladas en el tablero, pues dejar piezas aisladas conduce más fácilmente a estados incorregibles. Así, si N es la cantidad de fichas en el tablero y M es la cantidad de fichas aisladas, esta heurística tiene un valor de $2N + M$. El término $2N$ tiene la función de tender a la búsqueda en profundidad, ya que disminuye a medida que se avanza hacia el objetivo. El factor multiplicativo de 2 es empírico. Esta heurística no es admisible.

Dificutad de remoción

En esta heurística se asigna un valor a cada posición del tablero según cuán favorable es mantener fichas en esa posición. Los valores se muestran en la figura 6. La idea es que las fichas posicionadas en las esquinas son más dificiles de remover, especialmente en los estados finales del juego. Las esquinas internas se penalizan un poco menos por estar más cerca del centro.

Dado un estado del tablero, si S es la suma de los valores de las posiciones ocupadas y N es la cantidad de piezas en el tablero, la heurística retorna $2N + S$. El término $2N$ aparece por las razones descritas en la sección III.a. Esta heurística no es admisible.

Distancia entre fichas

Si $P = \{p_1, p_2, \cdots, p_n\}$ es el conjunto de las piezas en el tablero y $\delta : P^2 \to \mathbb{N}$ indica la distancia Manhattan entre dos piezas, esta heurística retorna

$$\sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \delta(p_i, p_j)$$

es decir, la suma de las distancias Manhattan entre todos los pares de piezas del tablero.

El objetivo de esta heurística es similar a la de fichas aisladas (sección III.b), con un poco más de generalidad: la intención es que las piezas se encuentren siempre lo más cerca posible entre ellas en las etapas iniciales del juego (nuevamente, mientras menos piezas alejadas o aisladas, mejor), y que finalmente se vayan acercando a un único cluster. Esta heurística no es admisible.

Distancia al centro

En esta heurística se asigna un valor a cada posición del tablero según su distancia Manhattan a la posición central. El valor de cada posición puede consultarse en la figura 6. Así, dado un estado, si S es la suma de los valores de las posiciones ocupadas del tablero, esta heurística retorna $2S$.

El objetivo es que utilizando esta heurística se tienda a agrupar las piezas alrededor del centro del tablero, liberando primero las posiciones extremas, con el fin de no dejar piezas dificiles de capturar (lejos del centro) tras varios movimientos. El factor multiplicativo

2, determinado empíricamente, sirve para guiar la búsqueda en profundidad (similar al término $2N$ de las heurísticas ya descritas). Esta heurística no es admisible.

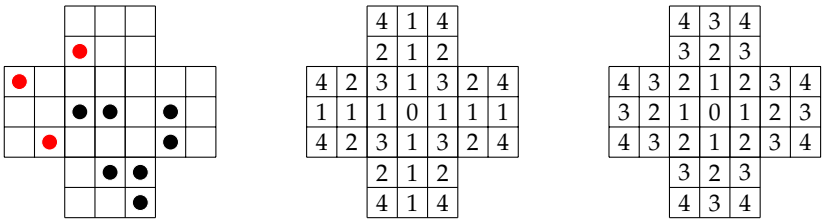

Figura 6. Fichas aisladas en rojo (izquierda), valor de cada posición según dificultad de remoción (centro) y distancia Manhattan al centro (derecha).

RESULTADOS

Resultados cuantitativos

Búsquedas

La tabla 1 compara los resultados para cada tipo de algoritmo implementado. En cuanto a las búsquedas desinformadas, únicamente la estrategia *DFS* logró alcanzar el objetivo, mientras que en las informadas ambas estrategias lo alcanzaron. Se muestran los resultados de la heurística *b*, dificultad de remoción.

Métrica \ Algoritmo	Iterativo	DFS	BFS	Uniform Cost	A*(c)	Greedy search(c)
Tiempo de procesamiento [ms]	-	79393	-	-	55	50
Cantidad de nodos expandidos	-	2910002	-	-	475	475
Cantidad de nodos frontera	-	116	-	-	166	166
Cantidad de nodos generados	-	2910118	-	-	641	641

Tabla 1. Comparación de los distintos algoritmos de búsqueda implementados.

