# 最適化

## ― ものごとを効率的に行うには ―

宇野 毅明
（国立情報学研究所・情報学プリンシプル系・助教授）
（総合研究大学院大学）

2007年 2月14日 市民講座「8語で談じる情報学」第8回

# 簡単に自己紹介

**名前**: 宇野 毅明

**年齢・職種**: 36歳、助教授

**研究分野**: アルゴリズム理論

- コンピュータプログラムの設計手法の理論
- 速いコンピュータを作ったり、プログラミングの腕を競うのではなく、「設計方法の違いによる性能の向上」を目指す

**最近の研究**: ゲノム情報学やデータマイングで出てくる巨大なデータベースの基礎的(とはいっても非常に時間のかかる)な解析を超高速で行うアルゴリズムの開発

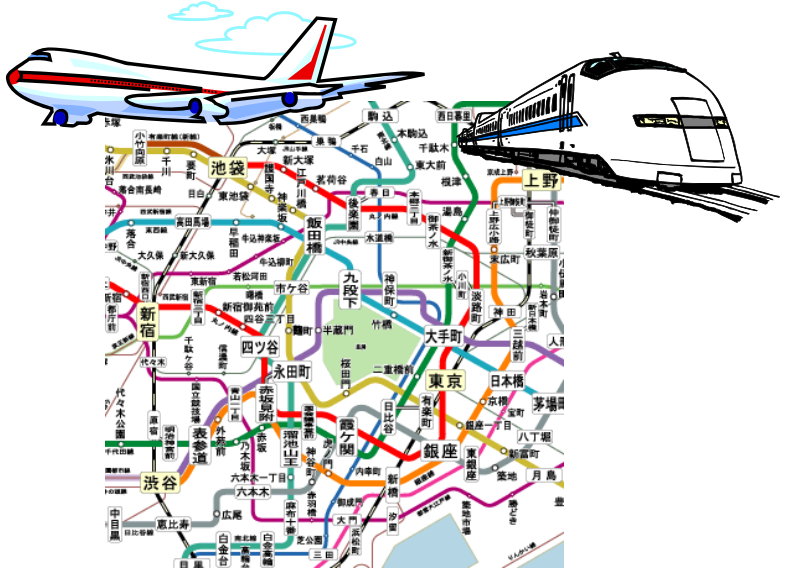
**日課**: 子供と遊ぶこと

# 最適化とは

- 最適化とは「一番いいものを選ぶ」こと
- 「一番いいものを選ぶ」って、どういうことでしょう？

今度、家を買おうと思う
どの家が一番いいかな？

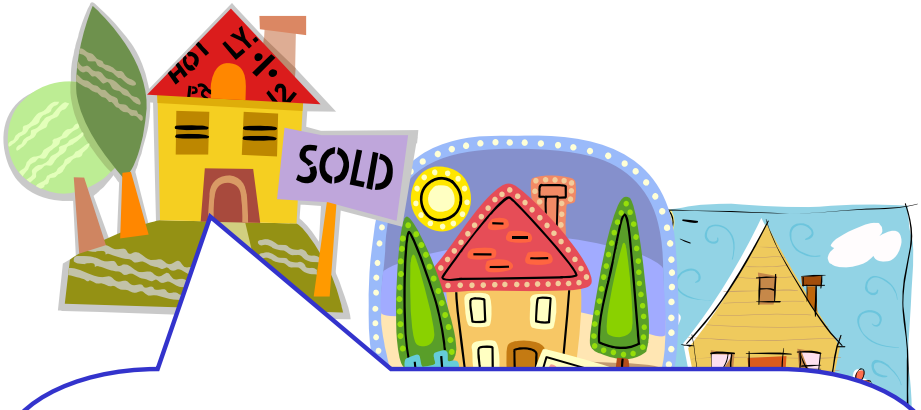
明日大阪に出張だ
どうやって行くのがいいかな


