Inteligencia Artificial

Estado del Arte: Problema KS

Rolando J. Casanueva Quezada

December 14, 2018

Evaluación

Mejoras 1ra Entrega (10 %):	_____
Código Fuente (10 %):	_____
Representación (15 %):	_____
Descripción del algoritmo (20 %):	_____
Experimentos (10 %):	_____
Resultados (10 %):	_____
Conclusiones (20 %):	_____
Bibliografía (5 %):	_____
Nota Final (100):	_____

Abstract

Este informe, presenta la implemtación de un algoritmo evolutivo con cruzamiento en un punto y mutación swap para el problema del Killer Sudoku. Se señalan importantes observaciones sobre el problema como separar las celdas de las restricciones, para facilitar la implementación o notar como un tamaño de población entre 30 y 50 es más eficiente para la realización de experimentos.

Keywords: *Killer Sudoku, CSP, NP-Hard, CSP Binario*

1 Introducción

En este informe se aborda el problema llamado *Killer Sudoku*, el cual es una particularización del problema *Sudoku*. Este mismo es un juego donde se debe rellenar una grilla con números del 1 al 9 en 9 secciones iguales, en su versión tradicional, manteniendo una consistencia de atomicidad de los números en filas, columnas y secciones, además de cumplir con una suma específica en subsecciones demarcadas. La metodología que se utiliza para entender el problema apunta a la investigación de modelos y estructuras que permiten describir y resolver el juego.

Ahora bien, este informe cuenta con una descripción a detalle del problema y sus variantes, una alución a métodos y algoritmos de resolución utilizados en la actualidad, un modelo matemático que describe el problema con sus variables y restricciones. A modo de contraste con el informe anterior, este trabajo cuenta con un cambio en el modelo matemático para ajustarse al método de resolución.

Luego, se presenta una descripción del algoritmo evolutivo utilizado junto con los experimentos y resultados obtenidos, para finalizar con unas breves conclusiones.

2 Definición del Problema

Para poder entender el problema de *Killer Sudoku (KS)*, hay que entender su origen y dado que el **KS** es una derivación del Sudoku tradicional, es imperativo conocer sobre este juego.

El **Sudoku** hace su primera aparicion en Estados Unidos bajo el nombre de *'Number Place'* en 1979 [6]. El juego fue diseñado por Howard Garns [5], un arquitecto que al aproximarse al retiro decidió dedicarse a la creación de puzzles. Con el cambio de década, el juego tomó popularidad en Japón siendo renombrado a *'Suji wa dokushin ni Kagiru'* que se traduce como *'Los dígitos deben permanecer solitarios'*. [13] Eventualmente tomó el nombre de *'Single Number'*, o *'Sudoku'* en japonés.

El Sudoku tradicional se compone de un tablero con 81 casillas separadas en 9 secciones de 9 casillas cada una, como se puede apreciar en la figura 1. En esta versión puede notarse que tiene un total de 5472730538 tableros completamente diferentes [10].

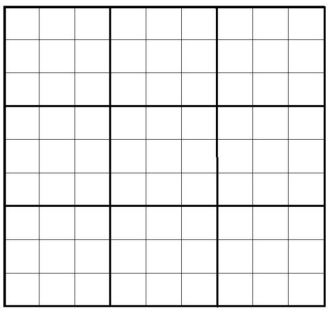

Figure 1: Tablero de Sudoku vacio.
Fuente: Elaboración propia

Para entender la dinámica del juego, usando la figura 3, se explican las reglas, digase restricciones, que deben cumplirse. No puede repetirse ningún número del 1 al 9 en una:

(A) Columna.

(B) Fila.

(C) Sección.

Asi, cada casilla equivale al dígito en la casilla y tiene un dominio de todos los dígitos excepto el 0. Mientras que cada uImportante a considerar es que un Sudoku cuenta con a lo menos 17 pistas [9], ya que de poseer menos pistas, la solución al puzzle no sería única.
na de las partes (A), (B) o (C) deben satisfacerse como restricciones del problema.

Con respecto a las variaciones existentes se hace referencia al articulo de Ed Pegg Jr [7], donde menciona una gran cantidad de variantes. Algunas de ellas son, "*Diagonal Sudoku*", que es igual a un Sudoku tradicional, solo que se agrega la variante donde las diagonales principales del tablero también deben ser distintos. "*Even-Odd Sudoku*"consiste en un Sudoku tradicional, donde las casillas del tablero están marcadas con uno de dos colores, significando que para un color, la casilla, solo puede tener un valor par mientras que el otro color, un valor impar. "*Domino Sudoku*", consta en lograr un el objetivo de un Sudoku tradicional pero con piezas del juego dominó. Generalmente, todas las variantes se basan en el juego tradicional y añaden una o dos restricciones sobre el juego. Pero dentro de ellas, este informe se enfoca en el Killer Sudoku, el cual agrega una nueva agrupación de casillas llamadas subsecciones. Cada subsección tiene un valor total, véase las esquinas de las casillas en la figura 2, el cual deberá ser la suma de las casilla dentro de la subsección. Agregando así las restricciones:

(D) No puedo repetirse ningún número del 1 al 9 en una subsección.