En los casos de *BFS* y *Uniform Cost* no fue posible alcanzar el estado solución debido al consumo completo de la memoria del sistema, alcanzando hasta los 4 GB de almacenamiento. En ambos casos no se logró superar el nivel de profundidad 12, que corresponde a tener 13 posiciones libres en el tablero. En el caso del método *Iterativo* el limitante fue el tiempo, no pudiendo superar tampoco el nivel 12 de profundidad incluso luego de 10 horas de procesamiento.

La principal diferencia entre *DFS* y las búsquedas informadas se da tanto en el tiempo de procesamiento como en la cantidad de nodos expandidos, siendo la cantidad de estos hasta 6000 veces superior. Esto da a entender que una búsqueda informada es superior a una *DFS* respecto tanto al costo temporal como espacial, incluso a pesar de que la profundidad de la solución es siempre conocida para este problema en particular.

Heurísticas

La tabla 2 compara los resultados para cada heurística implementada. La letra asignada a cada heurística se corresponde con las de la sección 2.III. Todas logran alcanzar el objetivo en un tiempo razonable, siendo el máximo poco más de 18 segundos. En cuanto al almacenamiento requerido, el máximo corresponde a 827981 nodos generados lo que también es un resultado razonable para la cantidad de almacenamiento que manejan la mayoría de los sistemas hoy día. Considerando que en nuestra implementación del problema un nodo (incluyendo el estado del tablero) ocupa alrededor de 64 B, la cantidad máxima de almacenamiento requerida sería alrededor de 51 MB.

Los mejores resultados claramente corresponden a la heurística dificultad de remoción teniendo un costo notablemente menor tanto temporal como de almacenamiento. La figura 7 muestra la cantidad de nodos generados por nivel para cada heurística. Es notable que la cantidad de los mismos aumenta consistentemente a partir de cierto nivel, alcanza un máximo y luego decrece consistentemente sin presentar oscilaciones. Se puede apreciar así el *branching factor* variable del problema: en los primeros niveles hay pocas jugadas para realizar pero en los niveles medios, a partir del $15 \sim 17$, cuando el tablero se encuentra medio lleno la posibilidad de jugadas aumenta notablemente.

Por último, deja en descubierto el nivel que a cada heurística le resulta más dificultoso superar. Cuanto más alto sea este nivel, menor la cantidad de nodos generados pues al estar tan vacío el tablero, menor es la cantidad de jugadas posibles. Es así como la mejor heurística alcanza su máximo recién en el nivel 25 cuando hay tan solo 7 fichas restantes.

Métrica \ Algoritmo	A*(*a*)	A*(*b*)	A*(*c*)	A*(*d*)	A*(*e*)	G(*a*)	G(*b*)	G(*c*)	G(*d*)	G(*e*)
Tiempo de procesamiento [ms]	714	7309	57	3476	1625	18390	8185	50	6031	1587
Cantidad de nodos expandidos	20835	319106	475	111782	59064	827873	372811	475	111782	56674
Cantidad de nodos frontera	180	692	166	96049	49326	108	129	166	96049	47807
Cantidad de nodos generados	21015	319798	641	207831	108390	827981	372940	641	207831	104481

Tabla 2. Comparación de las distintas heurísticas implementadas.

Poda

La poda descrita en la sección II.c fue incorporada solamente en las heurísticas (*d*) y (*e*). Para la heurística (*d*) resultó en una disminución de un 49,90 % en las expansiones aunque solamente del 8,25 % en la cantidad de nodos generados. Para la heurística (*e*) se logra una disminucion del 42,22 % de nodos expandidos aunque un aumento de 7,3 % de nodos generados. Para las otras heurísticas la diferencia no era tan interesante (especialmente en tiempo de ejecución) y se decidió implementarlas sin la poda.

Discusión cualitativa

Búsqueda desinformada

Los algoritmos *BFS* e *Iterativo* no resultan de mucho interés para el análisis: sabemos que la solución se encontrará siempre en el nivel 31 de profundidad, luego se encontrará (eventualmente) una solución en ese nivel: en este sentido, el método *Iterativo* termina siendo equivalente a un *DFS* normal, pues solo habrá posibilidad de encontrar una solución en la iteración 31. La expansión en anchura de *BFS* tampoco aporta demasiado, pues no existen soluciones óptimas que este algoritmo alcance *primero*. Finalmente, el alto *branching factor* de este problema es altamente detrimental para estos métodos.