他にも、いろいろな問題が考えられる

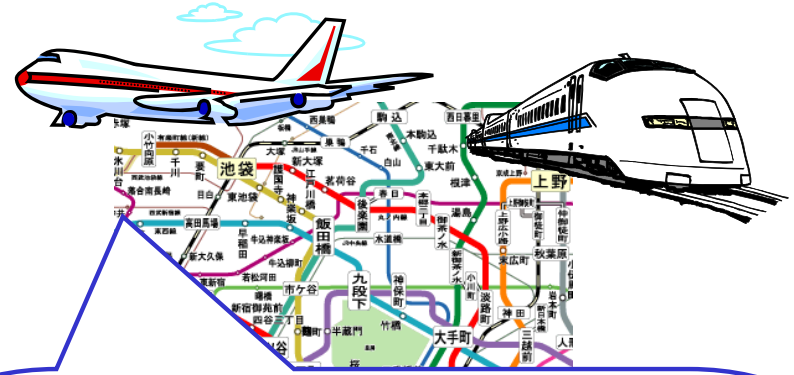
# 何が難しい

今度、家を買おうと思う
どの家が一番いいかな？




- 自分の要求がわからない
  （広い家がいい？
  駅から近いのがいい？）
- 評価するのが難しい
  （すみ易さ, 安全, 雰囲気…）

明日大阪に出張だ
どうやって行くのがいいかな

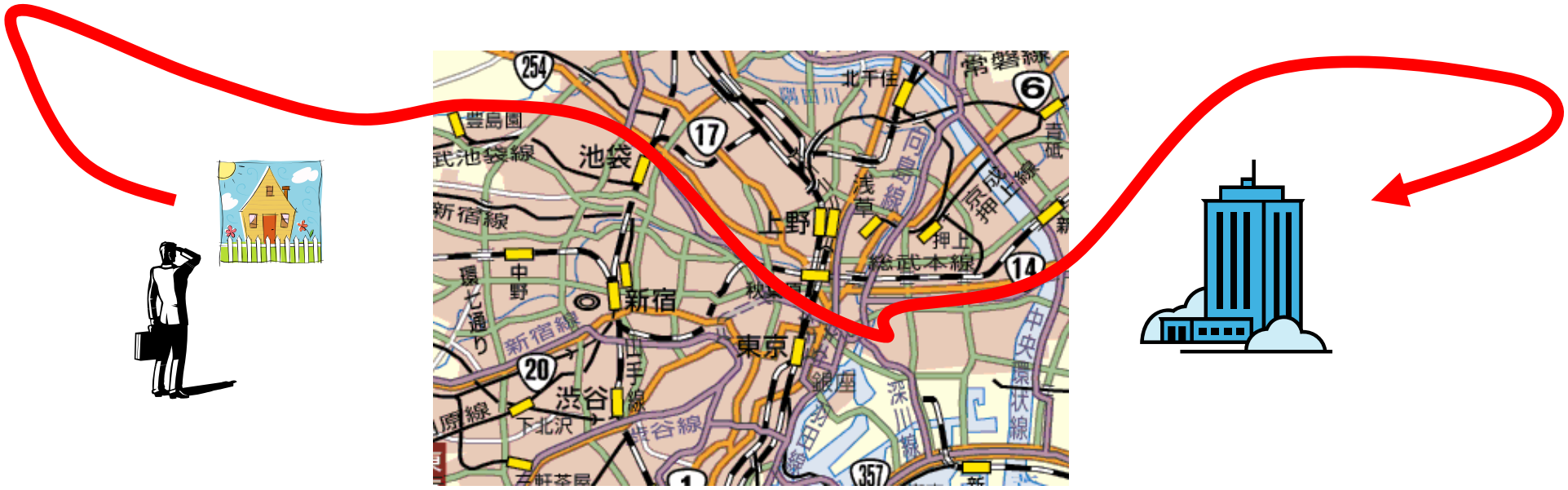



- 要求はわかる。評価も簡単
  （早く到着。安い）
- 選択肢が多いのが大変
  （どの道を通るか、どの交差点を曲がるか．．．）

たくさんの選択肢からいかにいいものを選ぶか、が最適化

# 速いルートを探す

- 目的地まで一番近い（速い）道を探そう
  - ➔ 道の通り方はたくさんある。どの通り方がいい？




- ゴールの方向だけ見ればいい、というわけではない
- 全ての経路をチェックしないとわからない？

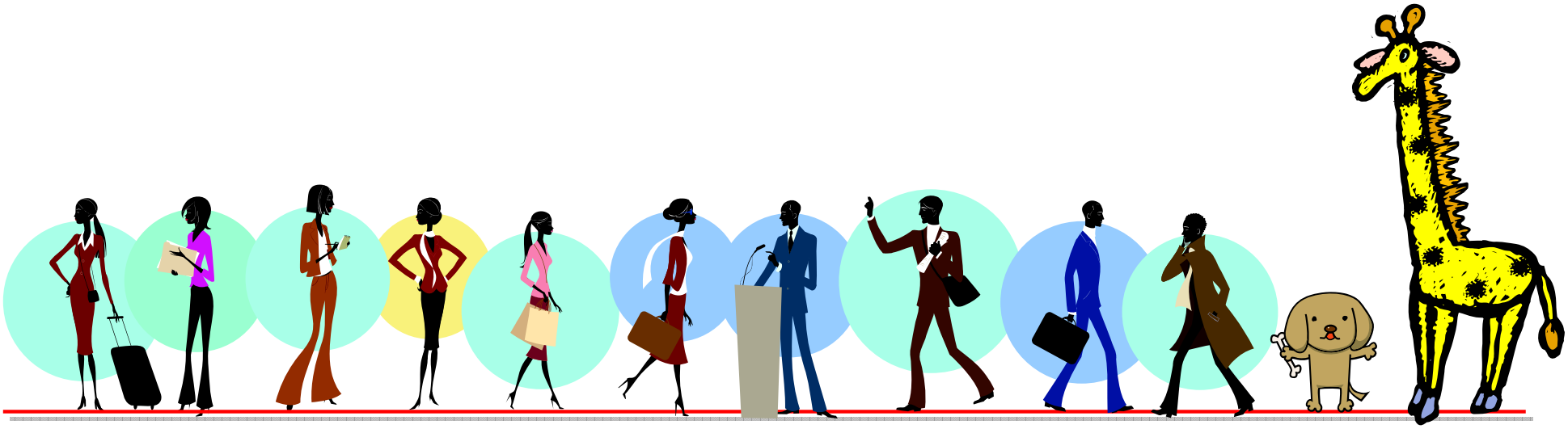
**数多くの選択肢の中からベストを見つける、というのが難しい**

# データは手の中にあるが…

- 他の、選択肢の多い問題として、こんなものを考える