(E) La suma de las casillas en una subsección debe ser igual al valor de la subsección.

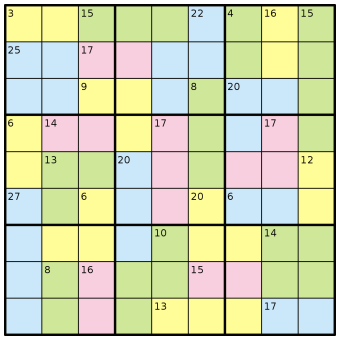

Figure 2: Tablero de Killer Sudoku con subsecciones resaltadas.
Fuente: Wikipedia

3 Estado del Arte

Como fue mencionado anteriormente, el juego se basa en satisfacer 5 restricciones y encontrar una solución única. Por ende, el Killer Sudoku se puede clasificar como un **CSP** [8].Como tal, Helmut [11] hace una investigación sobre el Sudoku como CSP, donde compara diferentes técnicas de filtrado aplicadas a distintas instancias del problema para luego comparar el porcentaje de problemas resueltos sin haber usado búsqueda. Su técnica modela el problema con una matriz $n^2 * n^2$ con un dominio de variable $D_x = 1, ..., n^2$. También esto requiere de $3 * n^2$

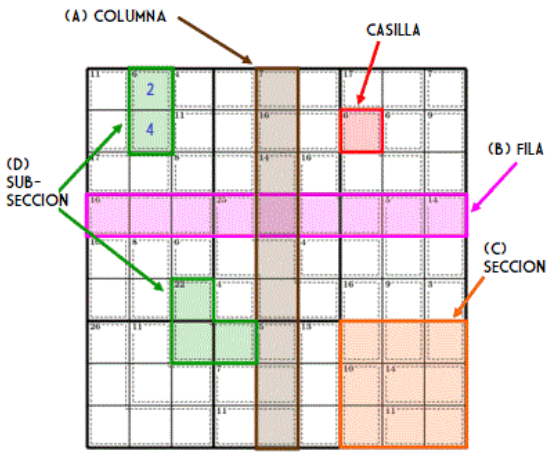

Figure 3: Tablero de Killer Sudoku y sus componentes.
Fuente: Modificación a la imagen de Killer Sudoku Online [2]

restricciones para cada fila, columna y sección de $n * n$.

En la investigación Felgenhauer [4] sobre la cantidad de escenarios, tras modelar el problema como bloques de 3 números, donde estos bloques son permutaciones de 3 valores distintos entre el margen de $1 - 9$, producciendo que para una sección existan $3 * 9! * 8! * 7!$ posibilidades, se filtran su posibilidades mediante patrones o valores duplicados y así se llega a un valor $N = 6.671 * 10^{21}$ de Sudokus válidos. Valor que por lo demás señala la inmensa dificultad que puede llegar a tener uno de estos puzzles basado en la cantidad de pistas como se mencionó anteriormente en la publicación de Gary [9].

En el trabajo de Bhaskar [1] se presenta una representación lineal entera binaría dando un punto de vista interesante, ya que propone aumentar el número de variables para atomizar cada casilla en n variables asociadas. Considerando un puzzle de esta forma, sin ninguna técnica de consistencia aplicada, se ve un espacio de búsqueda de 2^{729}.

Por otra parte Eppstein [3] muestra como una heurística permite mejorar el rendimiento de un *solver* aún cuando el Sudoku es un problema NP-Difícil. Con su investigación lográ probar que aplicando *Nishio Single-Digit* subproblemas, el Sudoku puede ser resuelto en tiempo $o(2^n)$. Este modelo de resolución básicamente aplica un modelo binario como el descrito previamente.

