Complejidad Computacional – Tarea 5

Cova Pacheco Felipe de Jesús

Actividades: 1. Investiga qué son los algoritmos de aproximación y redacta un escrito. 2. Demuestra en detalle si existe o no un algoritmo de aproximación para la versión de optimización del problema de tu tema de exposición. En caso de que no exista tal algoritmo, explica el por qué.

Profesor: María de Luz Gasca Soto
Ayudante: José Luis Vázquez Lázaro

1. Algoritmos de aproximación

Un algoritmo de aproximación es un algoritmo usado para encontrar soluciones aproximadas a problemas de optimización. Están a menudo asociados con problemas NP-hard; como es poco probable que alguna vez se descubran algoritmos eficientes de tiempo polinómico que resuelvan exactamente problemas NP-hard, se opta por encontrar soluciones no-óptimas en tiempo polinomial. A diferencia de las heurísticas, que usualmente sólo encuentran soluciones razonablemente buenas en tiempos razonablemente rápidos, lo que se busca aquí es encontrar soluciones que está demostrado son de calidad y cuyos tiempos de ejecución están acotadas por cotas conocidas. Idealmente, la aproximación mejora su calidad para factores constantes pequeños (por ejemplo, dentro del 5% de la solución óptima). Los algoritmos de aproximación están siendo cada vez más utilizados para resolver problemas donde los algoritmos exactos de tiempo polinomial son conocidos pero demasiado costosos debido al tamaño de la entrada.

Un ejemplo típico para un algoritmo de aproximación es uno para resolver el problema de la cobertura de vértices de la teoría de grafos: encontrar una arista no cubierta y añadir sus dos puntos finales a la cobertura de vértice, y repetir hasta que ya no queden aristas. Es claro que la cobertura resultante será a lo más dos veces del largo de la solución óptima. Este es un algoritmo de aproximación de factor constante con un factor de 2.

Los problemas NP-hard varían mucho en su aproximación; algunos, tales como el problema de la mochila, pueden ser aproximados mediante cualquier factor superior a 1 (tal familia de algoritmos de aproximación se conoce como esquema de aproximación de tiempo polinomial o PTAS). Otros, como el problema de la clique, son imposibles de aproximar dentro de cualquier constante, o incluso factor polinomiales, a menos que P = NP.

Los problemas NP-hard frecuentemente pueden expresarse como programación entera (PE) y ser resueltos exactamente en tiempo exponencial. Muchos algoritmos de aproximación surgen de la relajación de la programación lineal (PL), propia de la programación entera.

No todos los algoritmos de aproximación son adecuados para todas las aplicaciones prácticas. A menudo utilizan resolvedores (solvers) de IP, LP y programación semidefinida, estructuras de datos complejas o técnicas de algoritmos sofisticadas que tienden a dificultar los problemas de implementación. Además, algunos algoritmos de aproximación poseen tiempos de ejecución poco prácticos, incluso a pesar de ser polinómicos, como por ejemplo, del orden de $O(n^{2000})$. Sin embargo, a pesar de esto último, existen problemas donde los altos tiempos de ejecución y costos de

memoria pueden justificarse, tales como los relacionados con la biología computacional, ingeniería financiera, la planificación del transporte, y la gestión de inventario. En estos escenarios, se debe competir contra las correspondientes formulaciones de programación entera directa.

Otra limitación de la aproximación es que esta sólo es aplicable a los problemas de optimización, y no a los problemas de decisión en estado "puro", tales como SAT (a pesar de que es posible representar versiones de optimización para tales problemas, como el respectivo Problema de satisfacibilidad máximo).

2. Algoritmos de aproximación para:

a) Steiner tree

The harndess result

Dado un conjunto S con elementos 1, ..., n y una familia F = {S 1 , S 2 , ..., S m } de subconjuntos de S, el problema de la cobertura del conjunto es encontrar una subfamilia de tamaño mínimo de F que cubre todos los elementos de S. En esta sección demostramos que aproximarnos al problema del árbol terminal de Steiner es al menos tan difícil como aproximar el problema de la cubierta de vértices. Para el último problema, Feige ha demostrado que a menos que NP = DTIME (n O (registro log n)) ningún algoritmo de aproximación de tiempo polinomial puede tener una garantía de rendimiento mejor que (1 - o (1)) ln n

Teorema: Para cualquier constante α un algoritmo de aproximación α del tiempo polinomial para el problema del árbol de Steiner terminal se obtiene un algorítmo α-aproximado de tiempo polinomial para el problema de la cobertura de conjuntos.