**「世界で一番背が高い人を選ぶ」**

選択肢（候補）がとてつもなく多くて大変だ

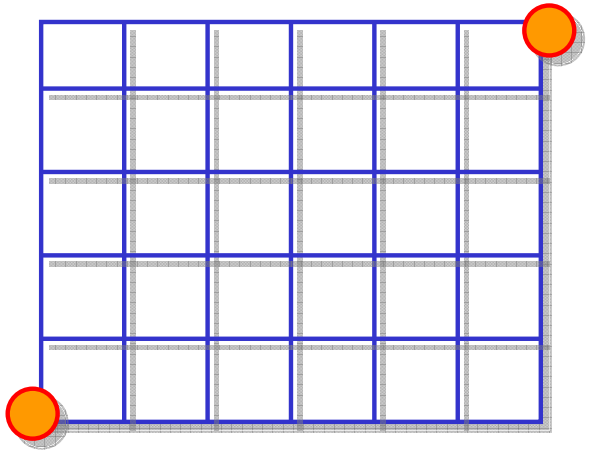


- しかし、この問題は先ほどとは本質的に違う
- この問題は「データを手に入れる」作業が大変なのだが、ルート検索は、道の長さは計算できる

# コンピュータなら

- 「時間はデータから計算できるし、カーナビはすぐ答えを見つけるんだから、コンピュータなら全部調べたって、すぐなんじゃない？」
  - ➔ そんなことはないです

- コンピュータの計算は1秒に1億回程度
- ルートの数は、爆発的に、指数的に増える
  - ➔ 20×20の碁盤の目でも、ルートの数は1000億をゆうに超える



**工夫しないと何もできない**

# 最適化とは

- 物の組合せの中から最も良いものを見つける問題、あるいは、「決めるもの（**変数**）の値をうまく調整して、状態を最も良くする問題」を**最適化問題**という

- 「現実の問題をどうやって最適化問題にするか」（**モデル化**）
「最適化問題をどうやって短時間で解くか」（**解法・アルゴリズム**）
という2つの難しさがある

