「Dijkstra法」
交通ネットワークの均衡分析
P133～P140
（土木学会、平成10年）

平成27年5月8日

理論勉強会2015 #4

交通研　B4　庄司惟

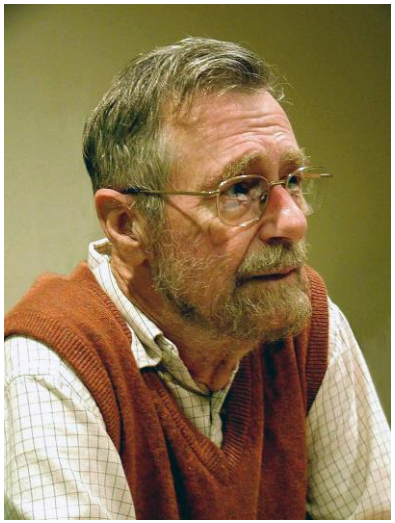
背景

・均衡配分を解くアルゴリズムは、「最小経路探索＝最小所要時間探索」を含む。
（詳しくは再来週の談話会でやります。）

・そのうちのひとつの解法として「Dijkstra法」を本日紹介します。

Dijkstra法（ラベル確定法）

・グラフ上の2頂点間の最短経路を効率的に求めるアルゴリズムで、1959年エドガー・ダイクストラによって考案された。



・カーナビの機能に利用されているとか。

Dijkstra法

＜Dijkstra法の概要＞

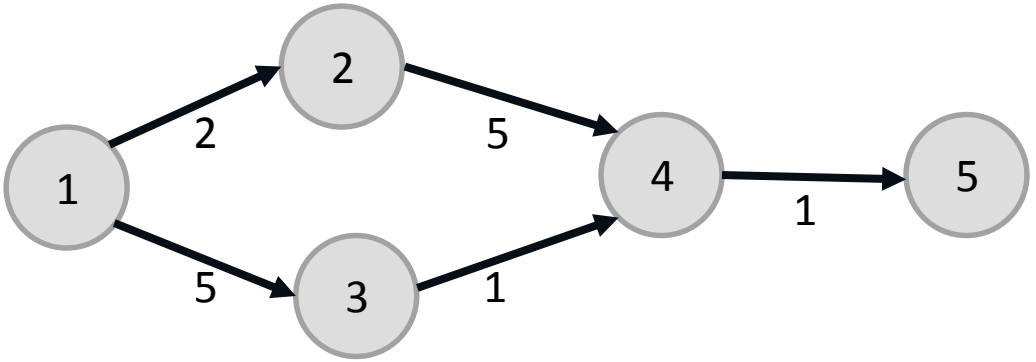
・起点ノードにより近いノードから順に、全方向に向かって、最短経路を列挙していく。

・ひとつの起点からすべての終点までの最短経路と最小交通費用を同時に求める。

※ノード2→ノード6の最短経路がほしい場合は、ノード2を固定して、ノード2からノード2以外への最短経路が全て求めることになる。

Dijkstra法

＜アルゴリズムのイメージ的理解＞
・始点から順にネットワークを作っていく。
（例）



（※重要なのは、ノード3or4の比較の段階。この場合は$t_{12}+t_{24}=2+5=7$が、$t_{13}=5$よりも大きいから、ノード4への最短経路がノード3を含む可能性が出てくる。よってノード4への最短経路決定は保留し、ノード3からのリンクを判断することになる。）

