Graphillion

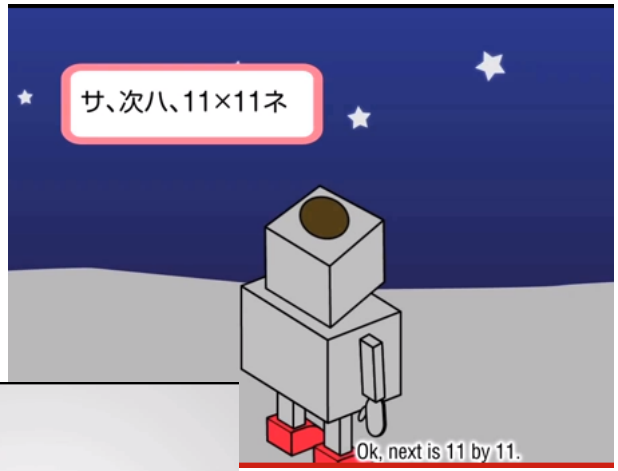
JST ERATO湊プロジェクト 技術員　中元政一

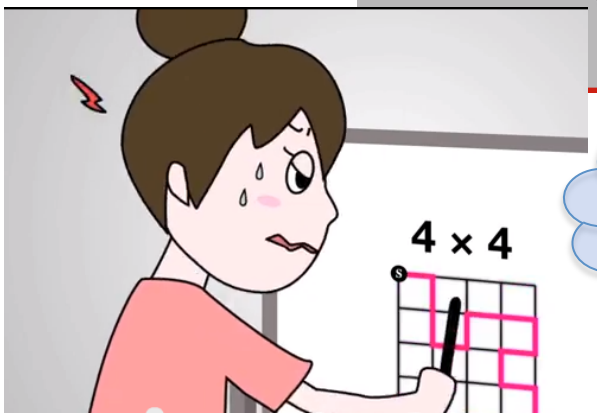
はじめに

- Graphillionを作った人
  - 井上 武
    - 元 JST ERATO湊プロジェクト，現 NTT未来ねっと研究所
  - 岩下 洋哲
    - JST ERATO湊プロジェクト・北大
  - 川原 純
    - 元 JST ERATO湊プロジェクト，現 奈良先端大
  - 湊 真一
    - 北大・JST ERATO
  - その他　JST ERATO湊プロジェクト関係者

はじめに

- Graphillionを作った人
  - 井上 武
    - 元 JST ERATO湊プロジェクト，現 NTT未来ねっと研究所
  - 岩下 洋哲
    - JST ERATO湊プロジェクト・北大
  - 川原 純
    - 元 JST ERATO湊プロジェクト，現 奈良先端大
  - 湊 真一
    - 北大・JST ERATO
  - その他　JST ERATO湊プロジェクト関係者



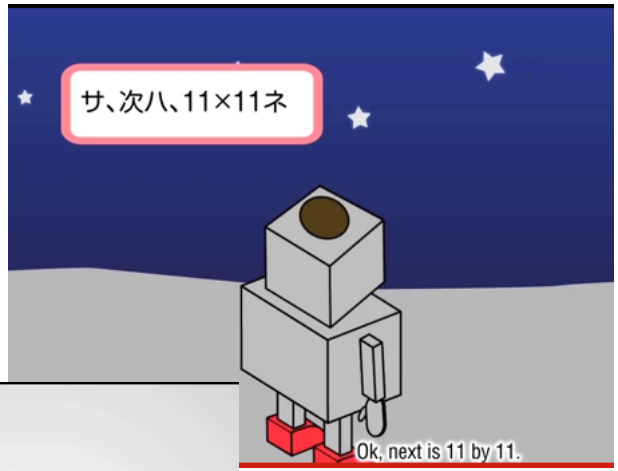


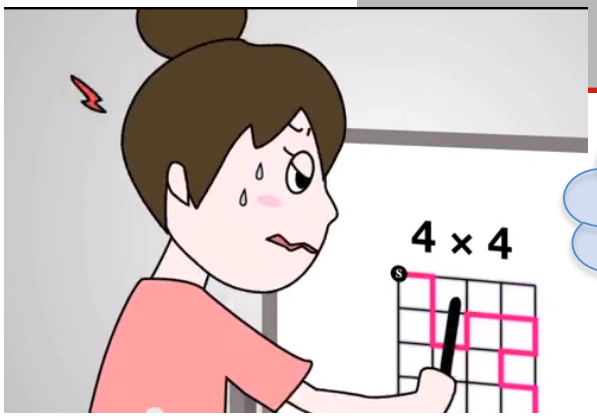
おねえさんの
活躍は
北大博物館で
展示中

- Graphillionを作った理由
  - 数え上げおねえさんを救うため

はじめに

- Graphillionを作った人
  - 井上 武
    - 元 JST ERATO湊プロジェクト，現 NTT未来ねっと研究所
  - 岩下 洋哲
    - JST ERATO湊プロジェクト・北大
  - 川原 純
    - 元 JST ERATO湊プロジェクト，現 奈良先端大
  - 湊 真一
    - 北大・JST ERATO
  - その他　JST ERATO湊プロジェクト関係者