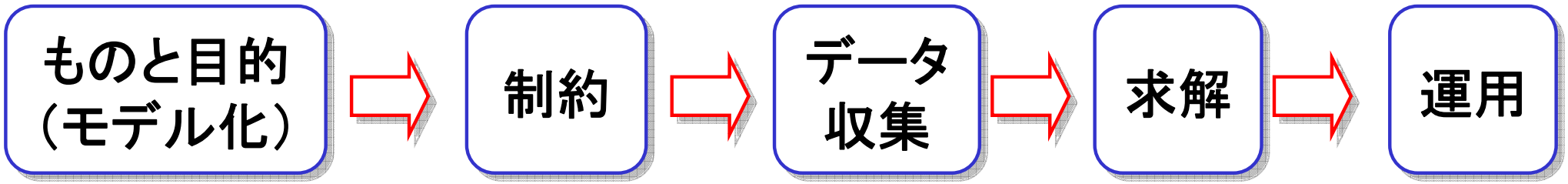
- 「どのように最適化問題にするか」が最適化モデルの研究
- 「どのようにして短時間で解くか」が計算・アルゴリズムの研究
- 「どのような性質を持つか」が数理の研究

**モデルと計算、それぞれを**
**いくつかの問題を例題として、見て行きましょう**

# 最適化を使うには

- 実際に最適化を使うには、どのようにするか？

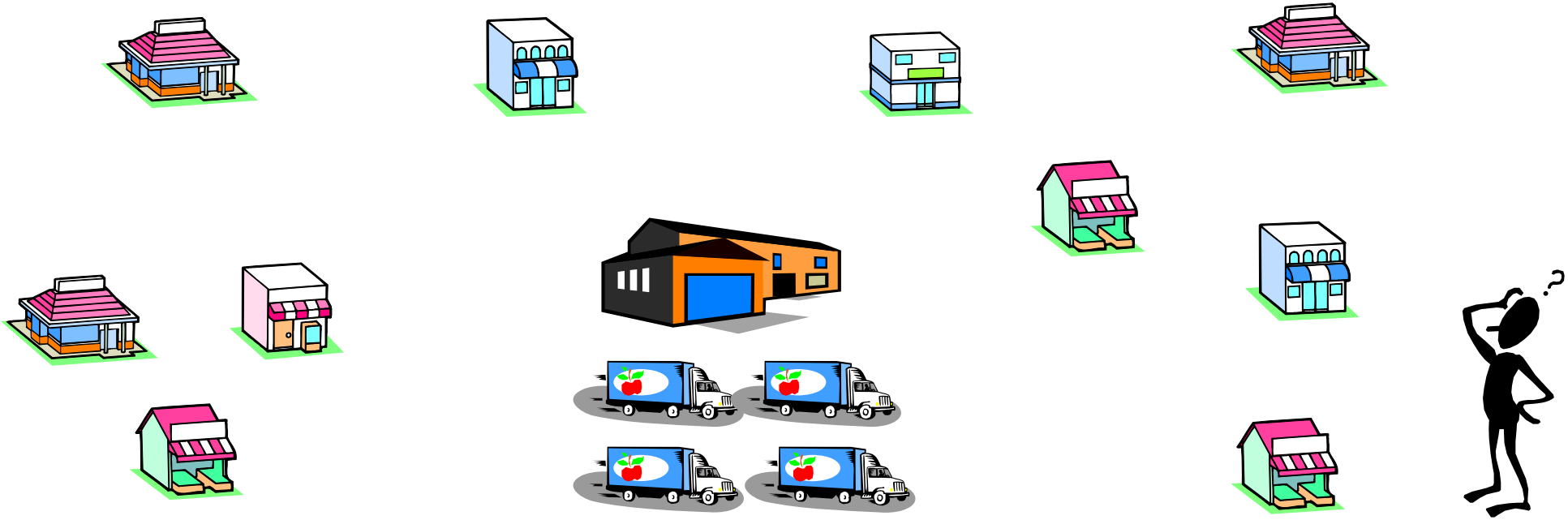
- まず、求めるものと目的を決める（**問題発見**）
  - ➔ 何を見つけるか、何が良いのかがわかる
- 条件・制約を考える ➔ 何ができるかわかる（**モデル化**）
- データを集める
- 最適化 して解を求める（ **解法＆求解**、後で解説）
- 実際に使う　（ **運用**）



