

「スタンフォード・グラフベース

：組合せ計算のためのプラットフォーム」

ドナルド・アービン・クヌース博士

汎用性の高いプログラム及びデータ集が、組合せアルゴリズムやデータ構造を専門とする研究者の手に届くようになりました。ファイルはすべてパブリックドメインに置かれ、「マスタファイル自体は変更されてはならない」という制約さえ守れば誰でも使えます。「ファイル変更」機能により、ローカルでカスタマイズすることができますが、その際マスタファイルは変更されず残ります。

このプログラムは「文芸的プログラミング」の例としてそれ自身興味深くなるように書かれています。したがって、“スタンフォード・グラフベース”は、アルゴリズムを研究していてもしていなくても、プログラマが楽しんで読める約30のエッセイ集と考えるのもよいでしょう。このプログラムは、**T_EX**と**C**を組合せた記述言語**CWEB**で書かれており、これらの2つの言語を知っている者なら誰でも簡単に使用でき、また**C**の基礎を習得した人なら簡単に読むことができます。(CWEBシステムはそれ自身汎用性を持ち、パブリックドメインに置かれています。)

次の4つのプログラムモジュールが“グラフベース”のカーネルを構成しています。

GB_FLIP は、汎用乱数ジェネレータであり、

GB_GRAPH は、グラフの標準データ構造を定義し、記憶領域割付けのルーチンを有し、

GB_IO は、データを読み取り、データが壊れていないことを確認し、

GB_SORT は、節の連結リストにおいて32ビットキーに対する汎用の整列ルーチンです。

他のプログラムは、すべてカーネルの**GB_GRAPH**及び上記の他の3つのモジュールのサブセットに依存します。

10数個のジェネレータ・モジュールがアルゴリズム研究において特に興味深いグラフを作図します。例えば、**GB_BASIC**は、 d 次元のチェス盤の上で選択された座標において「ぐるりとまわる」クイーンの動きのグラフなど、標準的なグラフを作る12のサブルーチンを含んでいます。また、もう一つのジェネレータ・モジュールである**GB_RAND**は様々な種類のランダムなグラフを作ります。

各グラフは、一意に決まる識別子を持ち、これにより、たとえグラフが任意に作られた「ランダム」な場合でも、世界中の研究者が全く同一のグラフを研究できるようになっています。これにより、反復実験やベンチマークテストが可能になり、多岐に渡って利用できるようになります。

ほとんどのジェネレータ・モジュールは“データセット”を利用しますが、これは著者が刊行する予定の組合せアルゴリズムの研究に関する本の中で、興味深くそしてわかりやすい例を提出するため、20年間にわたり収集してきたものです(『The Art of Computer Programming』第4A、4B、4C巻)。例えば、“データセット”のひとつに、**words.dat**と呼ばれるものがありますが、これは著者の読書経験に照らし「完全」だと思える5文字からなる英単語を収集したものです。必要な場合に最頻出単語を抽出できるように、各単語には様々な標準的著作全集における使用頻度が付記されています。**GB_WORDS**は5文字のうち4文字の位置が同じときに、二つの単語

は隣り合うとして、単語の部分集合をグラフにします。こうして、7つのステップで「words」から「graph」に到達することができます。

words, wolds, golds, goads, grads, grade, grape, graph.

実際、これは words.dat から導くことができた最短の連鎖です。

また、10 数個のデモンストレーション・モジュールが提供され、生成されたグラフがどのように使用されるかを示しています。例えば、LADDERS モジュールは対話型のプログラムで、データの任意の部分集合を用いて上に示したような5文字の単語の連鎖を作ります。単語の選択肢を、収集したすべての約 5700 語の代わりに最頻出単語 2000 語に限定すれば、「words」から「graph」への最短経路は 20 になります

words, lords, loads, leads, leaps, leapt, least, lease, cease, chase, chose, chore, shore, shone, phone, prone, prove, grove, grave, grape, graph.