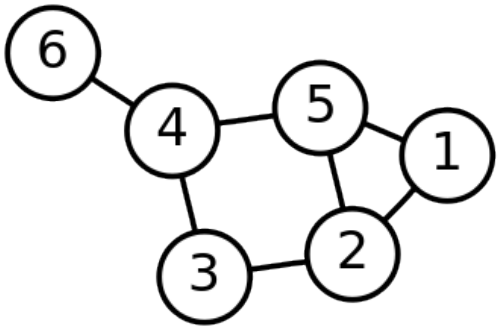


おねえさんの
活躍は
北大博物館で
展示中

- Graphillionを作った理由
  - 数え上げおねえさんを救うため

既存グラフツールの課題

- グラフとは？
  - 頂点ペアをつなぐ辺の集合
  - 様々な問題を表す数学モデル



- 多くのグラフツールが存在する
  - NetworkX, Boost Graph Library, ...
  - ひとつのグラフを対象とする
- 複数のグラフをまとめて操作するツールはない
  - 制約条件を満たすグラフ集合の選択
  - 集合中のグラフに共通する特徴の抽出
  - 最適なグラフの探索，など
- 指数的な数のグラフを効率的に扱えない
  - N辺のグラフから 2^N ものサブグラフが得られる
  - 既存ツールはグラフ数に応じた計算量がかかる

目標とアウトライン

- Graphillion：膨大な数のグラフを簡単に操作するためのライブラリ
  - 性能向上
    - 集合族演算（ZDD），フロンティア法などのアルゴリズムを適用
  - 生産性向上
    - グラフ集合向けの演算体系の提案
    - 科学計算に適した Python の採用

高度なアルゴリズムに適切な抽象を与え，
アルゴリズムの非専門家（応用研究者など）
でも使えるようにしたい

- アウトライン
  - Graphillion のデモと概観
  - グラフ集合と演算
  - Python 拡張モジュールとしての実装
  - Graphillionを利用したアプリケーション(JR近郊区間パス列挙)

Graphillion のデモ

- Graphillion: 数え上げおねえさんを救え / Don't count naively – YouTube

Graphillion の概観

Python

指定した構造を持つグラフ集合を構築

```
# (A) Two graph set objects, which hold trees
#       rooted at vertex u or v, are created

trees_u = GraphSet.trees(root=u)
trees_v = GraphSet.trees(root=v)
```

多様な演算によりグラフ集合を操作

```
# (B) Union of the sets is calculated

trees = trees_u | trees_v
```

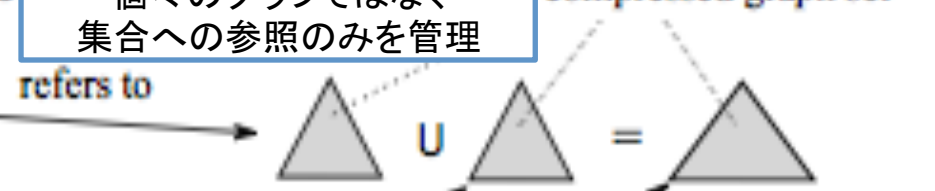
必要なだけの（上位 k 番目までの）グラフを取得

```
# (C) n largest trees are visited through the iterator

i = 0
for tree in trees:
    if i == n: break
    i += 1

    # do something with tree
```

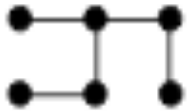
C++

巨大なグラフ集合を
「集合族」として圧縮表現

個々のグラフではなく
集合への参照のみを管理



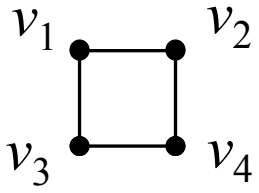
圧縮したまま効率的に演算



全グラフを解凍せずに
必要なグラフのみを抽出

Graphillion が扱うグラフ

- まずユニバースとなるグラフを定義する

$U = (V_u, E_u) =$ 

$V_u = \{v1, v2, v3, v4\}$

$E_u = \{(v1, v2), (v1, v3), (v2, v4), (v3, v4)\}$

- そのサブグラフを対象とする

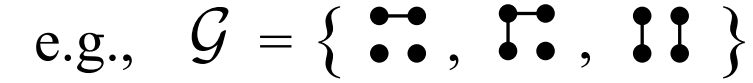
$G = E \subseteq E_u$

e.g., $G = \{(v1, v2), (v1, v3)\} =$ 

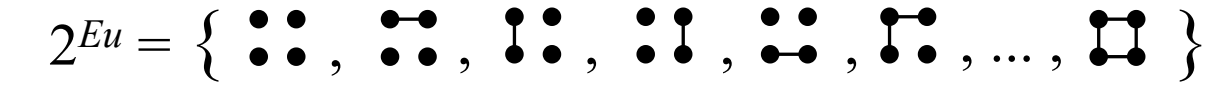
  - グラフを「辺の集合」として表す（頂点は無視する）
  - この簡易化により「集合族演算」を利用可能にする