Dijkstra法

＜アルゴリズムの定式化＞

Step1

すべてのノード{j}について部分的最小費用$c_j = \infty$（または十分に大きな値）、$F_j = 0, j \in \bar{K}$とする。

起点をoとする。ノードoに関して、$c_o = 0, i = o$とする。ノードoを集合Kに移す。

Step2

ノードiから出るすべてのリンクの終点ノード{m}について、

$c_m > c_i + t_{im}$ならば$c_m = c_i + t_{im}, F_m = i$とする。

Step3

最小交通費用が確定されていない集合$\bar{K}$に属するすべてのノードについて、次式より部分的最小交通費用の最小値c_jとそのノードをjと計算する。つまり、

$$c_j = \min_p(c_p)\ \{p: p \in \bar{K}\}$$

とする。ノードjを集合Kに移す。

Step4

$c_p = \infty$以外の全てノードが集合Kに移されれば終了。そうでなければi=jとしてStep2へ戻る。

Dijkstra法

上記アルゴリズムを終えると、先行ポインタF_jが得られる。これを利用して最短経路を求めることができる。

F_jの決め方（Step2参照）より、起点oからノードdの最短経路を求める場合は、

d→ $(F_d =)h \rightarrow$ $(F_h =)g \rightarrow \cdots \rightarrow$ $(F_b =)o$

