­­­­­­­

[Какой-то титульный лист]

План:

1. Введение
2. Классы P и NP, задачи SAT и CSP
3. Изоморфизм графов и техника нарушения симметрии
4. Нарушение симметрии на основе дерева BFS
5. Подходы, основанные на матрице смежности
6. Choco и choco-graph
7. Реализация

Введение

Теория графов является очень важным разделом информатики. Графы используются повсюду – в картографии, в компьютерной графике, в сетях и даже в военной промышленности.

Часть графовых задач сводится к поиску графов с заданными свойствами. Существует множество подходов к решению таких задач, каждый из которых чем-то лучше других. Алгоритмы поиска бывают точные и приближенные, узкоспециализированные и более общие, работающие за полиномиальное время и за экспоненциальное и т.п. Некоторые задачи поиска графов лежат в классе P и для них известны эффективные полиномиальные алгоритмы нахождения решения. Но многие задачи поиска графов не лежат в P, зато лежат в классе NP (существует полиномиальный сертификат, который можно проверить за полиномиальное время). Также известно, что задачи SAT и CSP являются NP-полными. Поэтому все графовые задачи из NP можно свести к одной из этих двух задач. Это распространенный подход, потому что для задач SAT и CSP существует множество очень хорошо оптимизированных солверов, которые решают эти задачи достаточно быстро.

Однако существует проблема в подобных задачах. Для многих задач поиска изоморфные графы эквивалентны (в смысле эквивалентности как решение задачи). Тогда при поиске решения будут проверены все изоморфные графы, что сильно замедляет поиск по сравнению с тем, если бы из всех изоморфных графов проверялся бы только один.

В данной работе были изучены некоторые способы так называемого нарушения симметрии, одного из способов не рассматривать множество изоморфных графов при поиске, а также до некоторой степени изучен SAT и CSP солвер choco и его модуль choco-graph для манипуляции графовыми переменными. Также на языке choco были реализованы предикаты нарушения симметрии из работы [\*] и проверена их эффективность на некоторых конкретных задачах поиска графов.

Классы P и NP, задачи SAT и CSP

На множестве языков над некоторыми объектами можно выделить несколько так называемых сложностных классов. Примерами таких классов являются классы P, NP, NP-трудные и NP-полные. В качестве объектов, над которыми строится язык, будем рассматривать графы. Каждая задача определяет некоторый язык L = {G | G – решение задачи}. Тогда вышеупомянутые классы сложности можно распространить и на задачи, имея в виду, что задача лежит в некотором классе, если язык, который она определяет, лежит в этом классе.

Класс P определяется следующим образом: язык L лежит в классе P, если существует программа q и полином p такие, что G ∈ L тогда и только тогда, когда q(G) = 1, причем q работает за O(p(|G|)). Такие задачи (языки) вычислительно более простые и для большинства из них известны полиномиальные алгоритмы, которые их решают. Однако существует множество задач, которые не лежат в классе P.

Рассмотрим одно из альтернативных определений класса NP: язык L лежит в классе NP, если существует полином p и полиномиальное отношение R(x, y) такие, что объект G ∈ L тогда и только тогда, когда существует “сертификат” y, такой, что |y| ≤ p(|x|) и R(G, y) = 1. Если для заданного объекта G легко построить сертификат y, то задачу поиска какого-нибудь решения можно решать перебором: перебирать все возможные объекты G, для каждого строить сертификат и за полиномиальное время проверять, правда ли R(G, y) = 1.

Еще одним классом сложности является класс NP-трудных задач. Задача называется NP-трудной, если любая задача из класса NP сводится к ней по Карпу за полином (существует полиномиальное отображение f : X → Y, такое, что x ∈ Lx тогда и только тогда, когда f(x) ∈ Ly). Задача называется NP-полной, если они лежит одновременно и в классе NP-трудных, и в классе NP.

NP-трудные и NP-полные задачи полезны тем, что если мы умеешь решать такую задачу за O(T), то любую другую задачу из NP мы можем решать за O(T + p), где p – некоторый полином.

Примером NP-полной задачи является SAT (*boolean SATisfiability*). Эта задача определяет, является ли заданная булева формула φ(x1, x2, …, xn) удовлетворимой, т.е. существует ли набор значений x1, x2, …, xn­ такой, что φ(x1, x2, …, xn) = 1.

Другой NP-полной задачей является CSP (*constraint satisfaction problem*). Постановка задачи: дан набор переменных a1, a2, …, an и набор ограничений – предикатов от этих переменных (F­­1(a1, a2, …, an), …). Требуется определить, существуют ли значения переменных a1, a2, …, an, такие, что все предикаты будут одновременно выполнены.

Благодаря полноте задач SAT и CSP, все задачи из NP, в том числе графовые, можно свести к одной из них. Этот метод часто используется, так как не существует представления, как решать произвольную задачу поиска графа с некоторыми нетривиальными свойствами. Однако можно закодировать эти свойства в части формулы для SAT или в ограничения для CSP, а для этих задач существует множество эффективных солверов, которые достаточно быстро решают подобные задачи.