グラフ集合とその圧縮表現

- 「辺の集合」の集合

  e.g., $\mathcal{G} =$ {  }

- 辺のベキ集合の部分集合

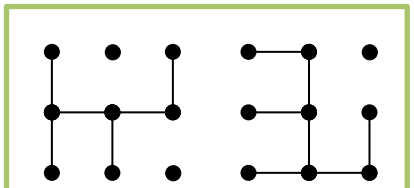
  $2^{Eu} =$ {  }

  $\mathcal{G} \subseteq 2^{Eu}$

  – ベキ集合はユニバースに対して指数的に増加

- 「集合の集合(集合族)」の圧縮表現 ZDD を利用

  – 例:グリッドグラフの隅を根とする木の数と ZDD サイズ



3×3 の木の例

Grid size	Number of trees	Memory of ZDD [Byte]
2×2	10	990
3×3	750	9870
4×4	737354	61830
5×5	8965981766	335190
6×6	1334122533591284	2364750
7×7	2417510626051127173092	18168510
8×8	531403153128266503005306201 74	56321790
9×9	1413043452230406655789221373 1297009012	207115950

10^{37} の木をわずか 207 MB に圧縮

グラフ集合の構築

- フロンティア法
  - 指定した構造のグラフを全列挙するアルゴリズム
    - パラメータにより多様な構造を列挙できる

Structure	Parameters
tree	a root vertex, spanning or not
forest	root vertices, spanning or not
path	terminal vertices, hamilton or not
cycle	hamilton or not
clique	size
connected component	vertices to be connected

  - 動的計画法に基づく効率的な探索
    - さらに，探索範囲を与えられた ZDD に限定できる
  - グラフを圧縮して出力（ZDD に変換可）
    - 計算量は出力グラフ数に依存しない

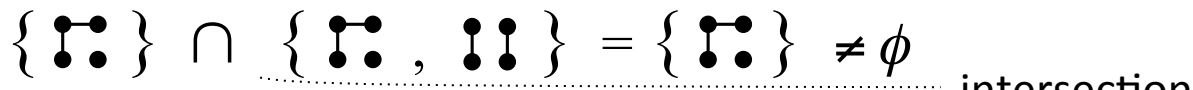
グラフ集合の操作

- 集合族演算
  - ZDD で表現された集合族（集合の集合）の演算体系
    - グラフ集合＝辺の集合の集合
    - 圧縮したまま演算を実行するため高効率
- グラフ集合向けに再構成
  - グラフ集合から条件に合うグラフ部分集合を選択
  - 集合中のグラフをまとめて変更
    - その他，最適な（重みを最大・最小にする）グラフの探索、hitting set, random sampling, 数え上げなど