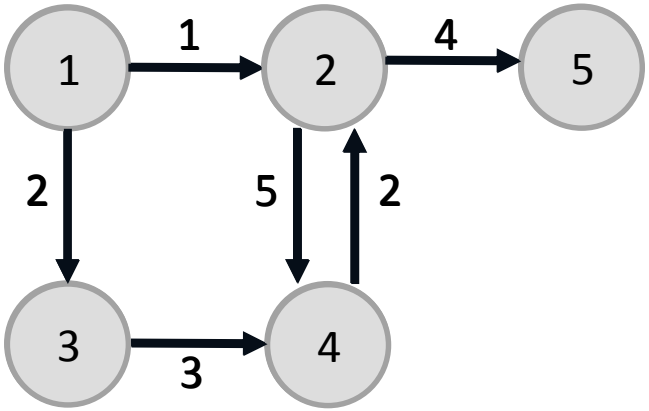
みたいに、逆からたどると、

最短経路o→… →g→h→d

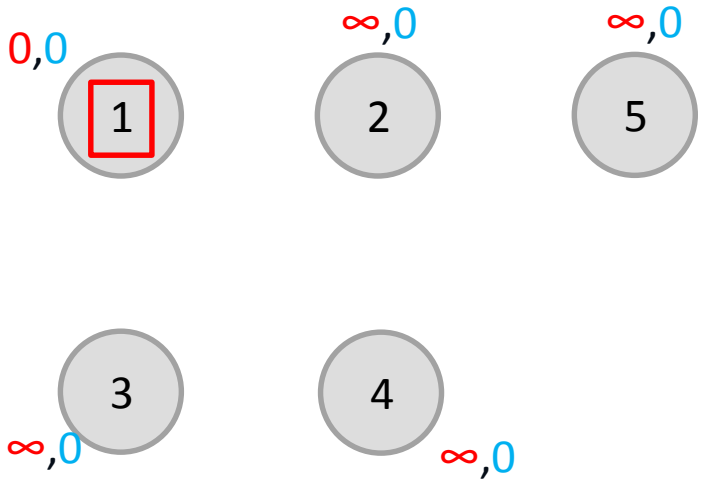
が得られる。

Dijkstra法（計算例）

※始点ノード1として、そこからの最短経路をダイクストラ法で求めます。



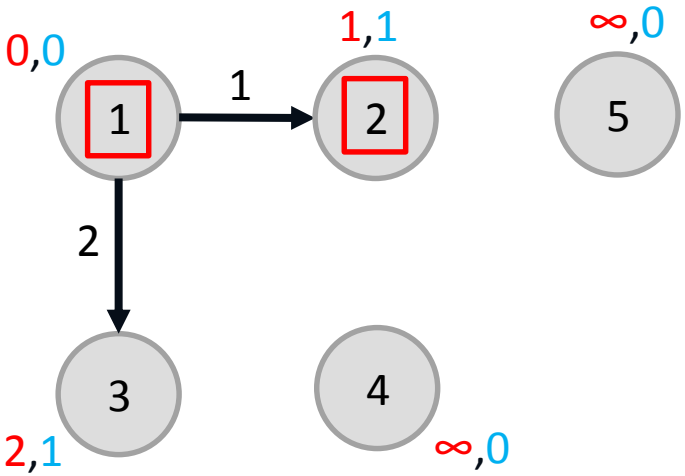
←想定するネットワークとリンクコスト

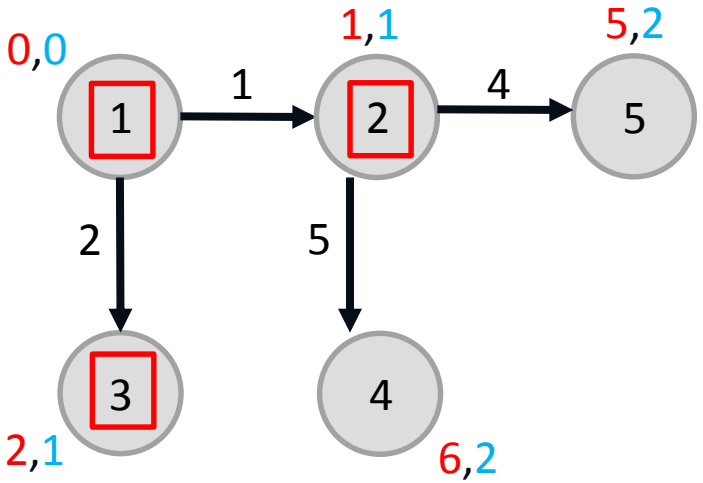


赤文字・・・部分的最小交通費用
青文字・・・先行ポインタ
最短経路が確定したノードは□で囲む

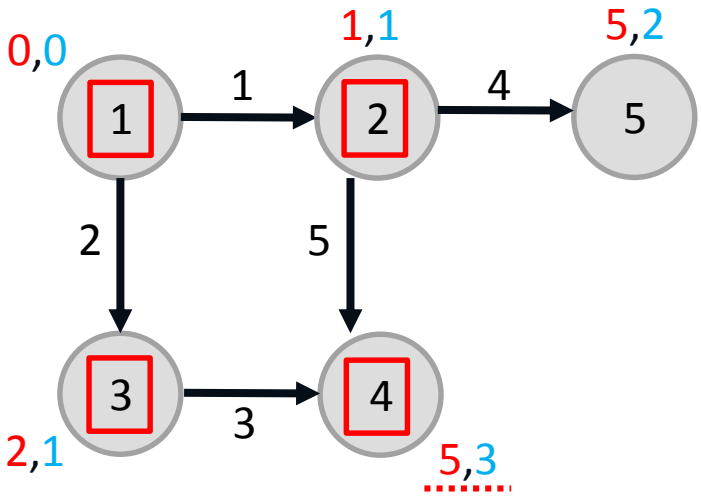
Dijkstra法（計算例）

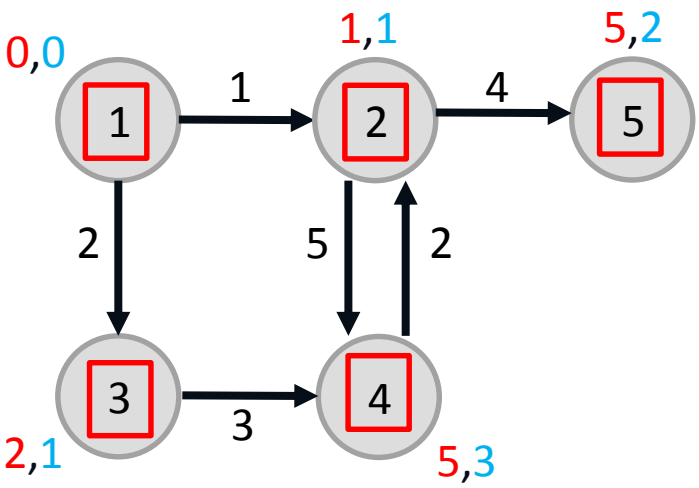
※始点ノード1として、そこからの最短経路をダイクストラ法で求めます。



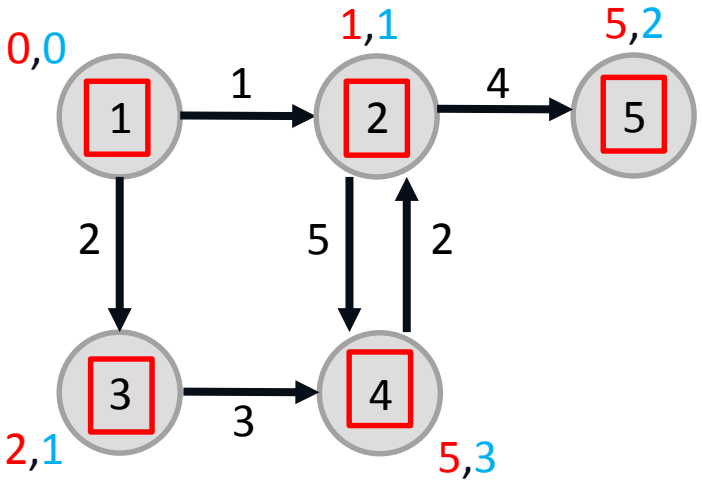


1 2
3 4





Dijkstra法（計算例）



青字（先行ポインタ）を見れば、最短経路がわかる！
つまり、
（1→2の最短経路）2→（F_2 =）1より、1→2
（1→3の最短経路）3→（F_3 =）1より、1→3
（1→4の最短経路）4→（F_4 =）3 →（F_3 =）1より、1→3→4
（1→5の最短経路）5→（F_5 =）2 →（F_2 =）1より、1→2→5