# 配送計画

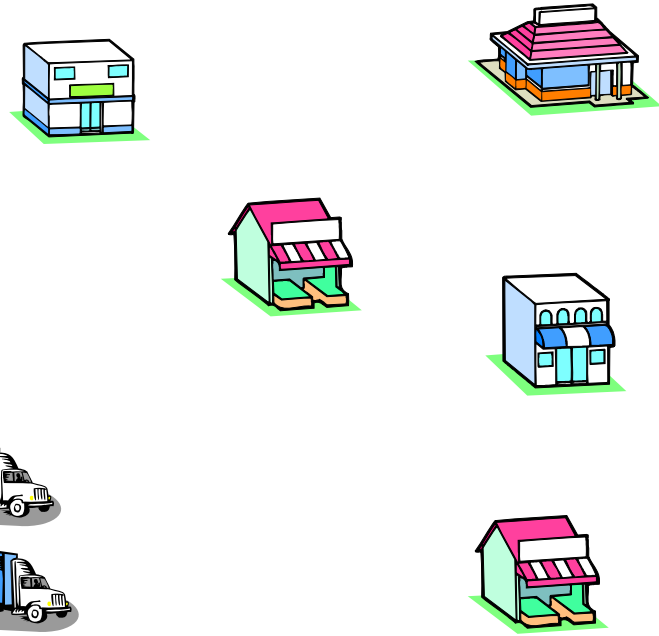
# 配送計画（目的）

- 宅急便の集配所から、トラックでお客に荷物を配送する
- このコストを下げたい



# 配送計画（手段と見つけるもの）

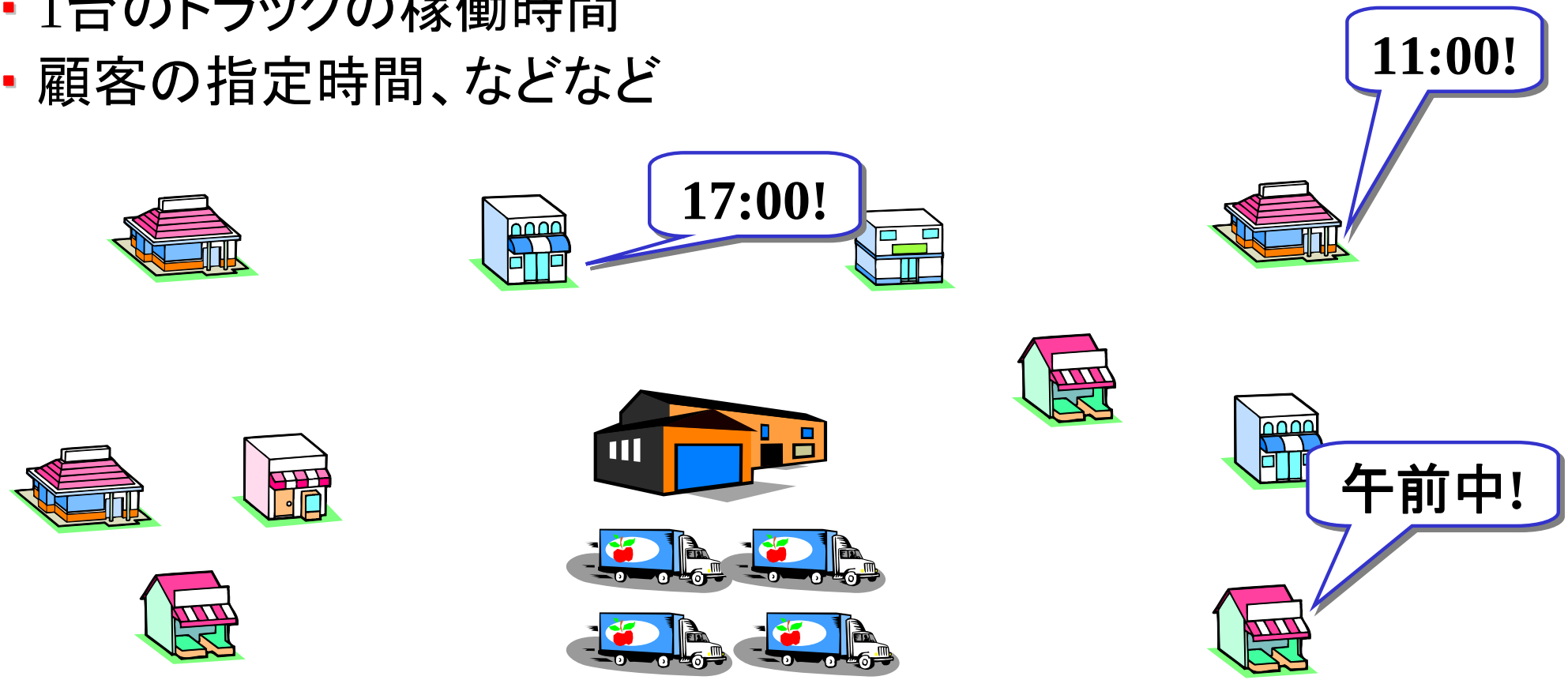
- コストを下げるには何をすればいいだろう？
  - 従業員の給料を下げる。
  - （コストのかかる）注文を断る
  - 安い車・ガソリンを使う
  - 配送拠点を変更
  - 配送ルートを工夫