Ahora entrando a los estudios sobre Killer Sudoku, el 2013 en la conferencia *Complex Adaptive Systems*, se presenta un estudio sobre una técnica llamada **EMA**, de *Exponential Moving Average*, que es una mutación a un **EA**, dígase algoritmo evolutivo. En este trabajo, se presenta una variante interesante donde una casilla se ve representada no con 2 sino que 3 sub-índices, x_{ijk}, donde i representa un cromosoma, es decir una cadena de caracteres, j representa el j^{th} elemento en el cromosoma mientras que k fórmula un índice. Tras comparar la cantidad promedio de evaluaciones realizadas, se puede notar que esta formulación permite retrasar la cantidad de evaluaciones totales realizadas hasta casi 40 casillas en blanco con cerca de 100 evaluaciones, sin embargo no más lejos a las 44 casillas en blanco sube en picada superando las 9000 evaluaciones, para luego estancarse en el límite propuesto en el algoritmo con los demás métodos. El EMA puede verse como un progresión geométrica $h_t = h_{t-1} * r + f_t * (1 - r)$. con f_t siendo una función de evaluación definida para tanto casillas, como para columnas, filas y secciones como tambien

para el cromosoma.

Por otro lado, en Holanda, Wang [12] muestra un estudio con el intento de aplicar un SAT solver, identifiquese SAT como Problema de satisfacibilidad booleana. Para este método, también se utiliza una variable con 3 sub-índices, sin embargo sus significados son distintos, dado que para este caso, se toma (i, j) como la fila i y la columna j, mientras que k referencia el dígito posible y el valor de la varíable toma valor *True* si la casilla corresponde a la etiqueta k. Luego presenta resultados sobre distintos valores de n para los KS, donde desde los KS de $2x2$ hasta los de $9x9$, aquel que más demora en promedio resulta ser interesantemente el de $6x6$.

4 Modelo Matemático

Luego de la entrega del algoritmo, el modelo matemático sufrio ciertos cambios sobre el modelo anterior, para beneficiar la representación. Así mismo, una explicación a detalle del modelo se entrega en la siguiente sección.

Principalmente cambia la representación del Killer Sudoku, separandose el valor de una casilla con la estructura del tablero.

$$x_{ij} = Valor\ de\ la\ casilla\ (i, j) \tag{1}$$

Con los valores de $i, j = \{1, ..., n^2\}$, y $x = \{1, ..., n^2\}$ dando así un espacio de búsqueda bruto sin filtros de $EB_x = n^{2n^4}$, que comparado con el espacio de búsqueda anterior $EB_{x_{anterior}} = 2^{n^6}$ resulta menor para el caso tradicional de $n = 3$.

Para controlar la estructura del tablero, se tiene un parametro que lleva tanto el tamaño total de la subsección, como otro parametro que señala las coordenadas de la subsección.

$$KS_i = Suma\ de\ la\ subsección\ i \tag{2}$$

$$KC_i = Conjunto\ de\ coordenadas\ de\ la\ subsección\ i \tag{3}$$

$$KN_i = Conjunto\ de\ coordenadas\ de\ la\ sección\ i \tag{4}$$

Donde KS puede tener un valor entero y el índice puede ser $i = \{C_1, ..., C_m\}$, tal que m sea la cantidad de subsecciones del problema. Por otra parte, si se considera un conjunto $C = \{... (x, y) ...\}$, que posea todos las coordenadas del tablero posibles, entonces $KC \subseteq C$ y el índice cumple la misma condición que para el caso anterior. Por último, KN cumple la misma funcion que KC pero para las secciones del tablero, donde el índice toma valores como $i = \{1, ..., n^2\}$.

Ahora bien, las restricciones se modifican con respecto al informe anterior, para adaptarse a esta nueva forma de modelar el problema, separando o delegando restricciones a partes distintas.

Por cada fila i los valores de la columna j no se repiten.

$$\bigcup_{j=1}^{n^2} x_{ij} = \{1, ..., n^2\} \quad \forall i \tag{5}$$

Por cada columna j los valores de la fila i no se repiten.

$$\bigcup_{i=1}^{n^2} x_{ij} = \{1, ..., n^2\} \quad \forall j \tag{6}$$

En cada sección k los valores de todas las filas y columnas pertenecientes a la sección no se repiten.

$$\bigcup_{(i,j) \in KN_k} x_{ij} = \{1, ..., n^2\} \quad \forall k \tag{7}$$

En cada subsección k los valores de todas las filas y columnas pertenecientes a la subsección suman el valor de la misma.

$$\sum_{(i,j) \in KC_k} x_{ij} = KS_k \quad \forall k \tag{8}$$

5 Representación

Como se mencionó anteriormente, el Killer Sudoku (figura 4), es tratado en este trabajo como dos juegos distintos, manteniendo separadas las representaciones de el tablero y los números.

Figure 4: Tablero de Killer Sudoku completado.
Fuente: Modificación a la imagen de Killer Sudoku Online [2]

El tablero, figura 5.a, se mantiene como un conjunto de coordenadas para poder guardar registro de las subsecciones y en que posición se encuentran las casillas de la misma. Por otra parte, los valores de las casillas, figura 5.b, se representa con un *vector* o arreglo de números unidimensional, que básicamente equivale a todas las filas leidas una a una de izquierda a derecha y de arriba hacia abajo.