Изоморфизм графов и техника нарушения симметрии

Существует множество задач поиска графов с заданными свойствами или доказательства того, что таких графов не существует. Многие их них решаются с помощью сведения к задачам SAT или CSP. Во многих подобных задачах графы можно рассматривать с точностью до изоморфизма.

Два графа называются изоморфными, если существует такая нумерация вершин первого, что он совпадет со вторым. Или, более формально, G1 = (V1, E1) и G2 = (V2, E2) называются изоморфными, если |V1| = |V2| = n и существует π: перестановка n чисел, такая, что (u, v) ∈ E1 ⇔ (π(u), π(v)) ∈ E2. Изоморфизм обозначают как G1 ~ G2. Класс эквивалентности графа G обозначают как C(G). Задача (некоторый предикат допуска P) рассматривает графы с точностью до изоморфизма, если ∀ G̃ ∈ C(G): P(G) = P(G̃), то есть свойство графа либо выполнено целиком для всего класса эквивалентности, либо не выполнено ни для одного его представителя. Если в графе изначально не было выделенных вершин и ребер, то любое свойство графа, которое не использует его нумерацию, рассматривает его с точностью до изоморфизма. Поэтому, если при решении подобных задач рассматривать только одного представителя из каждого класса эквивалентности, то скорость решения задачи возрастет.

Предикат (свойство) графа P называется предикатом нарушения симметрии, если для каждого класса эквивалентности существует хотя бы один его представитель, который удовлетворяет предикату, при этом как можно меньше представителей из одного класса ему удовлетворяют. Такие предикаты можно определить не только для графов, а для любых объектов, обладающих изоморфизмом.

Нарушение симметрии на основе дерева BFS

В [\*] был предложен метод нарушения симметрии для нарушения симметрии в конечных автоматах без бесполезных состояний, который можно адаптировать для некоторых классов графов.

Рассмотрим конечный автомат со стартовым состоянием 0, такой, что из него по переходам достижимы все остальные состояния. Если в автомате существуют состояния, не достижимые из стартового, то они “бесполезные” и существует эквивалентный автомат с количеством состояний, меньшим данного. Такие автоматы рассматривать не рассматриваются в силу их избыточности. Определим предикат P(A), который выполняется тогда и только тогда, когда нумерация автомата совпадает с обходом BFS из стартового состояния. В автомате без бесполезных состояний всегда существует дерево BFS, поэтому предикат всегда корректно определен. Для каждого автомата найдется такая нумерация вершин, что она совпадет с порядком обхода BFS (так как дерево BFS существует, в качестве нумерации подойдет порядок обхода). Следовательно, ∀A ∃Ã ∈ C(A): P(Ã) = 1. Значит, P(A) – предикат нарушения симметрии. Для каждого класса эквивалентности такой предикат допускает не более n автоматов, так как если стартовое состояние фиксировано, то существует ровно одна нумерация в порядке обхода BFS, а стартовым состоянием может быть любое, то есть n. Также в [\*] был предложен способ (набор предикатов), как закодировать этот предикат непосредственно в системах SAT и CSP. Это ограничение можно переформулировать и для графовых задач, однако оно перестает быть предикатом нарушения симметрии для произвольных графов. Но для более узких классов графов он остается нарушением симметрии: данный предикат является предикатом нарушения симметрии для двух классов графов – связных неориентированных графов и ориентированных графов, в которых существует исходящее дерево из 0. Второй класс недостаточно широкий для повсеместного использования, однако, существуют задачи, в которых решение лежит в этом классе, так что на практике он может применяться. А для неориентированных графов класс связных графов достаточно широк и в некотором смысле полон, так как любой неориентированный граф можно представить как объединение компонент связности, каждая из которых по отдельности представляет собой связный неориентированных граф. В этом ключе BFS-ориентированный предикат нарушения симметрии наиболее полезен для неориентированных графов и связанных с ними графовых задач.

Подходы, основанные на матрице смежности

Эти подходы позволяют определить ограничения нарушения симметрии для неориентированных графов. Для этого требуется несколько дополнительных определений.

*Матрица смежности*: AG[u, v] = 1 ⇔ (u, v) ∈ E

*Минимальная степень вершины*: δ(G) = min deg v (v ∈ V)

*Максимальная степень вершины*: Δ(G) = max deg v (v ∈ V)

*Строки матрицы*: A[i] – i-ая строка матрицы A

*Лексикографичекский порядок*: для строк обычный, для матриц A ≼ B если A[0]A[1]...A[n-1] ≼ B[0]B[1]...B[n-1], где A[i]A[j] – конкатенация, лексикографический порядок на графах: G1 ≼ G2 если AG1 ≼ AG2.