- 各顧客に行く順番を変えると走行距離が変わるので、
  時間・ガソリンのコストが変わる
  - ➔ 1台のトラックでより多くの配達ができるかも
  - ➔ 他の人の分も配達できるかも
  - ➔ 台数・人員を減らせるかもしれない

# 配送計画（制約）

- 客から客への移動距離は直線距離で概算（またはナビで計算）
- 最大積載量（荷物のつめこみには上限がある）
- 1台のトラックの稼働時間
- 顧客の指定時間、などなど




**見つけるもの・目的・条件が整理できた**
**実際に解くにはどうしたらいいだろう？**

# 配送計画を解く:難しさと要求

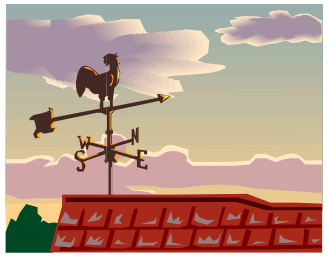
- ルートの候補は非常に多い

(2点間の最短経路がわかるだけではだめ。順番がわからないといけない)

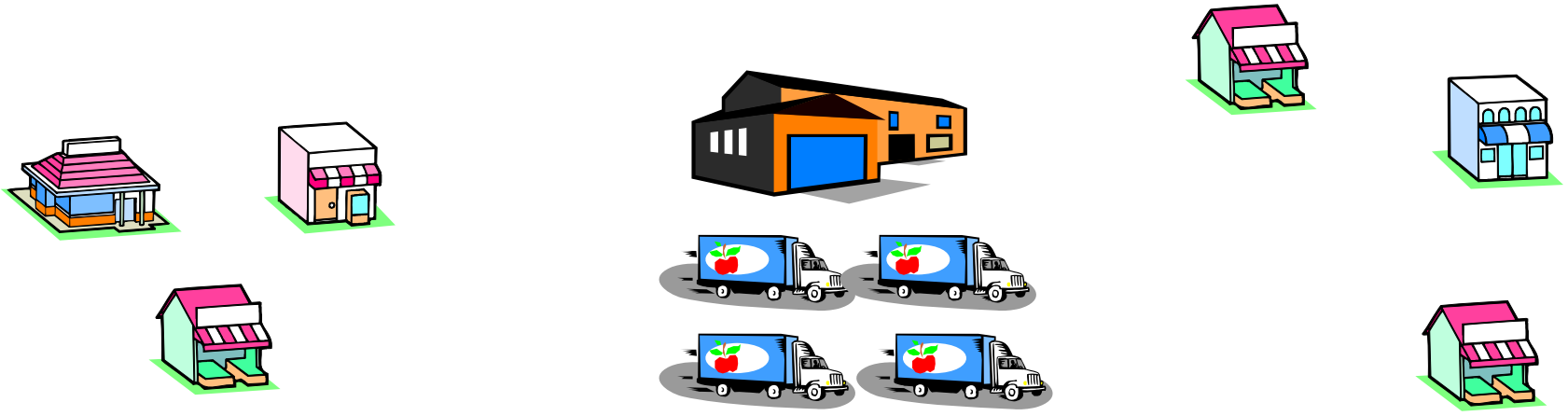
- 1台のトラックが20軒の配達をするルートは、

  20! = 243,2902,0081,7664,0000 通り