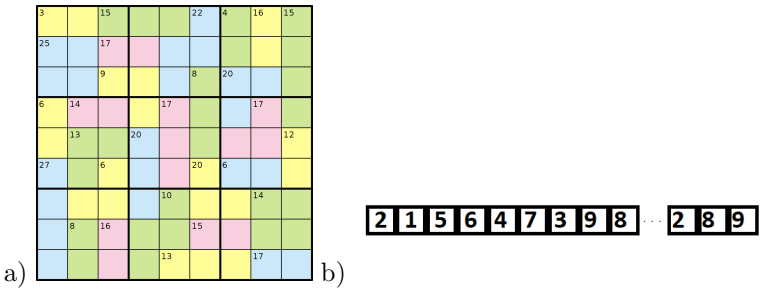

Figure 5: a) Tablero de Killer Sudoku y b) lista de números del juego.
Fuente: a) Wikipedia y b) elaboracion propia

De aquí en adelante, todo lo concerniente a esta representación de los valores, se hará referencia como **cromozoma**. Un cromozoma se considera como un tablero lleno de número, mas no

necesariamente un tablero correcto, en otras palabras, esta representación permite tener soluciones infactibles.

Como este problema es un CSP se formula una función de evaluación que considera la suma de restricciones insatisfechas, permitiendo, de esta forma, saber que una evaluación equivalente a 0 implica que el cromozoma es una solución factible.

Los movimientos que permiten la explotación como la exploración, fueron pensado en los movimientos básicos de un algoritmo evolutivo, donde el cruzamiento permite intensificar las caracteristicas de los cromozomas, mientras que la mutación permite diversificar. Es así como el cruzamiento se realiza en un punto aleatoreo realizando un corte e intercambiando las primeras secciónes de dos cromozomas padres, generando dos cromozomas nuevos. Mientras que la mutación es un movimiento *swap* entre dos casillas no entregadas como pista inicial. Este movimiento es un cambio válido al modelo, debido a que todos los cromozomas son inicializados de manera tal que todas las filas no tienen números repetidos. Con esta manera de inicializar los cromozomas, queda asegurado que existe almenos una combinación de movimientos que permitirá llegar a la solución.

Como es un algoritmo evolutivo el que se está planteando, es importante señalar que el tamaño de la población y las probabilidades de cruzamiento y mutación se mantendrán como parametros constantes. Por otra aprte el proceso de selección se basa en la **ruleta**, donde aquel individuo con mayor *fitness* tiene mayor probabilidad de ser elegido.

$$P_x = \frac{F(x)}{\sum_{y \in POB} F(y)} \quad \forall x \tag{9}$$

Sin embargo, como la representación elegida implica que un *fitness* de menor valor significa una cromozoma de mayor calidad, hay que hacer un reajuste a la manera de asignar la probabilidad, de esta forma luego de calcular P_x se invierte el valor obtenido de la forma $P'_x = 1 - P_x$, para luego recalcular la probabilidad.

$$P_x = \frac{P'_x}{\sum_{y \in POB} P'_y} \quad \forall x \tag{10}$$

6 Descripción del algoritmo

Lo primero a explicar son las estructuras creadas para el problema.

- cell
  Representa una celda del tablero, la cual posee un atributo entero para guardar el valor de la casilla y un atributo booleando para guardar si la casilla fue entregada como pista o si inicialmente estaba vacia. Esto permitirá saber si puede ser mutada la casilla o no.

- dna
  Representa un cromozoma, el cual contiene un vector de celdas para guardar todas las celdas del tablero, un valor entero que permite almacenar el *fitness* del cromozoma y finalmente un valor flotante para guardar la probabilidad que luego se utilizará en el proceso de selección.

- population
  Representa a la población del algoritmo evolutivo, la cual contiene un vector de cromozomas para guardar a todos los individuos y un atributo del tipo dna para almacenar al mejor individuo de la población.

- coord
  Representa una coordenada cartesiana, la cual tiene dos valores enteros para referencias los ejes carteseanos de una celda. Esta estructura permite almacenar una referencia a la celda que luego debe ser traducida a un valor lineal. Particularmente esta estructura fue creada para no perder de vista la estructura del tablero al cambiar la representación a un arreglo.

- group
  Representa una subsección, la cual posee un vector de coordenadas para almacenar todas las casillas pertenecientes a la subsección y un valor entero para guardar la suma de la subsección.

- killer
  Representa el tablero del killer sudoku perse, es decir, un vector de todas las subsecciones del problema.

- dom
  Esta es una estructura auxiliar creada para facilitar la implementación de ciertas partes del algoritmo y consiste en un vector de enteros.