グラフ集合から条件に合うグラフの選択

- 通常の集合演算（集合「族」ではない）
  - union, intersection, difference など

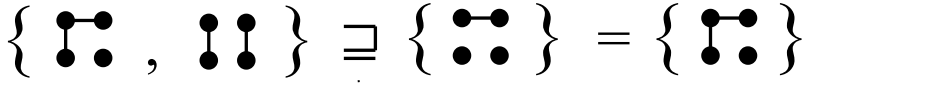
membership query: e.g.,  is found in



intersection

- グラフ構造に基づく選択演算
  - sub/super-graphs, maximal/minimal graphs など

search: e.g., structure  is found in 



Operation	Definition
union	$\mathcal{G}_1 \cup \mathcal{G}_2 = \{G \mid G \in \mathcal{G}_1 \vee G \in \mathcal{G}_2\}$
intersection	$\mathcal{G}_1 \cap \mathcal{G}_2 = \{G \mid G \in \mathcal{G}_1 \wedge G \in \mathcal{G}_2\}$
difference	$\mathcal{G}_1 \setminus \mathcal{G}_2 = \{G \mid G \in \mathcal{G}_1 \wedge G \notin \mathcal{G}_2\}$
symmetric difference	$\mathcal{G}_1 \oplus \mathcal{G}_2 = (\mathcal{G}_1 \setminus \mathcal{G}_2) \cup (\mathcal{G}_2 \setminus \mathcal{G}_1)$
subgraphs	$\mathcal{G}_1 \sqsubseteq \mathcal{G}_2 = \{G_1 \in \mathcal{G}_1 \mid \exists G_2 \in \mathcal{G}_2 (G_1 \subseteq G_2)\}$
supergraphs	$\mathcal{G}_1 \sqsupseteq \mathcal{G}_2 = \{G_1 \in \mathcal{G}_1 \mid \exists G_2 \in \mathcal{G}_2 (G_1 \supseteq G_2)\}$
maximal graphs	$\mathcal{G}^{\uparrow} = \{G_1 \in \mathcal{G} \mid G_2 \in \mathcal{G} \wedge G_1 \subseteq G_2 \rightarrow G_1 = G_2\}$
minimal graphs	$\mathcal{G}^{\downarrow} = \{G_1 \in \mathcal{G} \mid G_2 \in \mathcal{G} \wedge G_1 \supseteq G_2 \rightarrow G_1 = G_2\}$

集合中のグラフをまとめて変更

- 集合中の全グラフに対し，指定した辺の状態を変更
  - 辺を追加（graft），除去（remove），辺の状態を反転（flip）

graft: e.g., edges  are added to {  ,  }

{  ,  } ⊔  = {  ,  }

Operation	Definition
graft (join ⊔)	$\mathcal{G} \sqcup \{E\} = \{G \cup E \mid G \in \mathcal{G}\}$
remove (meet ⊓)	$\mathcal{G} \sqcap \{E^c\} = \{G \cap E^c \mid G \in \mathcal{G}\}$
flip (delta ⊞)	$\mathcal{G} \boxplus \{E\} = \{G \oplus E \mid G \in \mathcal{G}\}$

集合族演算（join, meet, delta）をグラフ集合向けに応用

Python 拡張モジュールとしての実装

- C++：集合族演算，フロンティア法
  - 計算コストの高い処理を高速化
- Python：プログラミングインタフェース
  - 組み込み set のインタフェースを採用，学習コストを低下
    - len（集合サイズ），in（メンバシップ），for（イテレータ），union など（集合演算）
  - グラフならではのメソッドを追加
    - sub/super-graphs, graft, 重みイテレータ（最適化）など
  - グラフ集合への参照のみを管理
    - オーバヘッドは集合サイズ（グラフ数）に非依存
  - 任意のグラフオブジェクト（NetworkXなど）を利用可
  - Python の豊富なモジュール（NumPyなど）と併用可

Graphillionを利用したアプリケーション

Graphillionを利用したアプリケーション (JR近郊区間パス列挙:Ekillion)

- バックエンドで Graphillionを利用
- 120円(130円)で行くJR大回り旅の検索エンジンとしての利用を想定。
- 大阪、東京、福岡、新潟近郊区間が対象。
- 北海道は72年当時の路線図(少し厳密性に欠ける)を利用(近郊区間ではない)
- 駅弁販売駅をできるだけ多く通るルート列挙など「なんちゃって検索」も可能。
- 関西サテライトラボで開発
- アマゾン上でβ公開中

JR大回り旅とは

- 旅客営業規則第157条2項

  大都市近郊区間内相互発着の普通乗車券及び普通回数乗車券(併用となるも のを含む。)を所持する旅客は、その区間内においては、その乗車券の券面に表示された経路にかかわらず、同区間内の他の経路を選択して乗車することができる。

  – 一筆書きであれば、どれだけ遠回りしても料金は最短経路で計算される!

- 近年は、この「大回り」を紹介した旅行ガイドが発売されるほど よく知られた“遊び”