La búsqueda en profundidad es un poco más interesante: es claro que intentar avanzar lo más rápido posible hacia el nivel de profundidad 31 es positivo para encontrar una solución con prontitud. Los resultados con *DFS* fueron positivos en nuestra implementación, pero se observa que esto depende fuertemente de qué estados se eligen tempranamente para expandir, y esto puede ser producto del azar y no consistente entre implementaciones. En efecto, se corrió el algoritmo *DFS* con un tablero *ya empezado* (dos movimientos hechos), y se observó que en ese caso la solución no se alcanza en un tiempo razonable. En contraposición, el comportamiento de las búsquedas informadas en este caso es consistente (se expanden menos estados comenzando desde este tablero).

Búsqueda informada

Resulta interesante analizar los caminos a la solución obtenidos por cada una de las heurísticas. En el Apéndice pueden encontrarse los caminos a la solución obtenidos por las distintas heurísticas.

Se observa que para la heurística que minimiza las piezas aisladas, se logra resolver el juego prácticamente sin aislar ninguna pieza. Para la heurística que pretende minimizar la distancia entre las piezas se nota una agrupación de las mismas en un único cluster hacia la izquierda del tablero, que se contrapone con la distribución que logra la heurística *distancia al centro*, donde las piezas están más dispersas. Finalmente, para la heurística más exitosa, la asignación ad-hoc de ”dificultad de remoción”, es notable la tendencia a liberar las esquinas, y no se observa la *clusterización* que caracteriza a las basadas en distancia.

El gráfico de la figura 7 presenta algo interesante: las heurísticas que consideramos ”mejores”, es decir, las que generan menos nodos, también presentan su máximo en niveles más profundos: esto es indicativo de que esas heurísticas son más apropiadas, pues pueden avanzar más sin recurrir a backtracking.

Otra cosa a notar es que la búsqueda utilizando algoritmos *greedy* condujo a las mismas soluciones, lo cual no es sorprendente, considerando que todos los pasos tienen costo 1.

CONCLUSIONES

Dados los resultados distintos obtenidos se puede hacer una valoración positiva del analisis preliminar: las heurísticas admisibles no resultan provechosas en este problema, y las no admisibles lograron buenos resultados. El uso de simetrías fue positivo en el uso de memoria y la poda, si bien las expectativas eran mas altas, hizo efecto en algunos casos.

Se logró llegar al objetivo con todas las heurísticas propuestas, pero cada una resuelve el problema de distinta manera. Cada una se debe evaluar segun qué se considera prioritario a la hora de hallar la solución: aquella forma que lleve menos tiempo, aquella que expanda una menor cantidad de nodos o tal vez la que produzca menos estados.

APÉNDICE 1. ALGORITMOS Y FIGURAS

Algoritmo 1. Graph Search

```
1:  function GRAPH SEARCH(problem)
2:      frontier ← new Queue
3:      explored ← new Set
4:      root ← new Node(problem.getInitialState())
5:      frontier.add(root)
6:      while not frontier.empty() do
7:          node ← frontier.remove()
8:          state ← node.state
9:          if problem.isResolved(state) then
10:             return node
11:         explored.add(state)
12:         for each rule in problem.getRules(state) do          ▷ Expandir nodo
13:             childState ← rule.applyToState(state)
14:             childNode ← new Node(childState, node, rule)
15:             if not explored.contains(childState) then
16:                 frontier.add(childNode)
17:     return failure
```