Dijkstra法（復習）

＜アルゴリズムの定式化＞

Step1

すべてのノード{j}について部分的最小費用$c_j = \infty$（または十分に大きな値）、$F_j = 0, j \in \bar{K}$とする。

起点をoとする。ノードoに関して、$c_o = 0, i = o$とする。ノードoを集合Kに移す。

Step2

ノードiから出るすべてのリンクの終点ノード{m}について、

$c_m > c_i + t_{im}$ならば$c_m = c_i + t_{im}, F_m = i$とする。

Step3

最小交通費用が確定されていない集合$\bar{K}$に属するすべてのノードについて、次式より部分的最小交通費用の最小値c_jとそのノードをjと計算する。つまり、

$c_j = \min_p(c_p)\ \{p: p \in \bar{K}\}$

とする。ノードjを集合Kに移す。

Step4

$c_p = \infty$以外の全てノードが集合Kに移されれば終了。そうでなければi=jとしてStep2へ戻る。

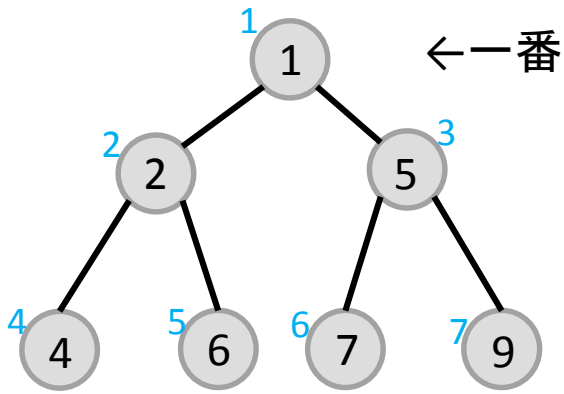
Dijkstra法＋α

・Step3で、「最小ラベル」を求めることになるが、毎回全ラベルのデータをソートするのは手間がかかる。

→ラベルのデータ構造を工夫して、効率化！

「ヒープ構造」（Heap:根付き木）

※特に、２分木ヒープ構造



←一番上のノード（根のノード）が常に最小ラベル！

←上下関係のあるデータ同士では
常に上のデータが小さい値をとる。

※青文字＝配列順位

Dijkstra法＋α

・「データの挿入」時と「最小データの取り除き」時に、Heap構造を維持しつつ並び替えるようにしなければならない。

→これが楽にできれば、Step3は効率化できる。

Dijkstra法＋α

「データの挿入」

最後尾に挿入→その親と比較して、小さければ親と入れ替え、を繰り返す。手間O($\log_2 n$)

「最小データの削除」

根（配列順位1）のデータを削除。続いて最後尾のデータhを根に挿入する。hの子ノード二つのうち小さいほうとhを比較して、hのほうが大きければ入れ替え、を繰り返す。手間O($\log_2 n$)

→結局、少ない手間でヒープ構造を維持できる。
よって最小ラベルの決定を効率化する上で有効なデータ構造であるといえる。

ラベル修正法

・Dijkstra法は「ラベル確定法」とも呼ぶらしい。
・一方で、ラベル修正法もある。
・Dijkstra法のアルゴリズムStep3で、最小のラベルを持つノードを導出する過程があるが、それを回避したい。
・ある規則で作成されるリンクのリスト順にラベリングを繰り返す。

ラベル修正法

<アルゴリズムの定式化>

Step1

すべてのノード{j}について部分的最小費用$c_j = \infty$（または十分に大きな値）、$F_j = 0$ とする。
起点をoとする。ノードoに関して、$c_o = 0, i = o$とする。
{ノードリスト}にノードoを登録しておく。

Step2

{ノードリスト}の先頭にあるノードiを取り出して、iを{ノードリスト}から削除する。

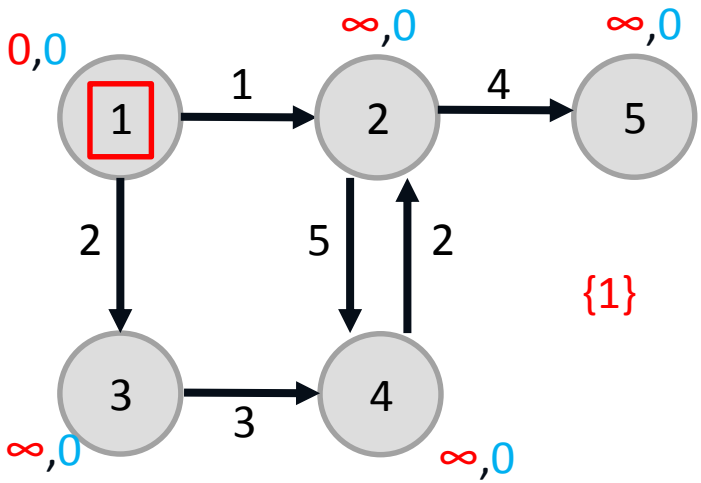
Step3

ノードiから出るすべてのリンクの終点ノード{m}について、
$c_m > c_i + t_{im}$ならば$c_m = c_i + t_{im}, F_m = i$とする。
また、ノードmが{ノードリスト}に含まれていなければmを{ノードリスト}の最後尾に登録する。
（まぁ、ラベル確定法のほうのStep2と同様。）