Demostración. Usamos una reducción de la cobertura de conjuntos, similar a la utilizada para mostrar la dureza del grupo del problema del árbol de Steiner.
Considerese una instancia del problema de la cobertura de conjuntos (no ponderado) con elementos 1, ..., n y conjuntos S 1 , ..., S m . Nosotros construimos desde esta instancia una instancia del problema del árbol de Steiner terminal como sigue. Tome una estrella con el vértice x como su centro y m los rayos que terminan en los vértices S 1 , ..., S m . Todos estos rayos tienen un peso de 1. Ahora agregue n terminales T 1 , ..., T n y un

terminal extra T 0 . El terminal T 0 está conectado al vértice x por un borde de longitud 0. Para cada j, conecte el terminal T j por bordes de longitud 0 con todos los vértices Si para los que j es un elemento del conjunto S i . Ahora una solución para el problema del árbol Steiner del terminal contiene el vértice x (como el terminal T 0 debe estar conectado con los otros terminales) y algunos de los rayos que emanan de x. El peso de la solución del problema del árbol terminal de Steiner es exactamente el número de estos rayos. Todos los conjuntos S i que están conectados a x por estos rayos forman una cubierta de conjuntos, porque para cada j terminal T j está conectada a otros terminales en la solución del problema del árbol de Steiner terminal solo a través de conjuntos que contienen elementos j. Por lo tanto, el peso de la solución al problema del árbol de Steiner terminal es exactamente tan grande como la solución al problema de cobertura de conjunto que es inducido por este solución. En particular, esto implica que un algoritmo de aproximación α para el el problema del árbol terminal de Steiner, produce un algoritmo de aproximación α para el problema de la cubierta de conjuntos

Tengamos en cuenta que la instancia del problema del árbol terminal de Steiner producida por esta reducción, no es métrica. El teorema dado en un principio, se extiende fácilmente al caso donde α es una función que depende del tamaño de entrada. Sea N el número de vértices en una instancia del problema del árbol terminal de Steiner y sea n y m el número de elementos y conjuntos en una instancia de una cobertura de conjuntos. Entonces un algoritmo α-(N) -aproximado para el problema del árbol Steiner terminal produce un algoritmo de aproximación α (n + m + 2) para el problema de cobertura de conjuntos.

b) Feedback set in digraphs

Resumen del algoritmo:
Dado un gráfico en n vértices y un entero k , el problema de conjunto de vértices de realimentación solicita la eliminación de a lo sumo k vértices para hacer la gráfica acíclica. Mostramos que un algoritmo de ramificación codicioso, que siempre se ramifica en un vértice indeciso con el mayor grado, se ejecuta en exponencial simple tiempo, es decir, $O (c k \cdot n 2)$ para alguna constante c .

Un algoritmo ingenuo para retroalimentación conjunto de vértices

Un algoritmo de ramificación trivial funcionará de la siguiente manera. Recoge un vértice y ramifica incluyéndolo en la solución V - (es decir, eliminándolo de G), o marcándolo como "no se puede vender"
Hasta que el gráfico restante sea ya un bosque. Sin embargo, este algoritmo lleva tiempo O (2 n). Una observación (más bien informal) es que un vértice de mayor grado tiene una probabilidad mayor de ser en un mínimo conjunto de vértices de retroalimentación, inspirando así un algoritmo codicioso de dos fases para el problema. Si hay vértices indecisos de grado mayor que dos después de algún preprocesamiento, entonces siempre se ramifica en un vértice indeciso con el mayor grado.

*Teorema: *El algoritmo codicioso se puede implementar en tiempo* $O (8_k \cdot n_2)$.

Algoritmo naive-fvs (G, k, F **)**

- Entrada: una gráfica G , un entero k y un conjunto $F \subseteq V (G)$ que induce un bosque.

- Salida: un conjunto de vértices de realimentación $V - \subseteq V (G) \setminus F$ de tamaño $\leq k$ o "no".

  0. si $k < 0$ entonces devuelve " no " ; si $V (G) = \varnothing$ entonces regresa ø;

  1. Si un vértice v tiene un grado menor que dos, entonces devuelve naive-fvs ($G - \{ v \}$, k, $F \setminus \{ v \}$);

  2. si un vértice $v \in V (G) \setminus F$ tiene dos vecinos en el mismo componente de $G [F]$ entonces $X \leftarrow$ naive-fvs ($G - \{ v \}$, $k - 1$, F); devuelve $X \cup \{ v \}$;

3. elija un vértice v de $V (G) \setminus F$ con el grado máximo;

4. si $d (v) = 2$ entonces
   4.1. $X \leftarrow \varnothing$;
   4.2. Si bien hay un ciclo C en G entonces tomar cualquier vértice x en $C \setminus F$; agrega x a X y elimínalo de G ;
   4.3. si $| X | \leq k$ luego devuelve X ; de lo contrario devolver "no";

5. $X \leftarrow$ naive-fvs ($G - \{ v \}$, $k - 1$, F); \\ *caso 1:* $v \in V$ - .
   si X no es "no" , devuelva $X \cup \{ v \}$;

6. devuelve naive-fvs ($G, k, F \cup \{ v \}$). \\ *caso 2:* $v \in V$ - .

Un algoritmo simple para la retroalimentación del conjunto de vértices se ramifica de manera codiciosa.

