3 PREGUNTA 3

PARTE 1

En esta parte se hizo un estudio experimental con diferentes combinaciones de num_pref y out_of. En la figura 7 se muestran los resultados de estas pruebas en donde los valores eran 1-1, 1-2, 9-10,10-11, 99-100 respectivamente.

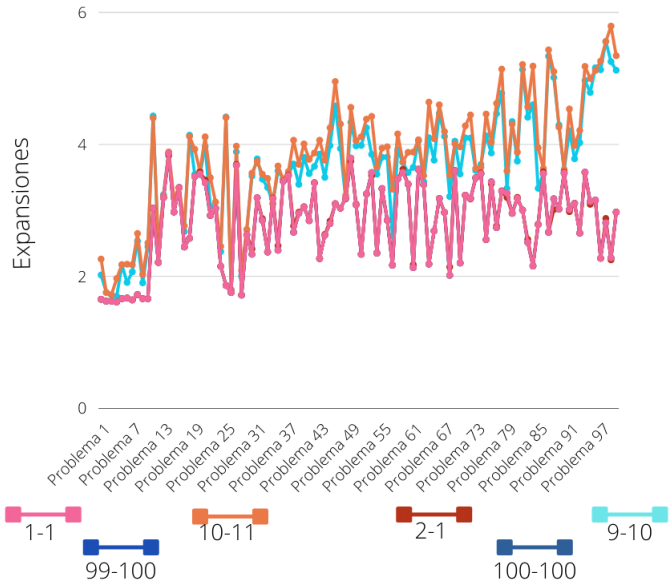

Figura 7: Expansiones del algoritmo probando diferentes combinaciones de num_pref y out_of

En la figura se puede apreciar que cuando igualamos los valores de num_pref y out_of es cuando se consiguen menores valores de expansiones.(1-1 y 100-100 están superpuestos). Esto significa que la mejor configuración para este algoritmo es cuando se sacan todos los nodos de la cola preferred antes que se empiece a sacar de la *OPEN*. Por lo mismo siempre que se prefiera vaciar preferred antes que *OPEN* llegaremos a la mejor configuración del algoritmo.

PARTE 3

Sea s un nodo extraído de Preferred y n un nodo extraído de la *OPEN* por admisibilidad es posible demostrar que ambas colas están acotadas por un valor $c*$.

Si se extrae un nodo de la *OPEN* por admisibilidad se tiene que:

$$f(n) \leq f*(n) = g*(n) + h*(n)$$

$$f(n) \leq f*(n) \leq cost(n_{start}, n*) + h*(n)$$

$$f(n) \leq cost(n_{start}, n*) + cost(n*, n_{goal})$$
$$\leq f*(n) \leq c*$$

De forma Análoga si se extrae un nodo de Preferred:

$$f(s) \leq f*(s) = g*(s) + h*(s)$$

$$f(s) \leq f*(s) \leq cost(n_{start}, n*) + h*(s)$$
$$f(s) \leq cost(s_{start}, s*) + cost(s*, s_{goal})$$
$$\leq f*(s) \leq c*$$

Por lo tanto, sea cual sea la cola de la cual extraemos un nodo, siempre existirá una cota superior para f y con ello, si definimos $c(\pi)$ como el costo óptimo del problema y considerando que de la $OPEN$ o Preferred se extrae el mínimo nos queda que:

$$\frac{c(\pi)}{c^*} \leq \frac{c(\pi)}{\text{mín}_{s \in G}\, g(s) + h(s)}$$

en donde $G = OPEN \cup$ Preferred

PARTE 5

Notemos que esto es una aplicación directa de lo demostrado en el inciso anterior. Sea n el nodo que sacamos de la $OPEN$ o Preferred y si $f(n) > w * m$ con $m = \text{mín}_{s \in G}\, g(s) + h(s)$ y w el parámetro de suboptimalidad. Entonces tenemos que:

$$c* \geq f(n) > w * m$$

Luego,

$$\frac{c(\pi)}{c^*} < \frac{c(\pi)}{w * \text{mín}_{s \in G}\, g(s) + h(s)} \leq \frac{c(\pi)}{\text{mín}_{s \in G}\, g(s) + h(s)}$$

lo que garantiza que las soluciones encontradas satisfagan la cota de suboptimalidad.

PARTE 6

En esta parte se comparó el rendimiento de **Weighted A*** de la parte 1 con el algoritmo recién implementado en los primeros 50 problemas propuestos. Se probó con pesos w= 1.5, 2, 3, 5, y 100. Los resultados se muestran en las figuras adjuntas en esta parte.