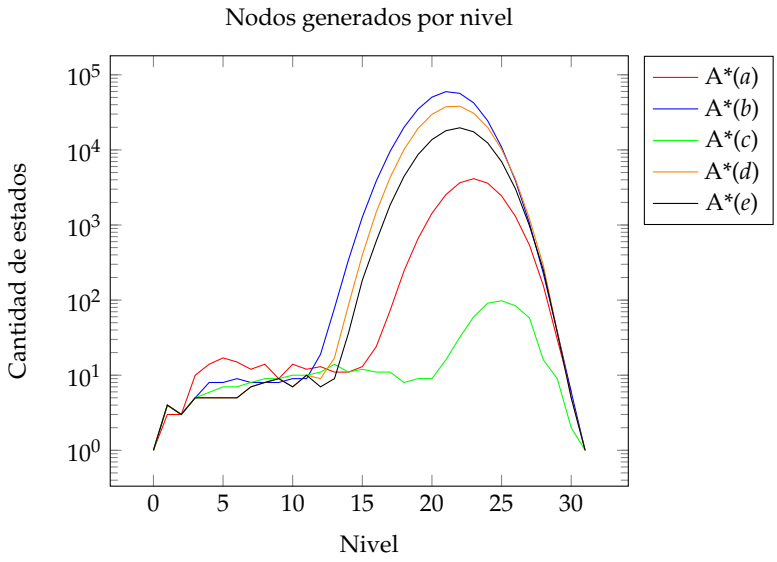

Figura 7. Nodos generados por nivel (nótese la escala logarítmica).

APÉNDICE 2. SOLUCIONES OBTENIDAS (A*)

