Trabajo Práctico 6: Satisfacción de restricciones

1. Describir en detalle una formulación CSP para el Sudoku.

2. Utilizar el algoritmo AC-3 para demostrar que la arco consistencia puede detectar la inconsistencia de la asignación parcial WA=red, V=blue para el problema de colorear el mapa de Australia (Figura 6.1 AIMA 3$^{\text{ra}}$ edición).

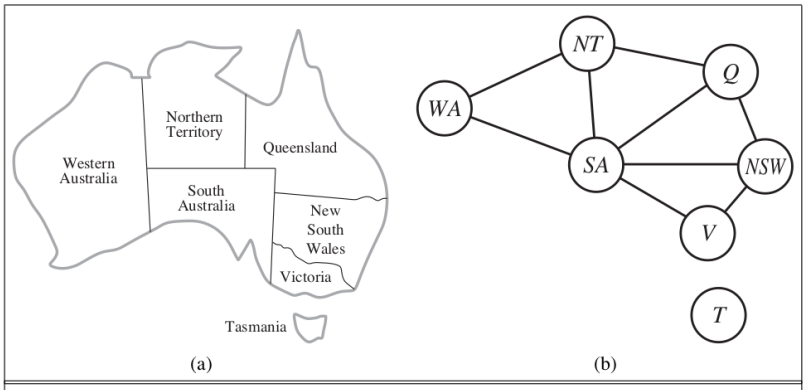

Figure 6.1 (a) The principal states and territories of Australia. Coloring this map can be viewed as a constraint satisfaction problem (CSP). The goal is to assign colors to each region so that no neighboring regions have the same color. (b) The map-coloring problem represented as a constraint graph.

3. ¿Cuál es la complejidad en el peor caso cuando se ejecuta AC-3 en un árbol estructurado CSP? (i.e. cuando el grafo de restricciones forma un árbol: cualquiera dos variables están relacionadas por a lo sumo un camino).

4. AC-3 coloca de nuevo en la cola todo arco (X_k, X_i) cuando cualquier valor es removido del dominio de X_i incluso si cada valor de X_k es consistente con los valores restantes de X_i. Supongamos que por cada arco (X_k, X_i) se puede llevar la cuenta del número de valores restantes de X_i que sean consistentes con cada valor de X_k. Explicar cómo actualizar ese número de manera eficiente y demostrar que la arco consistencia puede lograrse en un tiempo total $O(n^2d^2)$.

5. Demostrar la correctitud del algoritmo CSP para árboles estructurados (sección 6.5, AIMA 3$^{\text{ra}}$ edición). Para ello, demostrar:

   a) Para un CSP cuyo grafo de restricciones es un árbol, la 2-consistencia (consistencia de arco) implica $n-$consistencia, siendo n el número total de variables.

   b) Argumentar por qué lo demostrado en $5a$ es suficiente.

6. Implementar una solución al problema de las $N-$reinas utilizando una formulación CSP:

   a) Implementar una solución utilizando *backtracking*.

   b) Implementar una solución utilizando *forward checking*.

7. Ejecutar 30 veces cada uno de los algoritmos implementados en el ejercicio 6, para el caso de 4, 8 y 10 reinas (opcional: 12 y 15 reinas). Para cada uno de los algoritmos:

a) Generar una tabla con los resultados obtenidos y guardarla en formato `.csv` (*comma separated value*).

b) Calcular:

i) El número (porcentaje) de veces que se llega a un estado de solución óptimo.

ii) El tiempo de ejecución promedio y la desviación estándar para encontrar dicha solución (se puede usar la función `time.time()` de `python`).

iii) La cantidad de estados previos promedio, y su desviación estándar, por los que tuvo que pasar para llegar a una solución.

c) Realizar gráficos de caja y bigotes (boxplots) que muestren la distribución de los tiempos de ejecución de cada algoritmo, y la distribución de la cantidad de estados previos visitados.

d) Comparar los resultados con aquellos obtenidos en el Trabajo Práctico N° 5.

8. Forma de entrega:

a) Dentro del repositorio `ia-uncuyo-2024`, crear una carpeta con el nombre `tp6-csp`.

b) Colocar un archivo con el nombre `tp6-reporte.md` que contenga la respuesta a la preguntas de los ejercicios 1 a 7.

c) Dentro de la carpeta `tp6-csp` crear una nueva carpeta `code` para el proyecto desarrollado en `python`.

d) Colocar un archivo con el nombre `tp6-Nreinas.csv` que contenga la tabla generada en el ejercicio 7*a*.

e) Dentro de la carpeta `tp6-csp`, crear una nueva carpeta `images`, que incluya todos los gráficos e imágenes utilizados en el reporte final.