Con W=1.5:

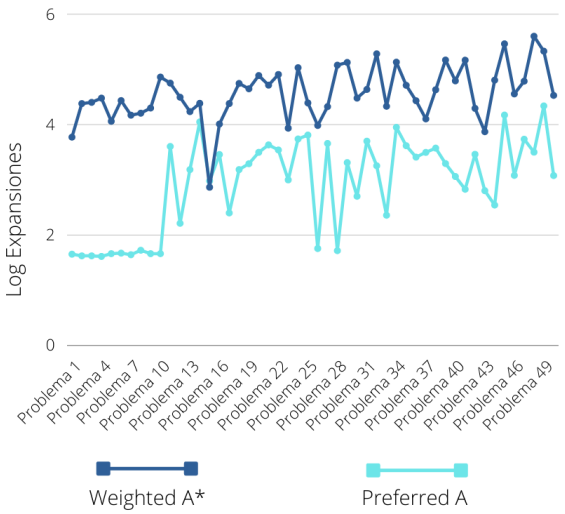

Figura 8: Diferencia de Rendimiento en termino de expansiones de Weighted A* y Preferred A con w= 1.5

Con W=2:

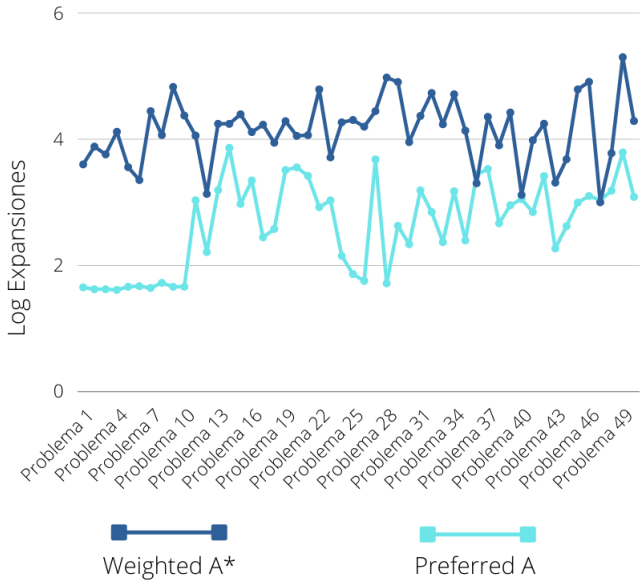

Figura 9: Diferencia de Rendimiento en termino de expansiones de Weighted A* y Preferred A con w= 2

Con W=3:

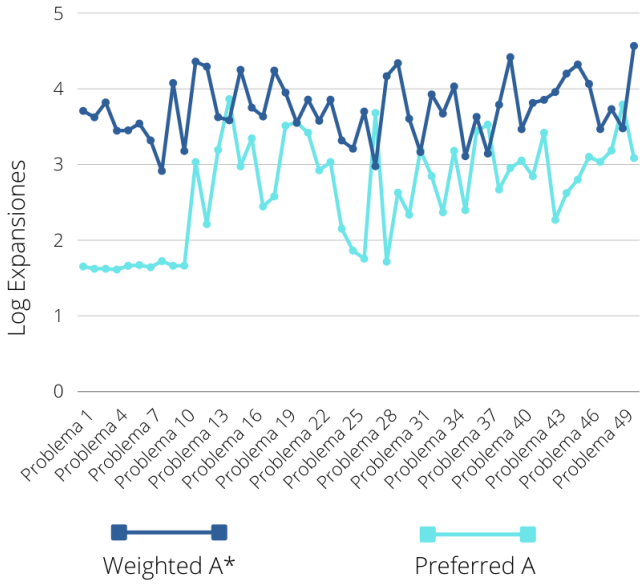

Figura 10: Diferencia de Rendimiento en termino de expansiones de Weighted A* y Preferred A con w= 2

Con W=5:

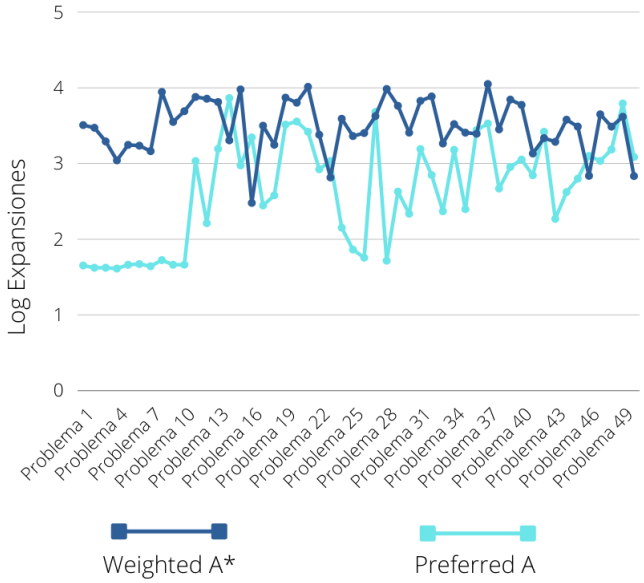

Figura 11: Diferencia de Rendimiento en termino de expansiones de Weighted A* y Preferred A con w= 5

Con W=100:

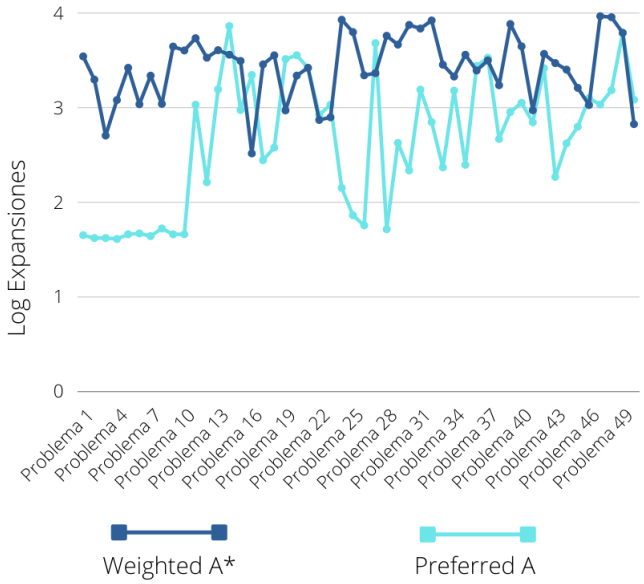

Figura 12: Diferencia de Rendimiento en termino de expansiones de Weighted A* y Preferred A con w=100

En los gráficos es posible notar que el algoritmo preferred A* es muy superior en todos los casos y casi constante luego de w=2. Esto se debe a que ese numero era el umbral de suboptimalidad para preferred A* y si le damos una cota mayor, esta no influirá en el desempeño del algoritmo. Caso contrario sucede con Weighted A en donde a medida que vamos aumentando el peso, mejor se va desempeñando el algoritmo. Sin embargo, ni con w=100 se logra acercar al buen desempeño que posee preferred A*.

BONUS

En esta parte se implementó el archivo best_discrepancy_algorithm.py el cual se basa en utilizar el método k_accs_successors() de la red Neuronal. Este método retorna los nodos hijos dado un nodo cualquiera y además calcula la probabilidad que la NN prefiera cierto nodo siguiente. Por lo mismo, best_discrepancy_algorithm.py funcionará como una versión levemente modificada de $A*$ en donde se sumaran 1 como valor de función a todos los nodos hijos que tengan una probabilidad menor al nodo hijo escogido por la NN. Con ello, los hijos que tienen bajas probabilidades de ser escogidos irán sumando valores cada vez mas grandes y con ello, será mas difícil que salgan de la *OPEN*. En síntesis, la NN funciona como Oráculo y va indicando por donde es mejor seguir expandiendo nodos. En la figura 13 se muestran los resultados obtenidos por este algoritmo comparado con preferred_A y Weighted_A.

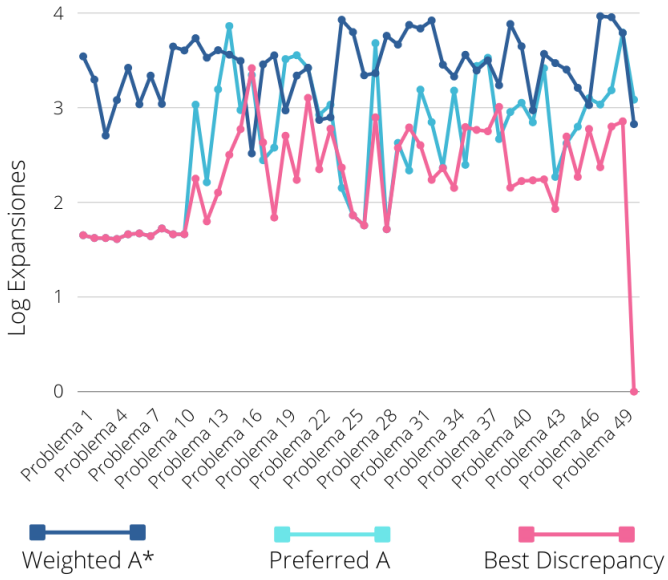

Figura 13: Comparación desempeño de algoritmos Best Discrepancy, Weighted A y Preferred A

Acá se muestra claramente que Best discrepancy es un mejor algoritmo en comparación con el resto medido en la calidad de sus expansiones. Finalmente un forma de garantizar la suboptimalidad del problema podría ser restringiendo la profundidad de búsqueda por parte de best_discrepancy_algorithm.py obligando que el algoritmo encuentre soluciones menos profundas y con ellos más acotadas en termino de la función objetivo. También se podrían penalizar los valores que se suman a la función objetivo si ya se ha avanzado más de lo $w-$subóptimo de tal forma que estos nodos queden mas al fondo de la *OPEN* y el algoritmo prefiera expandir otros nodos que quizás no eran los mas probables que se escogieran por la NN pero si garantizan una cota de suboptimalidad.

Para probar este método será necesario ejecutar la clase B_discrepancy solo con el init, por lo mismo cuando se ejecute la función test_preferred_astar.py se deberá cambiar la linea que inicia la clase de la busqueda por s=B_discrepancy(init).