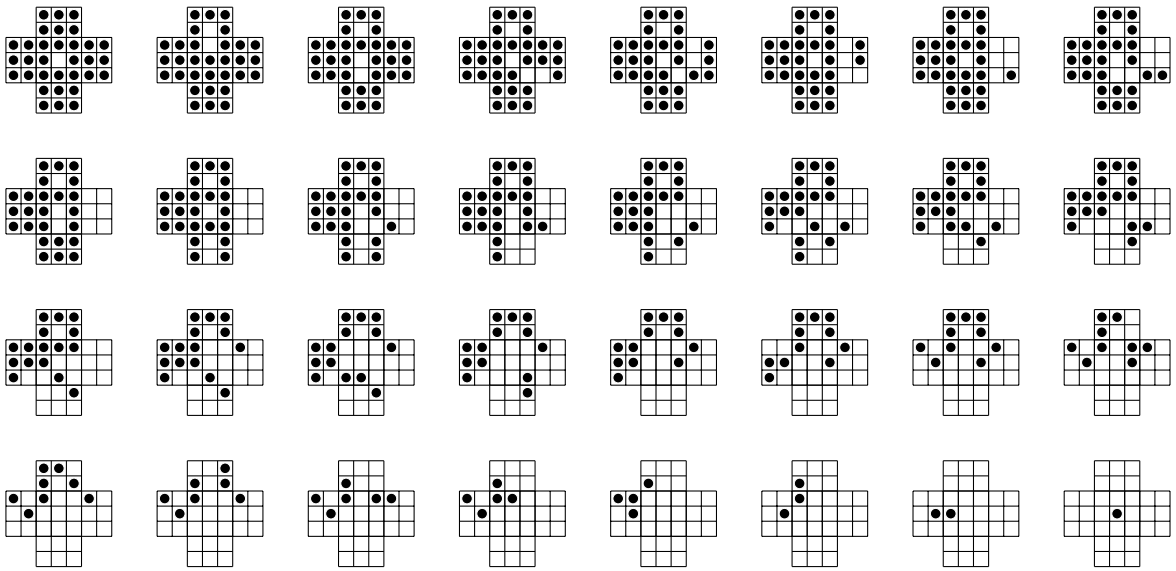

Figura 8. Heurística (a): Cantidad de fichas restantes

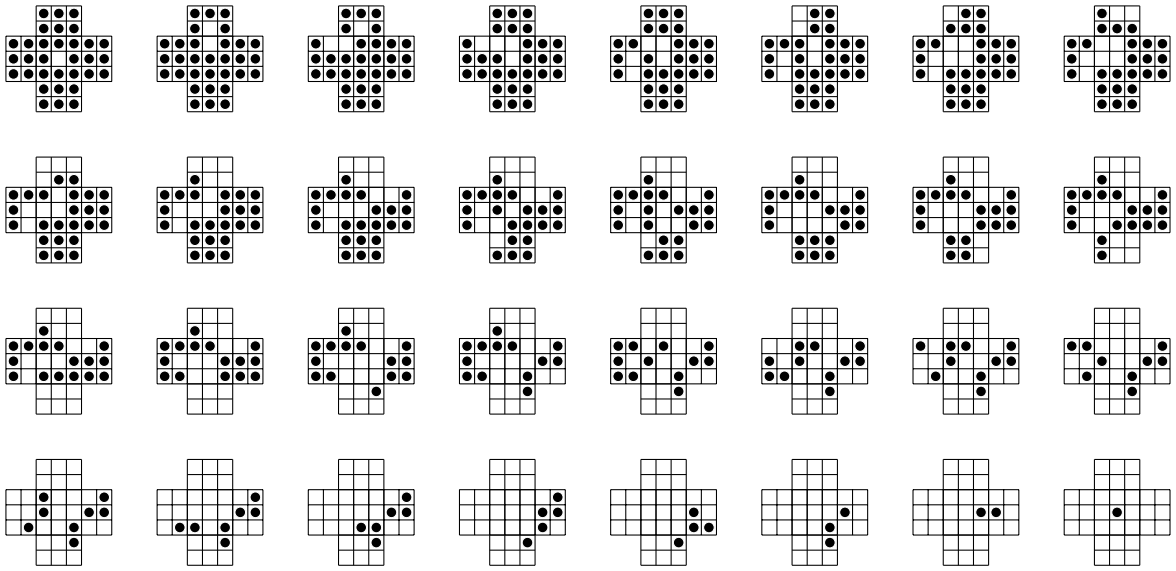

Figura 9. Heurística (b): Fichas aisladas

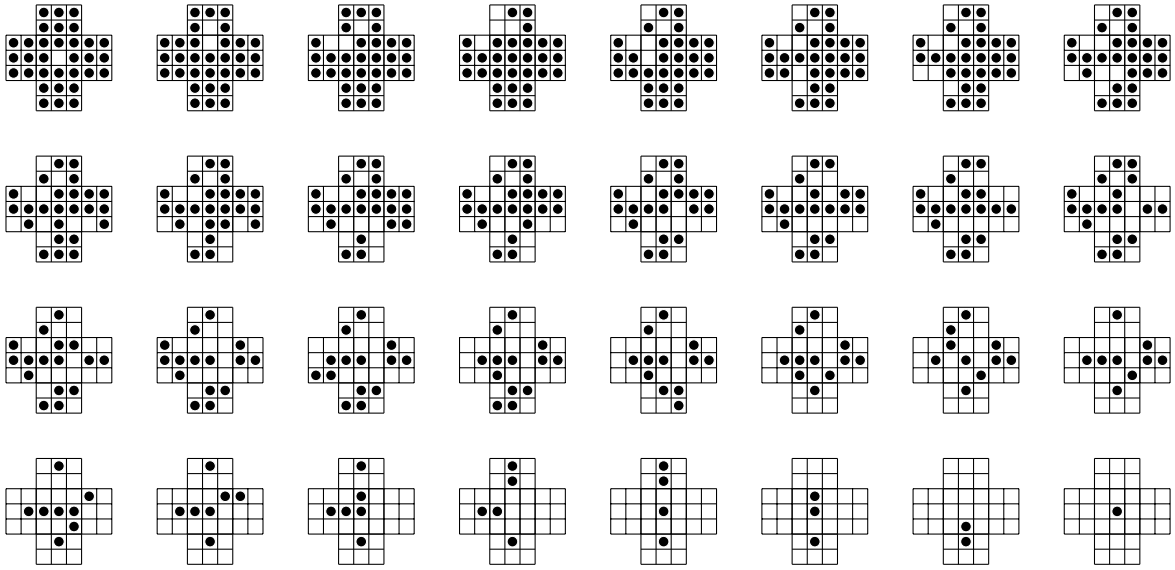