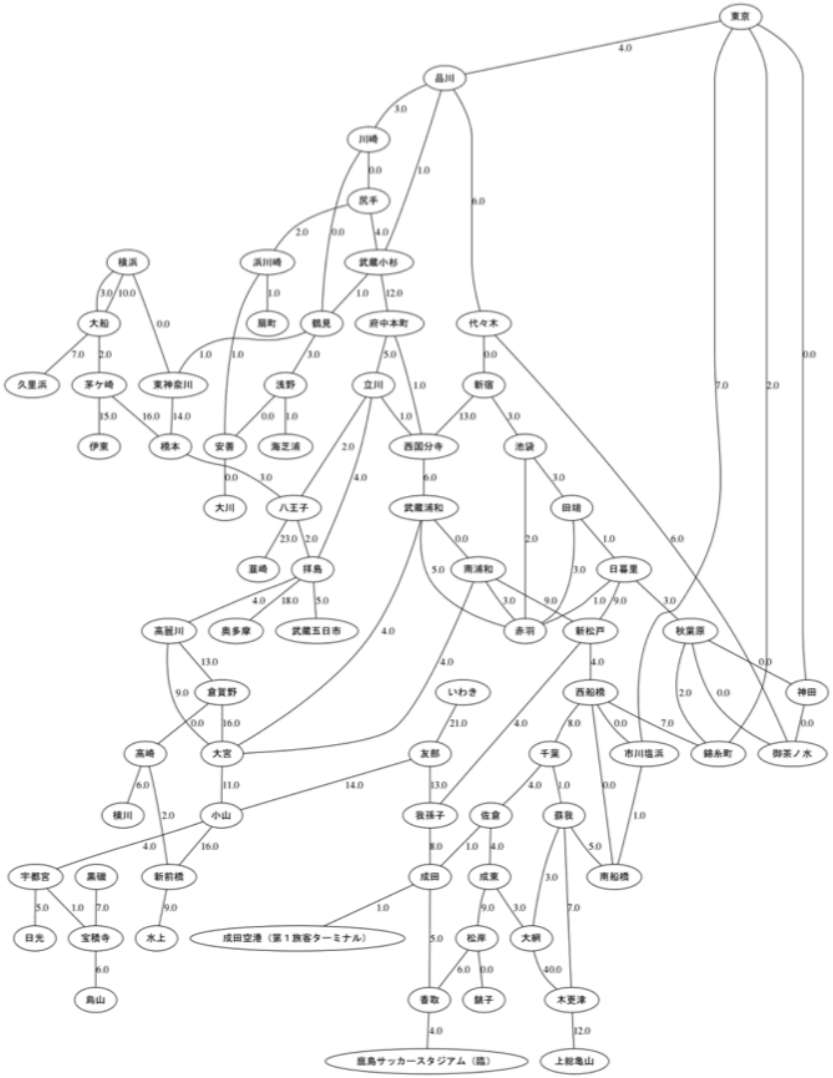
- 東京近郊区間では「800 km 超えの旅も可能」(1,000 km を超えるルートもあるが、1日で乗り切れない)