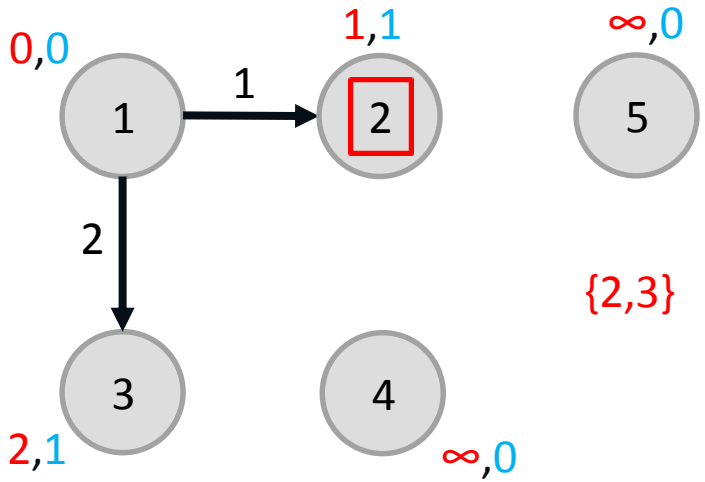
Step4

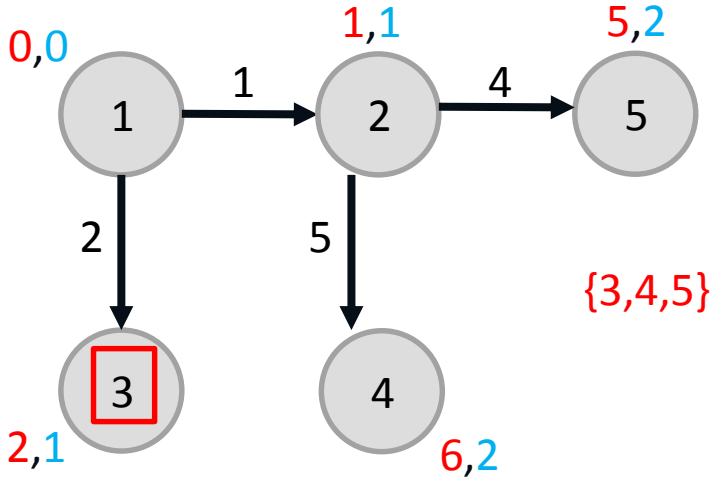
{ノードリスト}が空集合なら終了する。そうでなければStep2へ。

ラベル修正法（計算例）

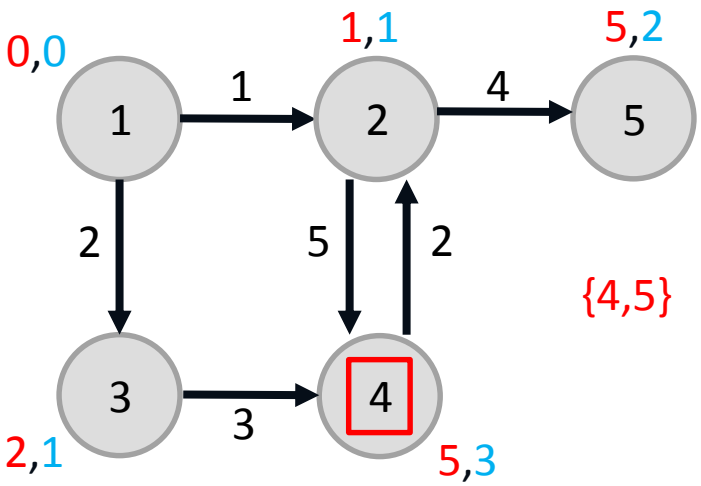


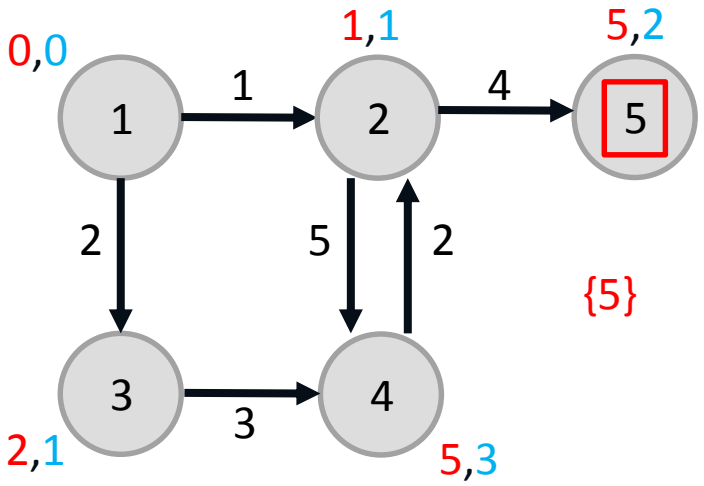
赤文字・・・暫定的な最小交通費用
青文字・・・先行ポインタ
ノードリストの先頭を□で囲む
{ノードリスト}={…}