Las constantes globales definidas son $PROB_c = 0.33$, $PROB_m = 0.33$ y $PROBLEM_{SIZE} = 3$. El tamaño del problema se define como 3 básicamente porque los ejemplos otorgados para el problema eran de un tablero de 9x9 es decir $n^2 = 9 \rightarrow n = 3$, mientras que las probabilidades se definieron en base a un criterio arbitrario de pensar que 1 de cada 3 individuos sufriría un cambio y/o se aparearía. Otro valor global como MAX_{gen}, que controla la cantidad máxima de generaciones permitidas en caso de no encontrar una solución en el camino, tras cierta cantidad de experimentación y análisis, se decidió mantener como un valor muy alto, para darle tiempo al algortimo de encontrar la solución, usualmente seteado en $MAX_{gen} = 10000000$. Por último se encuentra TAM_{pob} que regula el tamaño de la población y es claramente el valor global que ha de ser modificado para los experimentos posteriores.

Funciones auxiliares importantes creadas son las funciones de indexación, que permiten traducir un valor carteseano en un valor entero, haciendo el símil de una casilla en el tablero a su referente en el cromozoma.

Habiendo explicado lo anterior podemos adentrarnos un poco más en el algoritmo en sí. Partiendo por la lectura de un archivo base. Teniendo esta lectura puede originarse el cromozoma base para crear la población, donde éste es un arreglo con valores 0 donde la casilla es vacia, además es en este proceso que se genera la estructura **killer** para mantener el registro de las subsecciones. Tras haber realizado este proceso, se inicializa la población creando individuos de manera estocastica, aunque manteniendo la restricción de que cada fila no posea número repetidos. Al finalizar la creación de cada cromozoma se les otorga un valor de *fitness* que es calculado a través de una función auxiliar *findDuplicates*. La evaluación de la solución se divide en 4 funciones: row_fitness, col_fitness, nonet_fitness y groups_fitness. Las primeras dos funciones, evaluación de filas y evaluación de columnas, son muy sencillas de calcular debido a la función de indexación creada, permitiendo iterar por filas e iterar por columnas revisando la cantidad de duplicados y así sumarlo al total. La tercera función, evaluación de secciones, sigue el mismo patron anterior, sin embargo se utiliza una indexación distinta para que la iteración no sea lineal sino que itere sobre las secciones. Por último, utilizando la estructura **killer** se hace la suma de las coordenadas y se compara con el valor de la subsección, donde el no ser iguales implica una restricción insatisfecha más en el total de *fitness* previamente calculado.

Con esto los pasos iniciales están concluidos y llega la hora de realizar iteraciones, o bien, generaciones. Cada generación sigue un mismo patrón que es concordante con los algoritmos evolutivos vistos en clases, con un proceso de **selección**, seguido por un proceso de **cruzamiento**, luego un proceso de **mutación**, para finalizar con un proceso de **elitismo**.

El proceso de selección parte con el análisis de la población donde se le asigna a cada individuo la probabilidad en base a su evaluación, y gracias a que las estructuras de vectores son ordenadas, cada probabilidad asignada es equivalente al valor computado para su evaluación más la suma de las probabilidades entregadas previamente, de esta manera se puede generar la "*ruleta*". Una vez la ruleta es creada, de manera estocástica, se seleccionan individuos hasta tener seleccionados TAM_{pob} individuos. Esto generará una alta presión de selección en la población permitiendo que un individuo de buena evaluación sea escogido una mayor cantidad de veces.

El proceso de cruzamiento es muy simple, tomando dos individuos de la población seleccionada se procede, mediante la probabilidad de cruzamiento, a integrar en la nueva generación a los mismo individuos en caso de no cruzarse, o bien, los hijos en caso contrario. El cruzamiento, en principio, sufría de dos dificultades, los cuales eran que el movimiento de las casillas desordenaría las reglar de las subsecciones y que las pistas podrían terminar desordenandose también. Sin embargo, luego de un breve análisis, el segundo problema no hubo necesidad de solucionarlo, debido a que, como todos los individuos estan creados en base a un cromozoma padre con las pistas otorgadas, toda la población tendrá en las mismas casillas las pistas, por lo que el cruzamiento no afectará este hecho. Por otra parte, el primer problema mencionado, se soluciona gracias a que las reglas de las subsecciones se mantienen separadas del arreglo de valores, y se basan en la posición más alla del valor de la casilla.