Ekillionのデモ

- Ekillion

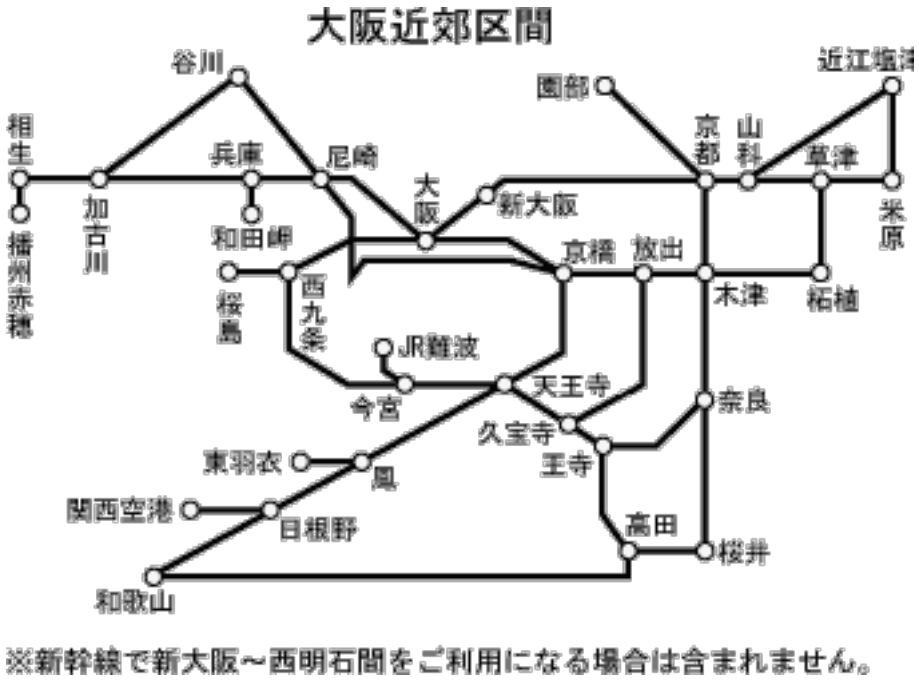
東京近郊区間

※新幹線で東京～熱海間、東京～那須塩原間、東京～高崎間をご利用になる場合は含まれません。



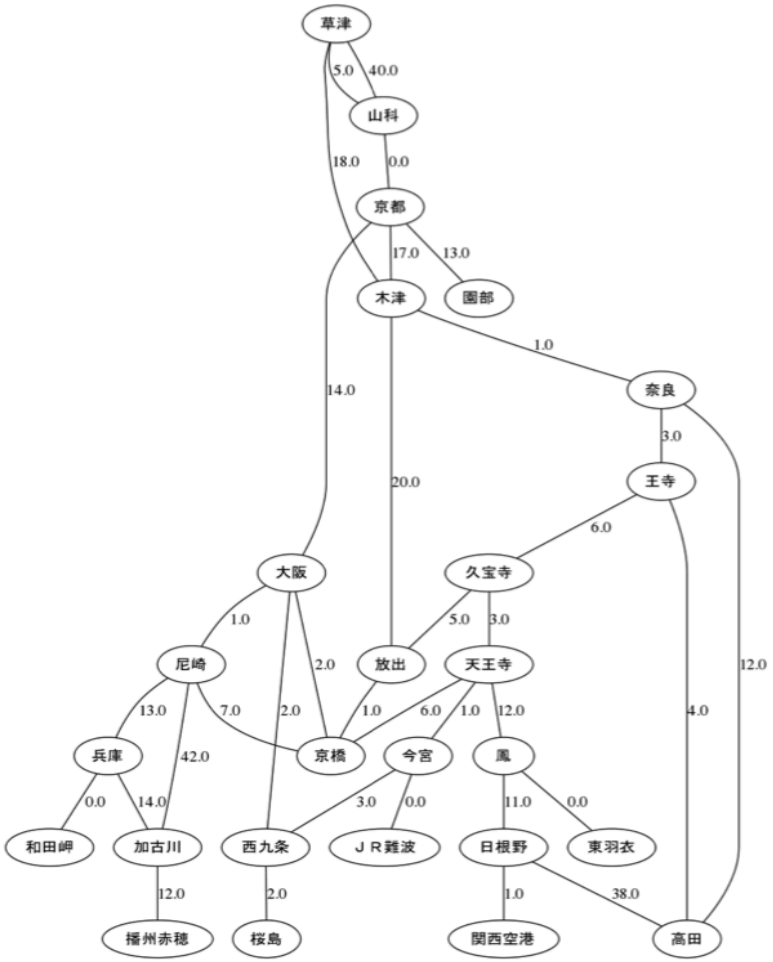
出典: JR東日本 Webサイト
http://www.jreast.co.jp/kippu/1103.html