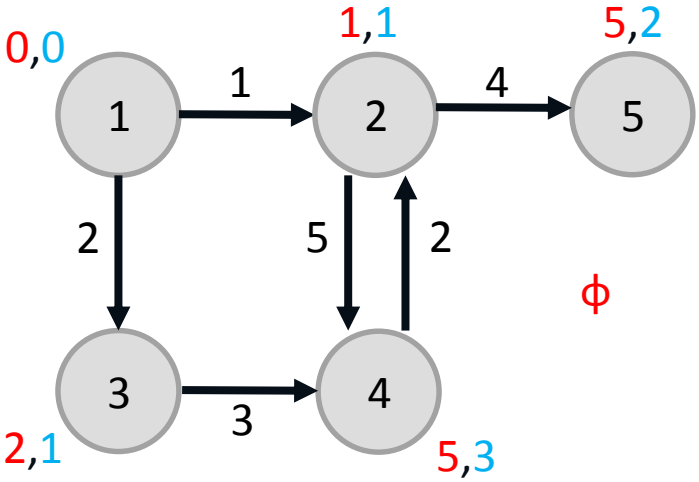




ラベル修正法（計算例）







・先行ポインタと最小費用の双方とも、Dijkstra法で得られたものと完全に一致。

・最短経路は、Dijkstra法と同様に先行ポインタを終点からたどって作成。

まとめ

＜疑問＞

・そもそもDijkstra法の何がすごいの？

→不要な経路であるとわかった時点でそれ以降の計算を省略できるところ。

まとめ

＜まとめ＞

・Dijkstra法（ラベル確定法）

・Heap構造

・ラベル修正法

あそび

・Dijkstra.javaを動かしてみる（wiki参照）
※ノードとリンクを擬似乱数を用いてランダム生成
（ケース１）
ノード100個リンク300本　→　計算時間0.016秒
（ケース２）
ノード1000個リンク3000本　→　計算時間0.063秒
（ケース３）
ノード10000個リンク30000本　→　計算時間0.234秒
（ケース４）
ノード100000個リンク300000本　→　計算時間0.484秒
※最短経路まで126リンク
（ケース5)
ノード1000000個リンク1048574本（エクセルの限界。笑）
→計算できない！ナゼ？
＝ノードに対してリンク本数が少なすぎて、有効なトリップを低確率でしか生成できない。孤立点が多すぎる。
※ノード一個に対してリンク何本とったら有効にまわるか。

あそび

※ノード一個に対してリンク何本とったら経路を結べるか。

「ネットワーク信頼性」＝ネットワークが正常に機能する確率。

総合信頼度（all-terminal reliability)を考察する問題

→NP困難問題！！（多項式時間で解けない）

あそび

つまり

「n個のノードに対して、任意の二点間を結ぶ有向リンクをm本張るとする。このとき、ある二点間がリンクとノードの組み合わせにより正常に結ばれる確率はどのくらいだろうか。但し、リンクの張り方は、必ず異なる二点間を結ぶ、という条件の下で同様に確からしいとする。」

あそび

Dijkstra.javaでは、triptest.csv（最短経路を求める対象のノード二つのみが書いてある）をインプットして、それに対する最短経路を出力している。