El proceso de mutación inicialmente se había pensado como una especie de *bitflip* donde cada casilla podría cambiar su valor por uno nuevo dentro del conjunto con n^2 valores sin contar el valor que ya posee, sin embargo este tipo de mutación resulta más destructivo que reparador, debido a que la probabilidad de tener un tablero con los números indicados y en el orden indicado es muy baja. Ahora bien, como se tomó lo decisión de inicializar el tablero con filas sin valores repetidos, el tipo de mutación más apropiado es el *swap*. La única dificultad a superar para este movimiento es cambiar un el valor de una celda preseteada como pista, pero gracias a la estructura creada para las celdas, que permite almacenar si es una pista o no, se puede evitar la mutación en dicha celda. Para aplicar la mutación, se toma la población obtenida por el cruzamiento y en base a la probabilidad de mutación se realiza el cambio de casillas donde ambas casillas son seleccionadas estocasticamente dentro de las casillas que no hayan sido preseteadas.

Finalmente, el proceso de elitismo se basa en tomar la población generada luego de la mutación y al comparar las evaluaciones de todos los individuos de la población, se compara el mejor individuo de la generación anterior con el mejor de la nueva generación, en caso de que el cromozoma antiguo sea de peor calidad, la nueva generación se mantiene igual y el mejor individuo es almacenado. Por otra parte, si el mejor individuo de la generación anterior resulta ser mejor, se escoge un cromozoma aleatoreo de la nueva generación y es reemplazado por el cromozoma antiguo.

7 Experimentos

Todos los experimentos realizados fueron basados en los ejemplos entregados por los ayudantes.

En un principio se varió el tamaño de la población y la cantidad de generaciones para medir el tiempo de su ejecución y comparar estos resultados. Mas no muchos experimentos fueron necesarios para darse cuenta que la medida de tiempo era irrelevante debido a la naturaleza estocastica de la técnica, lo que no permite apreciar realmente si estos parametros variados mejoraban el rendimiento del algoritmo. Ahora bien, dados los conocimientos que se tienen gracias a las clases, podemos saber teóricamente que una población de menor tamaño convergerá a un resultado o un patrón, con mayor rapidez que una población más grande.

Fue tras estos breves experimentos que se llega a la conclusión de definir parametros como la cantidad de generaciones máxima en un valor muy alto, debido a que esto permitirá al algoritmo tener tiempo de realizar los movimientos necesarios para llegar a una solución. Por otra parte, el tamaño de la población se mantuvo variante a lo largo de los experimentos.

Si bien se utilizaron prácticamente todos los ejemplos entregados por los ayudantes y algunos ejemplos reales de internet, la instancia que fue medida y guardada con sus resultados fue el ejemplo **extreme_filled**.

Otro parametro que se mantuvo estático fue la probabilidad de cruzamiento, la cual se mantuvo en 0.33 durante todos los ejemplos. Esto se debe a que aumentar su valor solo genera un convergencia ciertos patrones dentro de la población, pero no logra mejorar la calidad de la población una vez los valores ya sean muy parecidos. En cambio, la probabilidad de mutación, fue variada en algunas instancias para verificar la hipotesis de que la mutación es el movimiento más importante para este problema, donde en un lugar de convergencia, este movimiento permite escapar y buscar nuevos ordenes en los valores del cromozoma.

8 Resultados

Dentro de lo básico que puede mencionarse, es que se logró llegar a una solución factible que para lo conocido en el mundo del Killer Sudoku es la solución única al problema.

Como se menciona en los experimentos, se mantuvo registro por sobre una sola instancia debido al tema temporal que implica realizar estos experimentos.

Para todos los resultados obtenidos se puede apreciar como las primeras generaciones rapidamente convergen a una evaluación pequeña, sin embargo de manera aleatorea, las mutaciones logran mejorar de a poco esta convergencia hasta llegar a un resultado.

En los siguientes gráficos se podrá apreciar en el eje X (horizontal), la iteración o generación mientras que el eje Y (vertical), la evaluación o fitness obtenida por el mejor individuo de aquella generación.

La figura 6 muestra 6 instancias del algoritmo evolutivo aplicado al ejemplo extreme_filled, cabe mencionar que se hicieron más experimentos con mayor población, sin embargo la cantidad de iteraciones y el tiempo consumido debido a que a medida que mayor es la población más tiempo demora el algoritmo por generación no vale la pena mostrarlo en oposición a lo que se puede sacar en conclusión con lo mostrado a continuación.