このテーマで、いくつかの変形が実現されています。例えば、隣り合う単語の距離をアルファベットの距離で考えるならば、「words」から「graph」までの最短経路は、

words(3), woods(16), goods(14), goads(3), grads(14), grade(12), grape(3), graph,

のように、全長 65 となります。

LADDERS モジュールは別の GB_DIJK と呼ばれる”グラフベース “のモジュールを使用し、このグラフベース・モジュールは、ダイクストラの最短経路アルゴリズムを実行し、また、ユーザが任意に優先度付き待ち行列の実装を組み込むことも可能です。こうして、異なった待ち行列の効率を比較することができます。

GB_WORDS により作られたグラフは無向ですが、GB_ROGET のような他のジェネレータ・モジュールは、有向グラフを作ります。ロジェの有名な 1882 年の「シソーラス」(類語辞典)は全ての概念を 1022 のカテゴリーに分類しましたが、これを私達はグラフの頂点と呼んでいます。すなわち、ロジェの辞書の中で「u」のカテゴリーが「v」のカテゴリーへの相互参照を含むとき、「u」から「v」へ弧が書かれます。

ROGET_COMPONENTS と呼ばれるデモンストレーション・モジュールは GE_ROGET によって生成されたグラフの強連結成分を決定します。これは、タージャンの強連結成分と有向グラフのトポロジカル整列アルゴリズムをプログラム化したものです。

同様に、GB_BOOKS モジュールを使うと、世界文学も無向グラフのさらに興味深いファミリーとなります。anna.dat 及び david.dat, homer.dat, huck.dat, jean.dat の5つの”データセット”は、「アンナ・カレーニナ」及び「デビッド・カッパーフィールド」、「イリアッド」、「ハックルベリー・フィン」、「レ・ミゼラブル」に関する情報を提供します。お察しかもしれませんが、各作品の登場人物がグラフの頂点になっています。各頂点に対応する登場人物が、本の中の選択されたある章において、互いに遭遇することになる時に、二つの頂点が隣り合うこととなります。BOOK_COMPONENTS と呼ばれるデモンストレーション・プログラムは、ホップクロフトとタージャンの美しいアルゴリズムを用いてこれらのグラフのブロック(例えば2連結成分)を検出します。

別のモジュールである GB_GAME は大学のフットボールの点数に基づいてグラフを生成します。アメリカを代表する 120 チームの 1990 年のシーズン以降の全試合が、games.dat に記録されています。このデータは、完結した小さな「クリークな」グラフになります。なぜなら、たいていのチームはどこかのリーグに属し、そのリーグ内で他の全部のチームと対戦するからです。しかしながら、グラフ全体は連結されています。FOOTBALL と呼ばれるデモンストレーション・モジュールは長い点数の連鎖を検出し、例えば、もしスタンフォード大学がハーバード大学と対戦していたら、2000 点以上も差をつけ勝ったであろうことを証明できます。というのは、スタンフォード大学はノートルダム大学に 5 点差で勝ち、ノートルダム大学は空軍大学に 30 点、

空軍大学はハワイ大学に 24 点差で勝ち、そして…エール大学はハーバード大学に 15 点差で勝っているからです。(逆に、同様な「証明」により、ハーバード大学がスタンフォード大学に 2000 点以上も上位にランクされることにもなります。)このような問題の最適解を見出すよいアルゴリズムは発見されていないので、アルゴリズムが進歩する課程で、こうしたデータが、研究者によりよい技法で求めたよりよい解を披露する機会を提供してくれるのです。

GB_ECON モジュールは、アメリカ経済における産業間の貨幣の流れに基づいて有向グラフを生成します。経済活動は、このモデルでは 2 から 79 までのいくつかの部門にも分割できるので、様々なグラフが得られます。デモンストレーション・プログラムの ECON_ORDER は「供給者」から「消費者」の順番で部門を位置付けるようにしています。つまり、行列の対角線よりも上にある数字の合計が最小となるように、行や列の並べ変えを行います。この問題に関して適度に効率的なアルゴリズムは知られていますが、非常に複雑です。「貪欲(greedy)」と「慎重(cautious)」の二つの発見的手法が比較のために実装されています。少なくとも、経済の分野では貪欲が勝るようです。