*Перестановка вершин графа*: пусть есть перестановка π : {0 .. n - 1} → {0 .. n - 1}, тогда π((V, E)) = (V, {(π(u), π(v)) | (u, v) ∈ E})

*Каноническая форма графа*: can(G) = min≼ {π(G) | π – перестановка}

Лексикографическое нарушение симметрии

Лексикографическим предикатом нарушения симметрии назовем следующее: sblex(G) = AND {AG[i] ≼ AG[i + 1] | i ∈ {0, …, n - 1}}. Этот предикат был введен в [\* Miller, Prosser, 2012]. В [\*] было доказано,что ∀G: sblex(can(G)) = 1, то есть sblex является корректным предикатом нарушения симметрии.

Расширенный лексикографический порядок

Ограничение sblex может допускать много графов из одного и того же класса эквивалентности. Для того, чтобы усилить ограничение sblex потребуются дополнительные определения.

Пусть s – некоторая последовательность и I ⊂ {1, …, |s|}. Будем обозначать (s ↾ I) последовательность, из которой выбросили элементы на I[0], I[1], … местах.

*Расширенный лексикографический порядок:* s1 ≼I s2 ⇔ (s1 ↾ I) ≼ (s2 ↾ I).

На основе этого в [\*] был введен новый предикат нарушения симметрии и была доказана его корректность: sb\*lex (G) = AND {A[i] ≼{i, j} A[j] | i < j}.

Choco и choco-graph

Choco – один из солверов, позволяющий эффективно решать задачи SAT и CSP. В исходном солвере переменные могут быть целыми числами, boolean’ами, множествами и т.д. Модуль choco-graph добавляет новый тип переменных – графы. Графы бывают ориентированные (IDirectedGraphVar) и неориентированные (IUndirectedGraphVar). Они оба описываются интерфейсом IGraphVar. Вершины графа называются узлами (node), соединения ориентированного графа – дугами (arc), неориентированного – ребрами (edge).

В choco каждая переменная имеет свою область допустимых значений (domain). Это непрерывный отрезок, в котором обязательно должна содержаться данная переменная. Для целых чисел это отрезок [a, b]. Если a = b, то переменная ‘инстанциирована’ (instantiated), то есть её значение строго определено. В choco-graph для представления графов используется аналогичная концепция ‘графовых интервалов’. Графовый интервал [G1, G2] – это множество графов G, таких, что любое ребро и любая вершина из G1 содержатся в G и никакая вершина или ребро, которая не содержится в G2 не содержится и в G. То есть G ∈ [G1, G2] ⇔ G1 ⊂ G ⊂ G2. Диапазон [G1, G2] не пустой тогда и только тогда, когда G1 ⊂ G2.

Ограничение (constraint) – это именованное множество propagator’ов. Propagator – это класс, который реализует две функции: propagate и isEntailed. Функция propagate берет область значений переменной и выбрасывает оттуда те значения, на которых ограничение никогда не будет выполнено. Функция isEntailed возвращает ESat.TRUE, если для любой точки промежутка ограничение выполнено, ESat.FALSE, если ни для какой точки промежутка ограничение не выполнено и ESat.UNDEFINED иначе.

Существенного ускорения поиска решения можно добиться, указывая стратегию обхода значений переменных. Так, например, ISF.lexico(...) задает лексикографический обход заданного набора целочисленных переменных. Это может существенно ускорить поиск во многих ситуациях. В choco-graph также есть свои стратегии обхода графовых интервалов: GraphStrategyFactory.lexico(...) обходит графы в лексикографическом порядке (сначала пустой граф, потом все графы с 1 ребром по порядку номеров ребер, потом с 2-умя и т.д.) и GraphStrategyFactory.random(...), которая обходит графовый интервал в случайном порядке. Также можно создавать собственную стратегию обхода переменных.

Реализация

В ходе работы были реализованы предикаты нарушения симметрии на основе дерева BFS [\*] с использованием choco 3.3.0 и choco-graph. В качестве тестов корректности и скорости работы была использована следующая задача.

Даны числа n, m и l. Требуется определить, существует ли неориентированный связный граф из n вершин, m ребер и с обхватом l. Обхватом графа называется длина кратчайшего цикла в графе.

Были проведены эксперименты с двумя решениями этой задачи – простым решением и тем же решением, в которое был добавлен предикат нарушения симметрии. Результаты измерения времени приведены в таблице.

Таблица 1. Время работы в случае существования решения на входах вида (n, n, n - 1)

|  |  |  |
| --- | --- | --- |
| n, m, l | T1, с | T2, с |
| 5, 5, 4 | 0.20 | 0.31 |
| 6, 6, 5 | 0.25 | 0.37 |
| 7, 7, 6 | 0.38 | 0.50 |
| 8, 8, 7 | 1.2 | 0.8 |
| 9, 9, 8 | 11.6 | 1.6 |
| 10, 10, 9 | 226 | 4.2 |
| 11, 11, 10 | 5525 | 14.6 |
| 12, 12, 11 | \* | 81.1 |
| 13, 13, 12 | \* | 354 |