↓

これを、全ノードn個に対して、二つの異なるノードの組み合わせを全て試して、計算不可能なノードの組み合わせ数を求めるように書き直す。

あそび

＜計算結果＞

ノード二つの全組み合わせが、それぞれ適切なリンクとノードで接続可能かどうかを調べる。

（ケース２）ノード1000リンク3000

無効経路数136798(13.69%）計算時間217.593秒

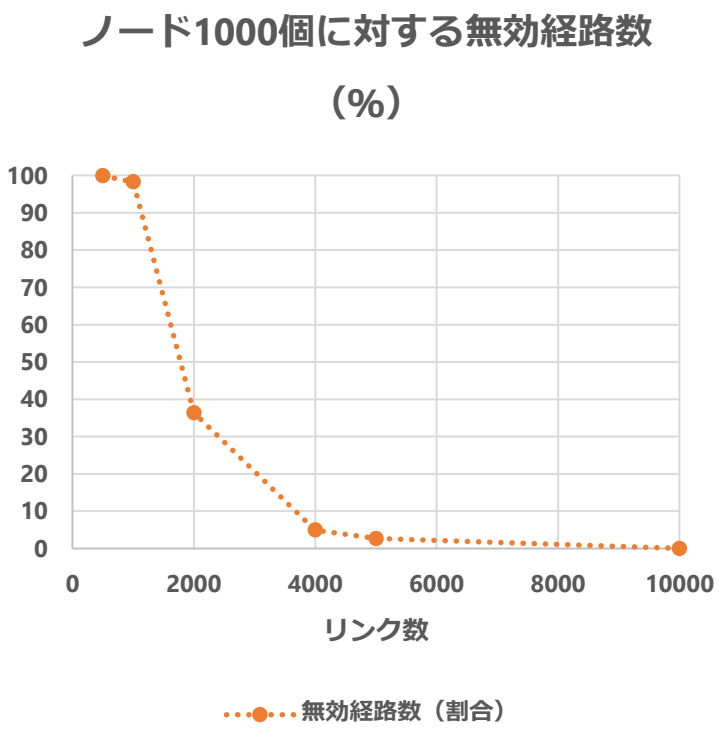
（ケース３）ノード10000リンク30000

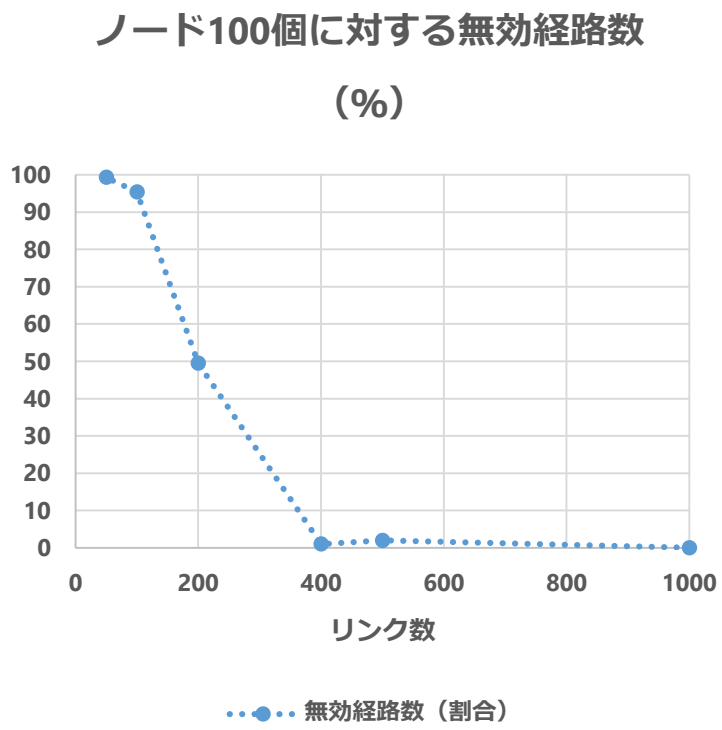
→計算おわらねぇ（推定152時間、今5月5日なので三日後の理論談話会には間に合わない！！）