大阪近郊区間

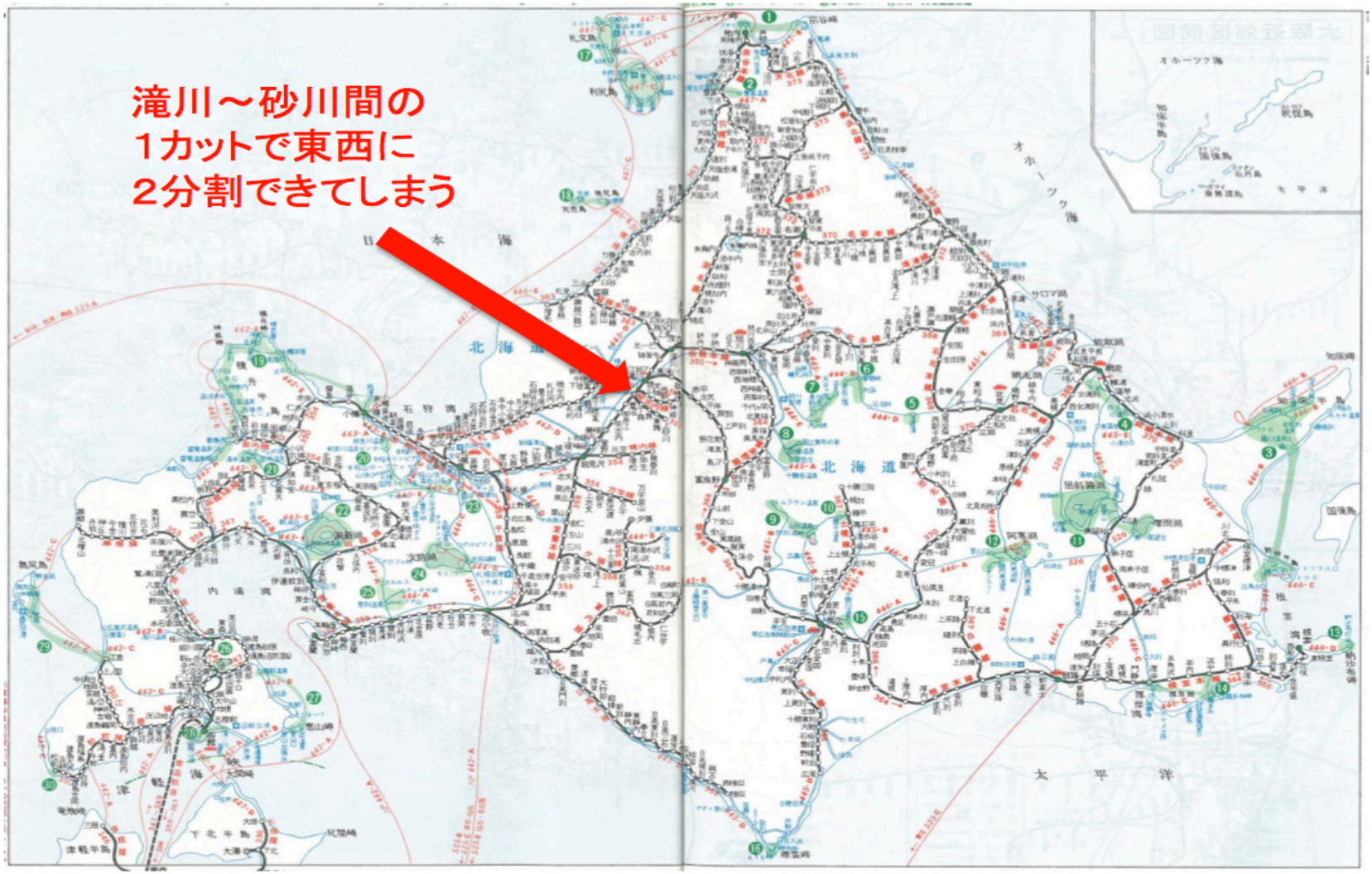


※新幹線で新大阪～西明石間をご利用になる場合は含まれません。

出典: JR東日本 Webサイト
http://www.jreast.co.jp/kippu/1103.html

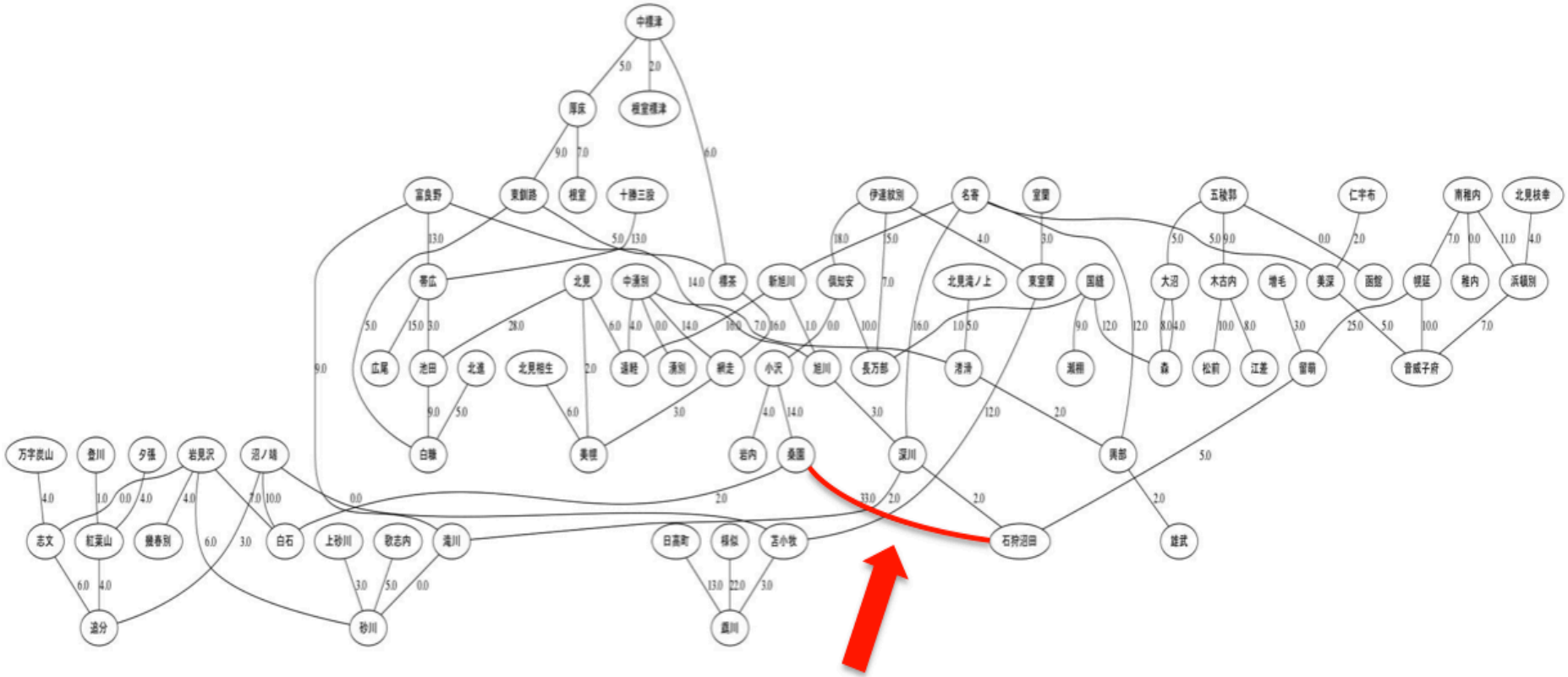


1980 年当時の北海道路線図



出典:『交通公社の時刻表』1980年10月号

北海道路線図グラフ



1972 年に廃線となった 札沼線(学園都市線) 新十津川駅～石狩沼田駅間を 復活させることでトポロジを複雑化してある

路線データの節点数と辺数

地域	全駅路線図			乗換駅路線図		
	節点数	辺数	総距離	節点数	辺数	総距離
大阪	355	364	941.0km	26	34	924.3km
北海道	708	721	3,984km	74	86	3927.7km
東京	623	653	2055.6km	72	101	2037.9km
福岡	128	131	323.0km	19	22	323.0km
新潟	58	59	174.2km	7	7	159.0km

路線図グラフを用いた問題

- 最長路問題
  - 長さが最大のパスを求める
- s-t 最長路問題
  - s, t を端点とする、長さ最大のパスを求める
- 全点対最長路問題
  - すべての s, t の組み合わせに対し、s-t 最長路問題の解を求める
- 最大経路差問題
  - 端点が同じパス P1, P2 で、長さの差 |P1| - |P2| が最大のものを求める
- 最大経路数問題
  - パス数が最も多いs,tの組み合わせを求める

最多ルート駅ペア(隣接駅探索)

乗換駅路線図による探索

	s–t 駅名	ルート数	最長距離(km)	駅数	探索s– t総件数	処理時間	処理時間/件
大阪	京都–大阪	103	514.8	16	34	1.49	0.043
北海道	滝川–砂川	691	2053.4	35	86	4.14	0.048
東京	我孫子–成田	9,427,117	976.4	52	101	15.07	0.149
福岡	桂川–新飯塚	9	157.5	11	22	0.94	0.043
新潟	東三条–吉田	3	108.5	5	7	0.37	0.053

全駅路線図による探索

	s–t 駅名	ルート数	最長距離(km)	駅数	探索s– t総件数	処理時間	処理時間/件
大阪	東淀川–新大阪	103	556.9	208	364	23.55	0.065
北海道	上徳富–北上徳富	691	2059.3	362	720	88.08	0.122