北アメリカの 128 都市を結ぶハイウェイ距離は miles.dat にあり、GB_MILES モジュールはこのファイルを用いて様々なグラフを生成します。中でも特に興味深いのは MILES_SPAN と呼ばれるデモンストレーション・モジュールで、これは GB_MILES によって出力されたグラフの最小木を計算します。最小木を求める 4 つのアルゴリズムが実装及び比較されますが、中には理論的には魅力があってもあまり実用的でないアルゴリズムも含まれます。このデモンストレーションは、「メモリ計算」と呼ばれるアルゴリズムを比較する方法が、簡単に実装でき、また計算機の種類によらず効率性を計測できるため、競合する技法を公平に比較できることを示しています。

GB_RAMAN と呼ばれるジェネレータ・モジュールは、コミュニケーションに役立つ拡張グラフの役割を果たす重要な「ラマニヤン・グラフ」を作り出します。GIRTH と呼ばれるデモンストレーション・モジュールはラマニヤン・グラフの最短回路や直径を計算します。GB_BASIC や GB_RAMAN によって作り出されたようなグラフは厳格な数学的構造を持つことに注意してください。一方、GB_ROGET や GB_MILES によって作り出されたようなグラフは、より「有機的な」性格を持っています。パフォーマンスに有意な差があるかどうかを見るために、両方の種類のグラフにおけるアルゴリズムをテストすることは興味深く重要です。

GB_GATES と呼ばれるジェネレータ・モジュールは論理回路のグラフを作り出します。このグラフのファミリの 1 つは、単純な RISC チップ、つまり可変個のレジスタをもつプログラム可能なマイクロコンピュータと同等です。このようなゲートの「メタ・ネットワーク」を用い、自動設計アルゴリズムを広いパラメタの範囲でテストすることが可能です。デモンストレーション・モジュール TAKE_RISC は、サンプルプログラム上でチップの実行をシミュレートします。別のゲートのメタ・ネットワークは複数個の m ビットの数と複数個の数または一つの n ビットの定数との並列乗算を実行します。MULTIPLY モジュールは、これらの回路をデモンストレーションします。

平面グラフは GB_PLANE によって生成されます。これは最近著名なデローニーの三角化アルゴリズムの他いろいろなアルゴリズムの実装を含んでいます。

ピクセルデータは興味深い 2 部グラフになります。レオナルド・ダ・ビンチの「モナリザ」は lisa.dat で表わされ、これは GB_LISA によって異なった種類のグラフに変換されたピクセルの配列です。デモンストレーション・ルーチンの ASSIGN_LISA は各行、各列から 1 ピクセルを選択することによって割当問題を解き、選択されたピクセル全体の輝度を最大にすることができます。ここで解かれた割当問題は芸術作品としての批判や賞賛とは無縁ですが、教育的な価値は多大です。なぜなら、おそらく大きな数値配列の特徴を理解するためには、画像として配列を視

る以上に良い方法がないからです。

GB_SAVE と呼ばれるモジュールは、”グラフベース “によるグラフを、他のグラフ操作システムと容易にデータ交換の出来る ASCII フォーマットに相互変換します。

詳しい情報については 1993 年に ACM 出版から刊行された「スタンフォード・グラフベース」をご覧ください。この本はまた「スタンフォード・グラフベースで楽しくゲーム」とも称することができます。なぜなら、デモンストレーション・プログラムはすばらしい玩具だからです。実際、著者は最も真剣な研究は、また、最大の楽しみとなると固く信じています。研究を楽しむことは、罪ではないのです。

“The Standard GraphBase: A Platform for Combinatorial Computing”

Dr. Donald Ervin Knuth

A highly portable collection of programs and data is now available to researchers who study combinatorial algorithms and data structures. All files are in the public domain and usable with only one restriction: They must not be changed! A “change file” mechanism allows local customization while the master files stay intact.