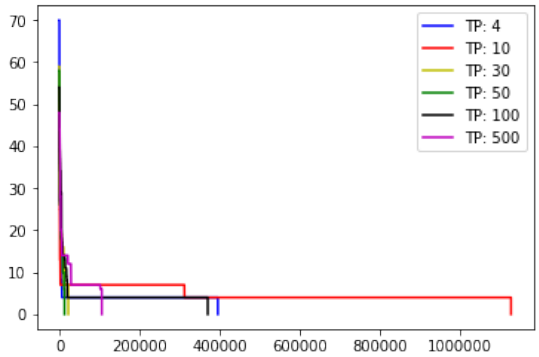

Figure 6: Comparación de AE para distintas poblaciones en el ejemplo 'extreme_filled' .
Fuente: Elaboración propia

Ahora bien, uno podría pensar que las poblaciones pequeñas requieren de mayor cantidad de generaciones para encontrar el máximo, sin embargo esto solo es un supuesto, debido a que la naturaleza estocástica del algoritmo podría permitir que cualquiera sea la cantidad de población la solución se encuentre en pocas como en muchas generaciones. Sin embargo, es cierto que una población pequeña tendrá menor probabilidad de obtener la mutación particular deseada, porque tiene menos individuos a los que mutar, pero este problema se traspasa también en un sentido distinto a poblaciónes de mayor cantidad debido a que aún demorando menos generaciones los tiempos por generación son mucho más extensos.

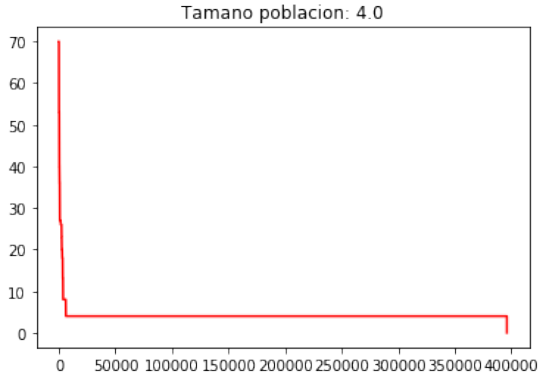

Figure 7: AE en el ejemplo 'extreme_filled' con población 4 .
Fuente: Elaboración propia

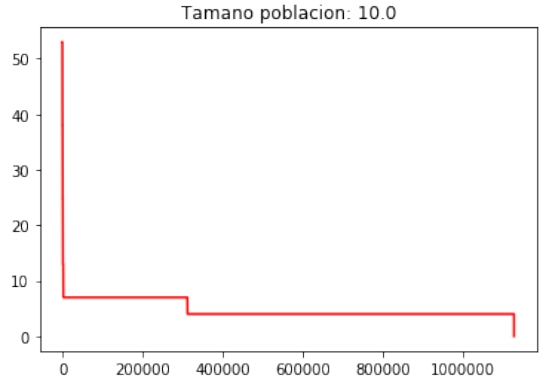

Figure 8: AE en el ejemplo 'extreme_filled' con población 10 .
Fuente: Elaboración propia

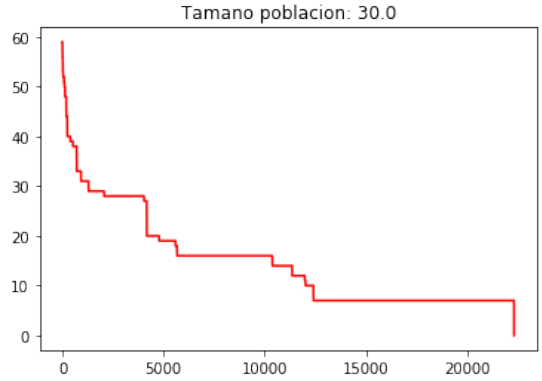

Figure 9: AE en el ejemplo 'extreme_filled' con población 30 .
Fuente: Elaboración propia

.

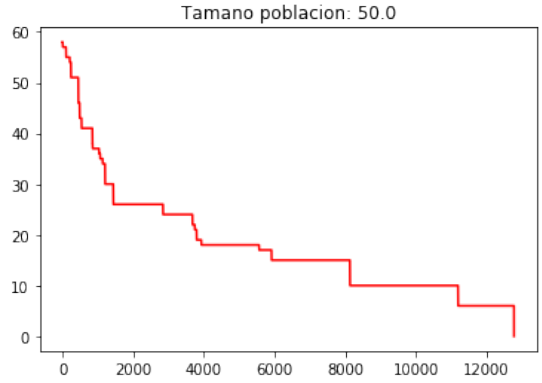

Figure 10: AE en el ejemplo 'extreme_filled' con población 50 .
Fuente: Elaboración propia

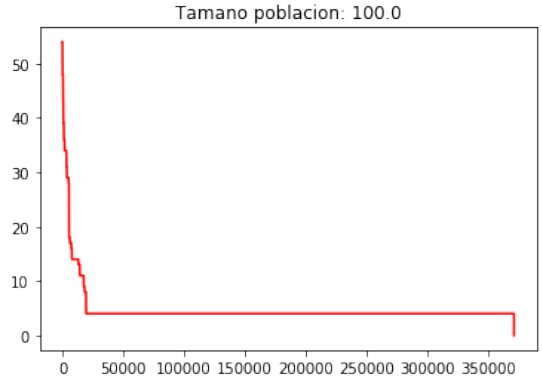

Figure 11: AE en el ejemplo 'extreme_filled' con población 100 .
Fuente: Elaboración propia

.