Lema 2.1. *El algoritmo de llamada naive-fvs con (G, k, ⌀) resuelve la instancia (G, k) de problema de conjunto de vértices de retroalimentación*

Demostración. Las dos condiciones de terminación en el paso 0 son claramente correctas. Para cada llamada recursiva.
En los pasos 1 y 2, mostramos que la instancia original es una instancia de sí si y solo si la nueva instancia es una instancia de sí. Tengamos en cuenta que ningún vértice se mueve a F en estos dos pasos. En el paso 1, el vértice v no está en ningún ciclo y, por lo tanto, puede evitarse con cualquier solución. En el paso 2, hay un ciclo que consiste en el vértice v y los vértices en F (cualquier ruta que conecte estos dos vértices en $G [F]$), y por lo tanto, cualquier solución debe contener v .
Para argumentar la corrección del paso 4, mostramos que la solución encontrada en el paso 4 es óptima.

Sea c la cantidad de componentes en G y l el tamaño de las soluciones óptimas. Tenga en cuenta que cada vértice en una solución tiene el grado dos y, por lo tanto, después de eliminar l vértices, la gráfica tiene $n – l$ vértices y al menos $m - 2 l$ *de* aristas. Además, eliminar vértices de una solución óptima no disminuimos el número de componentes del gráfico, tenemos $(n - l) - (m - 2 l) \geq c$.
Por lo tanto, $l \geq m - n + c$, y mostrando $| X | = m - n + c$ terminaría la tarea. Borrando un vértice de grado 2 de un ciclo nunca aumenta el

número de componentes. También tenga en cuenta que una gráfica de c componentes contiene un ciclo si y solo si tiene más de $n - c$ bordes. por lo tanto, el
mientras que el bucle en el paso 4 se ejecutaría exactamente $m - n + c$ iteraciones: después de eliminar $m - n + c$ vértices, cada uno de los grados dos cuando se eliminan, el gráfico restante tiene $2n - m - c$ vértices y $2n - m - 2c$ bordes, que tiene que ser un bosque de c árboles.

Los dos últimos pasos son triviales: si hay una solución que contiene v , entonces se encuentra en el paso 5;
de lo contrario, el paso 6 siempre da la respuesta correcta.

Ahora analizamos el tiempo de ejecución del algoritmo, que es simple pero no trivial. La ejecución del algoritmo se puede describir como un árbol de búsqueda en el que corresponde cada nodo
a dos instancias extendidas del problema, la *instancia de entrada* y la *instancia de salida* . La entrada de la instancia del nodo raíz es $(G, k, \varnothing)$. La instancia de salida de un nodo es la que se encuentra después de los pasos 0–2.
Se han aplicado exhaustivamente en la instancia de entrada. Si se llama además el paso 5, entonces dos nodos secundarios se generan, con instancias de entrada $(G - \{v\}, k - 1, F)$ y $(G, k, F \cup \{v\})$ respectivamente. (Tengamos en cuenta que el algoritmo no puede explorar el segundo hijo, pero esto no es de nuestro interés.) Un nodo de hoja del árbol de búsqueda devuelve una solución o un "no".
Está claro que cada nodo se puede procesar en tiempo polinomial, y por lo tanto el foco de nuestro análisis consiste en enlazar el número de nodos en el árbol de búsqueda. Dado que el árbol es binario, basta con atar su profundidad. Decimos que una ruta desde la raíz del árbol de búsqueda a un nodo hoja es una *ruta de ejecución* . Fijemos una ruta de ejecución arbitraria en el árbol de búsqueda del cual el nodo de hoja devuelve una solución $V -$, y permite que F denote todos los vértices movidos a F por el paso 6 en esta ruta de ejecución. La longitud de esta ruta de ejecución es como máximo $|V -| + |F|$: Cada nodo no-hoja pone al menos un vértice en $V -$ o F. Se nos permite poner como máximo k vértices en $V -$,
es decir, $|V -| \leq k$, y por lo tanto nuestra tarea en el resto de esta sección es consolidar $|F|$.

Empecemos por algunos datos elementales sobre los árboles. Cualquier arbol *t* satisface

$$\sum_{v \in V(T)} d(v) = 2|E(T)| = 2|V(T)| - 2 \quad \text{y} \quad \sum_{v \in V(T)} (d(v) - 2) = -2 .$$

Sea L el conjunto de hojas de T y V_3 el conjunto de vértices de grado al menos tres. Si $V(T) \geq 2$, entonces $|L| \geq 2$ y

$$-2 = \sum_{v \in L} (d(v) - 2) + \sum_{v \in V(T) \setminus L} (d(v) - 2) \geq \sum_{v \in L} (-1) + \sum_{v \in V_3} (d(v) - 2) \geq \sum_{v \in V_3} (d(v) - 2) - |L| .$$

Por lo tanto

$$\sum_{v \in V_3} (d(v) - 2) \leq |L| - 2 .$$