→もっと早い方法ないんすかね・・・

あそび

ノードに対するリンクの本数を変化させてみる。

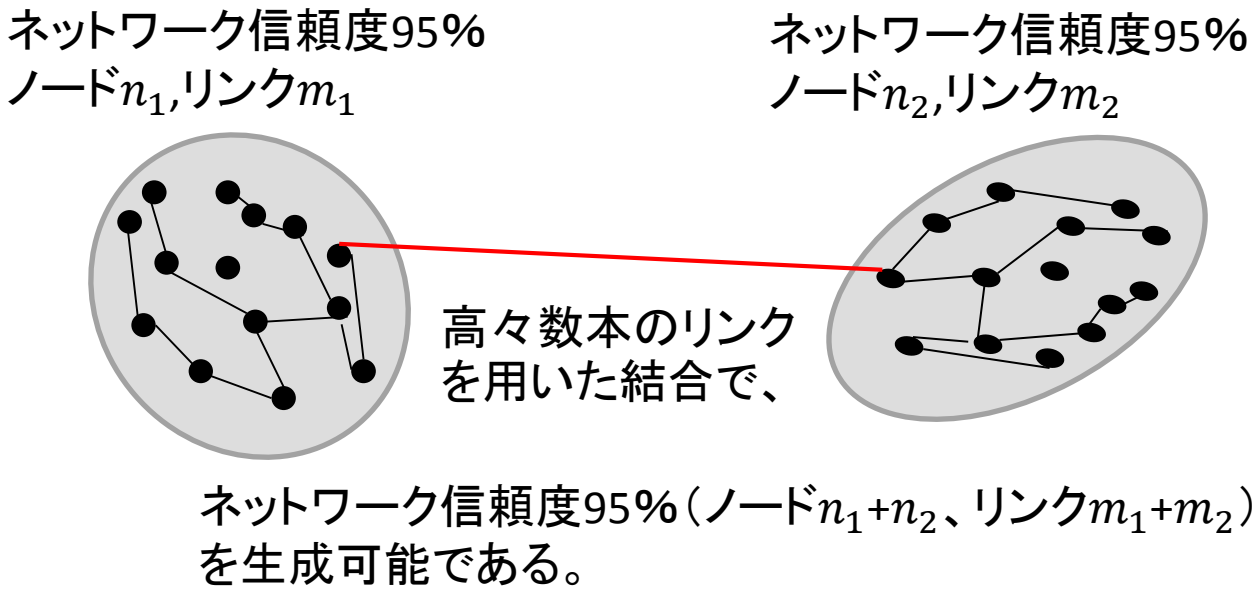




あそび

・ある割合以上のネットワーク信頼度を得るために必要なリンク本数比（=リンク本数/ノード個数）はノード個数によらずある程度一定なのではないか？（適当）

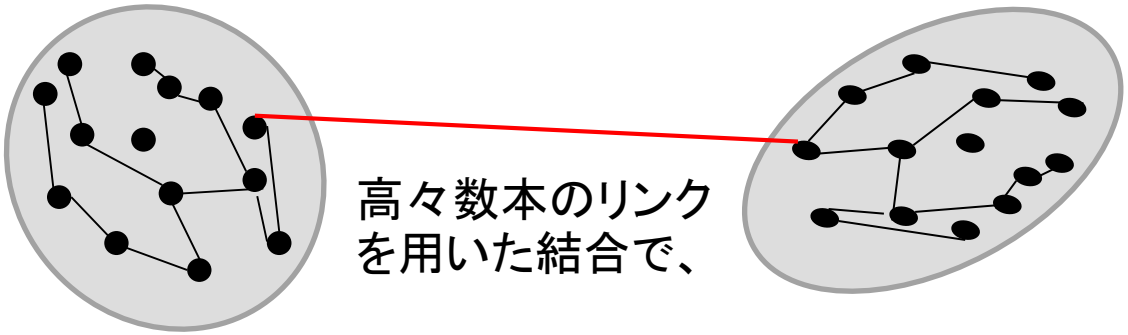
＜イメージ＞



あそび

ネットワーク信頼度95%
ノードn_1,リンクm_1

ネットワーク信頼度95%
ノードn_2,リンクm_2



高々数本のリンク
を用いた結合で、

ネットワーク信頼度95%（ノードn_1+n_2、リンクm_1+m_2）
を生成可能である。

n_1/m_1=n_2/m_2 = kとすれば、
$(n_1+n_2)/(m_1+m_2)=(km_1+km_2)/(m_1+m_2)$ = kだし、
予想はある程度当たってそう。

あそび

＜さいごに…＞

・wikiに改造版（海賊版？）もupしておきますのでどうしようもなく暇だったら見てみてください。

・既にwikiに落ちている有り難いDijkstra法のjavaコードの方には、Heap構造のコードも含まれているので、是非見てみてください。

おつかれさまでした！

後日の質問等は　shoji@trip.t.u-tokyo.ac.jp までお願いします。

勉強会でのはなし

＜本論に関して＞

・ダイクストラ法では、「最小費用」がある値aのとき、そのaを与える全く違う経路によっても新たにaで与えられることがわかった場合は、後者の経路（先行ポインタ）が採用され（更新がおこる。）、前者の経路は結果に出てこなくなってしまう。

・つまり、ある始点終点間の最短経路が実際は二つ以上ある場合でも、求めることができるのは一つのみである。

・すべての最短経路を得たい場合は、現状のDijkstra法のコードを書き替えなくてはならない。

・まぁ、恐らくそんな難しくないです。

・ダイクストラ法は計算量O((ノード数)×(ノード数))

・二分ヒープを用いるとO(リンク数×log(ノード数))

勉強会でのはなし

＜おまけに関して＞

・アルゴリズムでは「計算量」「メモリ消費」に注意すべき。（メモリ消費は近年進化してきてるからあんま問題にはならないが…）

・ネットワーク信頼性には、データの疎密が強く関わっているよね。

・Dijkstra法が最短経路を与えるという保証は、Bellmanの最適性原理でできます。

・ノードに対するリンクの本数で一意に信頼性が決まるわけではなく、「一本のノードから出ているリンクの本数」が強く関わってくるかもね。