The programs are intended to be interesting in themselves as examples of “literate programming.” Thus, the Standard GraphBase can also be regarded as a collection of approximately 30 essays for programmers to enjoy reading, whether or not they are doing algorithmic research. The programs are written in CWEB, a combination of T_EX and C that is easy to use by anyone who knows those languages and easy to read by anyone familiar with the rudiments of C. (The CWEB system is itself portable and in the public domain.)

Four program modules constitute the *kernel* of the GraphBase:

GB_FLIP is a portable random number generator;

GB_GRAPH defines standard data structures for graphs and includes routines for storage allocation;

GB_IO reads data files and makes sure they are uncorrupted;

GB_SORT is a portable sorting routine for 32-bit keys in linked lists of nodes.

All of the other programs rely on GB_GRAPH and some subset of the other three parts of the kernel.

A dozen or so *generator modules* construct graphs that are of special interest in algorithmic studies. For example, GB_BASIC contains 12 subroutines to produce standard graphs, such as the graphs of queen moves on d -dimensional rectangular boards with “wrap-around” on selected coordinates. Another generator module, GB_RAND, produces several varieties of random graphs.

Each graph has a unique identifier that allows researchers all over the world to work with exactly the same graphs, even when those graphs are “random”. Repeatable experiments and standard benchmarks will therefore be possible and widely available.

Most of the generator modules make use of *data sets*, which the author has been collecting for 20 years in an attempt to provide interesting and instructive examples for some forthcoming books on combinatorial algorithms (*The Art of Computer Programming*, Volumes 4A, 4B, and 4C). For example, one of the data sets is **words.dat**, a collection of 5-letter words of English that the author believes is “complete” from his own reading experience. Each word is accompanied by frequency counts in various standard corpuses of text, so that the most common terms can be singled out if desired. GB_WORDS makes a subset of words into a graph by saying that two words are adjacent when they agree in 4 out

of 5 positions. Thus, we can get from **words** to **graph** in seven steps:

words, wolds, golds, goads, grads, grade, grape, graph.

This is in fact the shortest such chain obtainable from **words.dat**.

A dozen or so *demonstration modules* are also provided, as illustrations of how the generated graphs can be used. For example, the LADDERS module is an interactive program to construct chains of 5-letter words like the one just exhibited, using arbitrary subsets of the data. If we insist on restricting our choices to the 2000 most common words, instead of using the entire collection of about 5700, the shortest path from *words* to *graph* turns out to have length 20:

words, lords, loads, leads, leaps, leapt, least, lease, cease, chase, chose, chore, shore, shone, phone, prone, prove, grove, grave, grape, graph.

Several variations on this theme have also been implemented. If we consider the distance between adjacent words to be alphabetic distance, for example, the shortest path from *words* to *graph* turns out to be

words(3) woods(16) goods(14) goads(3) grads(14) grade(12) grape(3) graph,

total length 65.

The LADDERS module makes use of another GraphBase module called GB_DIJK, which carries out Dijkstra's algorithm for shortest paths and allows the user to plug in arbitrary implementations of priority queues so that the performance of different queuing methods can be compared.

The graphs produced by GB_WORDS are undirected. Other generator modules, like GB_ROGET, produce directed graphs. Roget's famous *Thesaurus* of 1882 classified all concept into 1022 categories, which we can call the vertices of a graph; an arc goes from *u* to *v* when category *u* contains a cross reference to category *v* in Roget's book. A demonstration module called ROGET_COMPONENTS determines the strong components of graphs generated by GB_ROGET. This program is an exposition of Tarjan's algorithm for strong components and topological sorting of directed graphs.

Similarly, world literature leads to further interesting families of undirected graphs via the GB_BOOKS module. Five data sets **anna.dat**, **david.dat**, **homer.dat**, **huck.dat**, and **jean.dat** give information about *Anna Karenina*, *David Copperfield*, *The Iliad*, *Huckleberry Finn*, and *Les Misérables*. As you might expect, the characters of each work become the vertices of a graph. Two vertices are adjacent if the corresponding characters encounter each other, in selected chapters of the book. A demonstration program called BOOK_COMPONENTS finds the blocks (i.e., biconnected components) of these graphs using the elegant algorithm of Hopcroft and Tarjan.