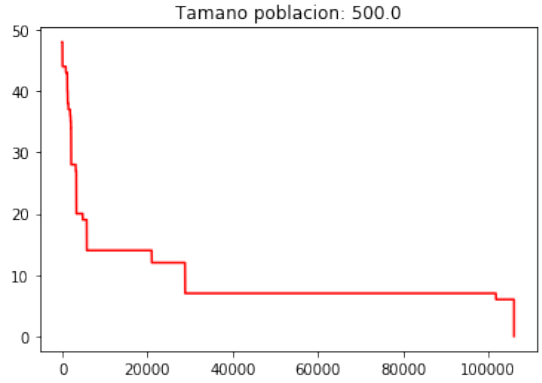

Figure 12: AE en el ejemplo 'extreme_filled' con población 500 .
Fuente: Elaboración propia

Como se mencionó, puede apreciarse inmediatamente que la convergencia entre las primeras generaciones es veloz, mientras que la parte final del algoritmo es netamente estocastica la cantidad de generaciones que tomará encontrar la mutación requerida para llegar a la solución. Aún así, la solución es encontrada, y a la vez como se menciona anteriormente, una población pequeña tiene menos individuos a los que mutar, también sus generaciones son mucho más veloces que las de grandes cantidades de cromozomas por generación, lo que resulta más eficiente a la hora de realizar experimentos.

9 Conclusiones

Probablemente una de las conclusiones más importantes para este problema, es que para un humano, mientras mayor sean las pistas, más sencillo es resolver el problema, sin embargo esto mismo le generá más restricciones al algoritmo, otorgandole una mayor dificultad. Es importante mencionar que una gran cantidad de pistas, implicaría una gran cantidad de restricciones extra por lo que en esos casos la dificultad no aumentaría sino que incluso sería fácil de resolver como se ha visto en clases para CSP con un cantidad importante de restricciones.

Por otra parte las decisiones de separar los valores de las casilla con las restricciones del killer sudoku, fue un momento crucial para lograr reducir la dificultad de mitigar la mezcla o desorden de las restricciones. Así como también fue importante la utilización del movimiento swap, junto a la inicialización para reducir el espacio de búsqueda.

Un conclusión particular del análisis de resultados, fue que en términos prácticos los valores de poblaciones más eficientes en términos de tiempo transcurrido por generación y cantidad de generaciones utilizadas para encontrar una solución es entre aproximadamente 30 y 50 individuos.

Finalmente, es relevante entender como la mutación es la que finalmente encuentra la solución debido a que el cruzamiento permite llegar a una convergencia rápidamente pero es mediante la mutación que se lográ realizar el trabajo fino sobre el arreglo de valores.

10 Bibliografía

A continuación se presentan todas las referencias utilizadas en este informe.

References

[1] Ch. Bhaskar, K. Krishna, and D. Upendra. An integer programming model for the sudoku problem. *International Journal of Mathematics Trends and Technology*, 40(2), 2016.

[2] Desconocido. Killer sudoku rules. *https://bit.ly/2OdHbGP*, Revisado 2018.

[3] David Eppstein. Solving single-digit sudoku subproblems. *International Conference on Fun with Algorithm*, pages 142–153, 2012.

[4] Bertram Felgenhauer and Frazer Jarvis. Enumerating possible sudoku grids. *Research Gate*, 2005.

[5] Rick France. Howard s garns. *https://bit.ly/2AwX1ZB*, Revisado 2018.

[6] Howard Garns. Number place. *Dell Pencil Puzzles and Word Games*, 16, 1979.

[7] Ed Pegg Jr. Math games, sudoku variations. *https://bit.ly/2Jli4ks*, Revisado 2018.

[8] Apt K. Principles of constraint programming. *Cambridge University Press*, 2003.

[9] Gary Mcguire. There is no 16-clue sudoku: Solving the sudoku minimum number of clues problem. *University College Dublin*, 2012.

[10] Ed Russell and Frazer Jarvis. There are 5472730538 essentially different sudoku grids. *http://www.afjarvis.staff.shef.ac.uk/sudoku/sudgroup.html*, 2005.

[11] Helmut Simonis. Sudoku as a constraint problem. *Research Gate*, 2003.

[12] Shuai Wang and Aashish Venkatesh. A sat attack on killer sudoku. *UNIVERSITY OF AMSTERDAM*, 2016.

[13] Wikipedia. Sudoku. *https://bit.ly/2OdZQCt*, Revisado 2018.