Figura 10. Heurística (c): Dificultad de remoción

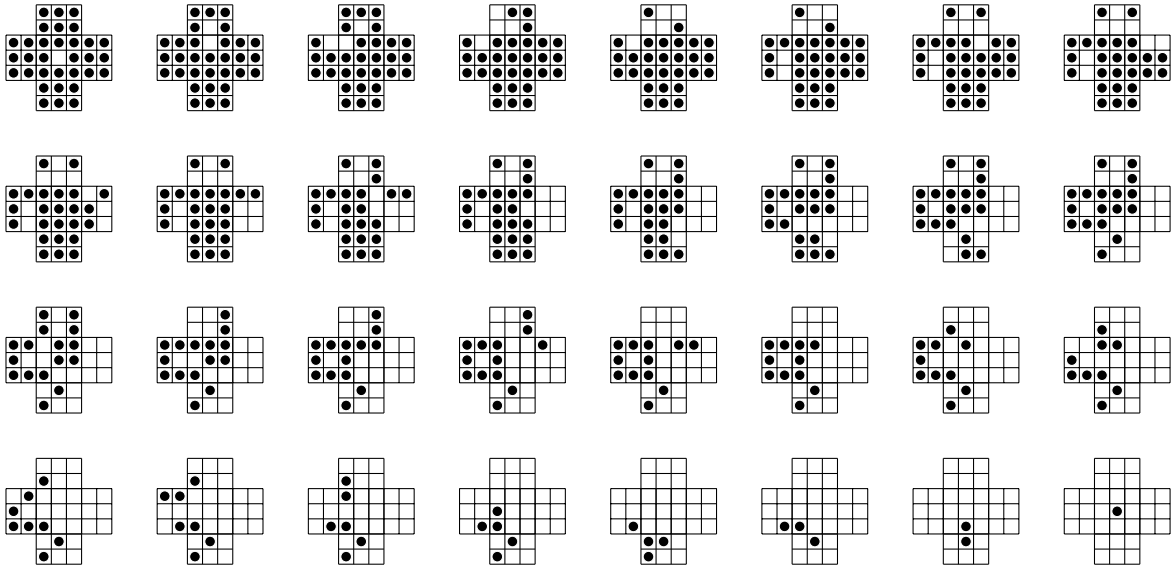

Figura 11. Heurística (d): Distancia entre fichas

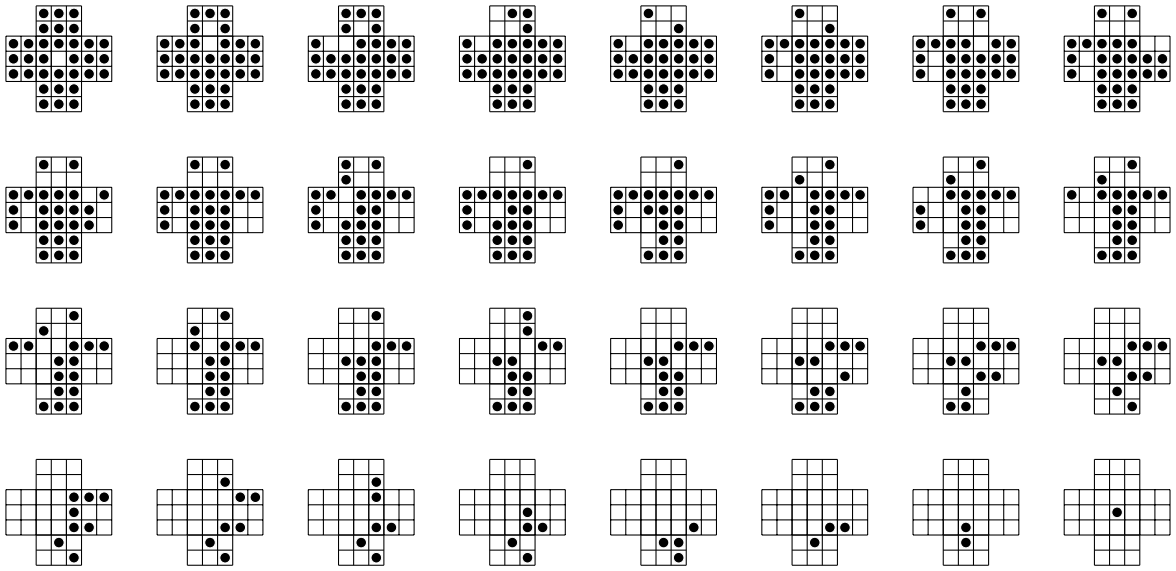

Figura 12. Heurística (e): Distancia al centro