東京	布佐–木下	9,427,117	1007.4	331	653	2715	4.16
福岡	福工大前–九産大前	9	163.4	66	131	6.45	0.049
新潟	田上–矢代田	3	121	41	59	2.57	0.043

ルート数が等しい場合、最長距離が最も長い駅ペアを掲載している。実行環境 amazonAWS: m3.2xlarge(8 cores, 26ECUs, 30GB memory)

最多ルート駅ペア(全対探索)

乗換駅路線図による探索

	s–t 駅名	ルート数	最長距離(km)	駅数	探索s– t総件数	処理時間	処理時間/件
大阪	播州赤穂–近江塩津	392	653.4	20	325	5.73	0.018
北海道	根室標津–松前	5,520	2176.6	44	2701	49.1	0.018
東京	磯子–上総亀山	26,440,720	1121.1	52	2556	143.8	0.0056
福岡	今山–博多南	14	200	11	171	2.82	0.016
新潟	東三条–新発田	4	87.7	5	21	0.32	0.015

全駅路線図による探索

	s–t 駅名	ルート数	最長距離(km)	駅数	探索s– t総件数	処理時間	処理時間/件
大阪	栗東–塚口	392	743.1	283	62790	29.6m	0.028
北海道	五十石–松前	5,520	2192.9	394	250121	231.7m	0.056
東京	未対応	–	–	–	339076	––	
福岡	原町–東水巻	16	200.3	81	8125	2.5m	0.019
新潟	越後石山–長岡	4	134.1	46	1653	0.5m	0.018

ルート数が等しい場合、最長距離が最も長い駅ペアを掲載している。実行環境 amazonAWS: m3.2xlarge(8 cores, 26ECUs, 30GB memory)

まとめ

- Graphillion：
  膨大な数のグラフを簡単に操作するためのライブラリ
  - 性能向上
    - グラフを「辺の集合」とみなし，ZDD 圧縮・集合族演算を適用
    - グラフ列挙アルゴリズム（フロンティア法）によりグラフ構造を解釈
  - 生産性向上
    - グラフ集合向けの演算体系の提案
    - 科学計算に適した Python の採用

  オープンソース　http://graphillion.org
- Ekillion:
  JR近郊区間パス列挙(Graphillionを利用したアプリケーション)
  - β公開中 http://54.249.81.169:8080/eki