Another module, GB_GAMES, generates graphs based on college football scores. All the games from the 1990 season between America's leading 120 teams are recorded in **games.dat**; this data leads to "cliquey" graphs, because most of the teams belong to leagues and they play every other team in their league. The overall graph is, however, connected. A demonstration module called FOOTBALL finds long chains of scores, to prove for instance that Stanford might have trounced Harvard by more than 2000 point if the two teams had met---because Stanford beat Notre Dame by 5, and Notre Dame beat Air Force by 30, and Air Force beat Hawaii by 24, and..., and Yale beat Harvard by 15. (Conversely, a similar "proof" also ranks Harvard over Stanford by more than 2000 points.) No good algorithm is

known for finding the optimum solution to problems like this, so the data provides an opportunity for researchers to exhibit better and better solutions with better and better techniques as algorithmic progress is made.

The GB_ECON module generates directed graphs based on the flow of money between industries in the US economy. A variety of graphs can be obtained, as the economy can be divided into any number of sectors from 2 to 79 in this model. A demonstration program ECON_ORDER attempts to rank the sectors in order from “suppliers” to “consumers,” namely to permute rows and columns of a matrix so as to minimize the sum of entries above the diagonal. A reasonably efficient algorithm for this problem is known, but it is very complicated. Two heuristics are implemented for this problem is known, but it is very complicated. Two heuristics are implemented for comparison, one “greedy” and the other “cautious”. Greed appears to be victorious, at least in the economic sphere.

The highway mileage between 128 North American cities appears in **miles.dat**, and the GB_MILES module generates a variety of graphs from it. Of special interest is a demonstration module called MILES_SPAN, which computes the minimum spanning trees of graphs output by GB_MILES. Four algorithms for minimum spanning trees are implemented and compared, including some that are theoretically appealing but do not seem to fare so well in practice. An approach to comparison of algorithms called “mem counting” is shown in this demonstration to be an easily implemented machine-independent measure of efficiency that gives a reasonably fair competing techniques.

A generator module called GB_RAMAN produces “Ramanujan graphs” which are important because of their role as expander graphs, useful for communication. A demonstration module called GIRTH computes the shortest circuit and the diameter of Ramanujan graphs. Notice that some graphs, like those produced by GB_BASIC or GB_RAMAN, have a rigid mathematical structure; others, like those produced by GB_ROGET or GB_MILES, are more “organic” in nature. It is interesting and important to test algorithms on both kinds of graphs, in order to see if there is any significant difference in performance.

A generator module called GB_GATES produces graphs of logic circuits. One such family of graphs is equivalent to a simple RISC chip, a programmable microcomputer with a variable number of registers. Using such a “meta-network” of gates, algorithms for design automation can be tested for a range of varying parameters. A demonstration module TAKE_RISC simulates the execution of the chip on a sample program. Another meta-network of gates will perform parallel multiplication of m -bit numbers by n -bit numbers or by an n -bit constant; the MULTIPLY module demonstrates these circuits.

Planar graphs are generated by GB_PLANE, which includes among other things an implementation of the best currently known algorithm for Delaunay triangulation.

Pixel data can lead to interesting bipartite graphs. Leonardo’s *Gioconda* is represented by **lisa.dat**, an array of pixels that is converted into graphs of different kinds by GB_LISA. A demonstration routine ASSIGN_LISA solves the assignment problem by choosing one pixel in each row and in each column so that the total brightness of selected pixels is maximized. Although the assignment problem being solved here has no relevance to art criticism or art appreciation, it does have great pedagogical value, because there is probably no better way to understand the characteristics of a large array of numbers than to perceive the array as

an image. A module called GB_SAVE converts GraphBase graphs to and from an ASCII format that readily interfaces with other systems for graph manipulation.

For future information see *The Stanford GraphBase*, published by ACM Press in 1993. The book could also be called "Fun and games with the Stanford GraphBase", because the demonstration programs are great toys to play with. Indeed, the author firmly believes that the best serious work is also good fun. We needn't apologize if we enjoy doing research.