- 最短経路を見つけるときのような、良い構造はまだ知られていない

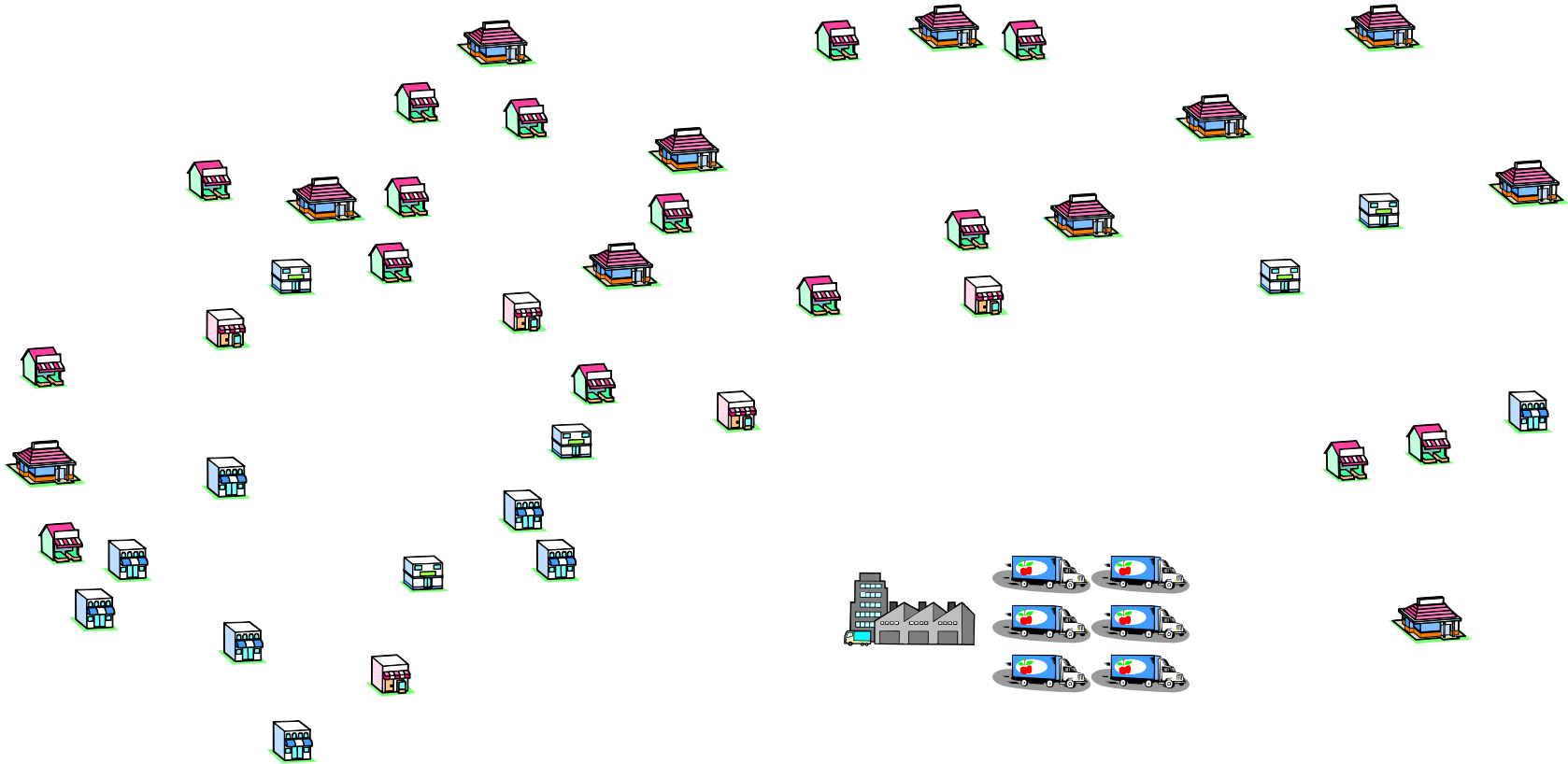


- 誤差が多いので、それほど厳密でなくて良い
- 毎日毎朝解きたいので、速く解けてほしい



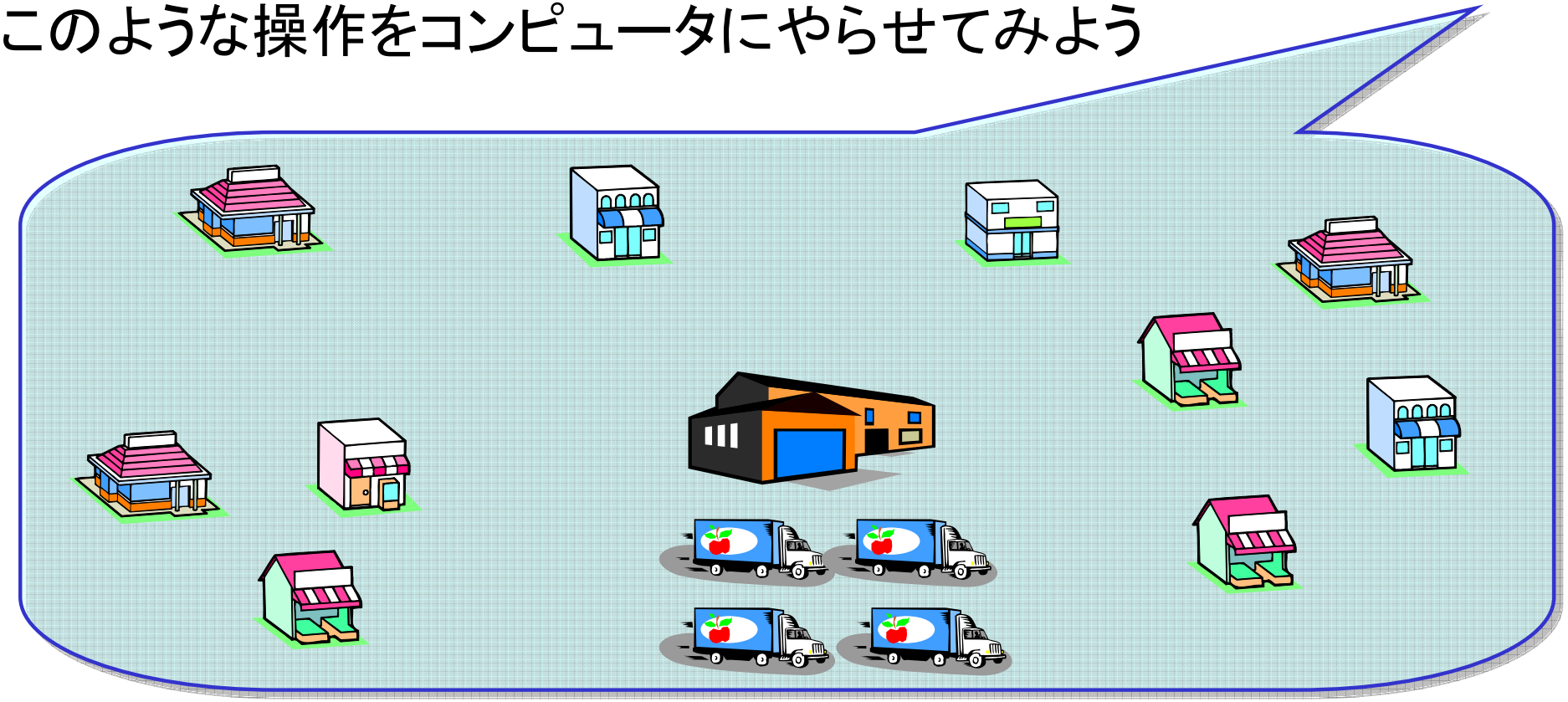
# お客が沢山いると難しい

- 小さい問題は人手で簡単に解けるが、客が増えると、とりあえずのルートを見つけるだけでも大きな手間がかかる



# 人間の作業を真似る

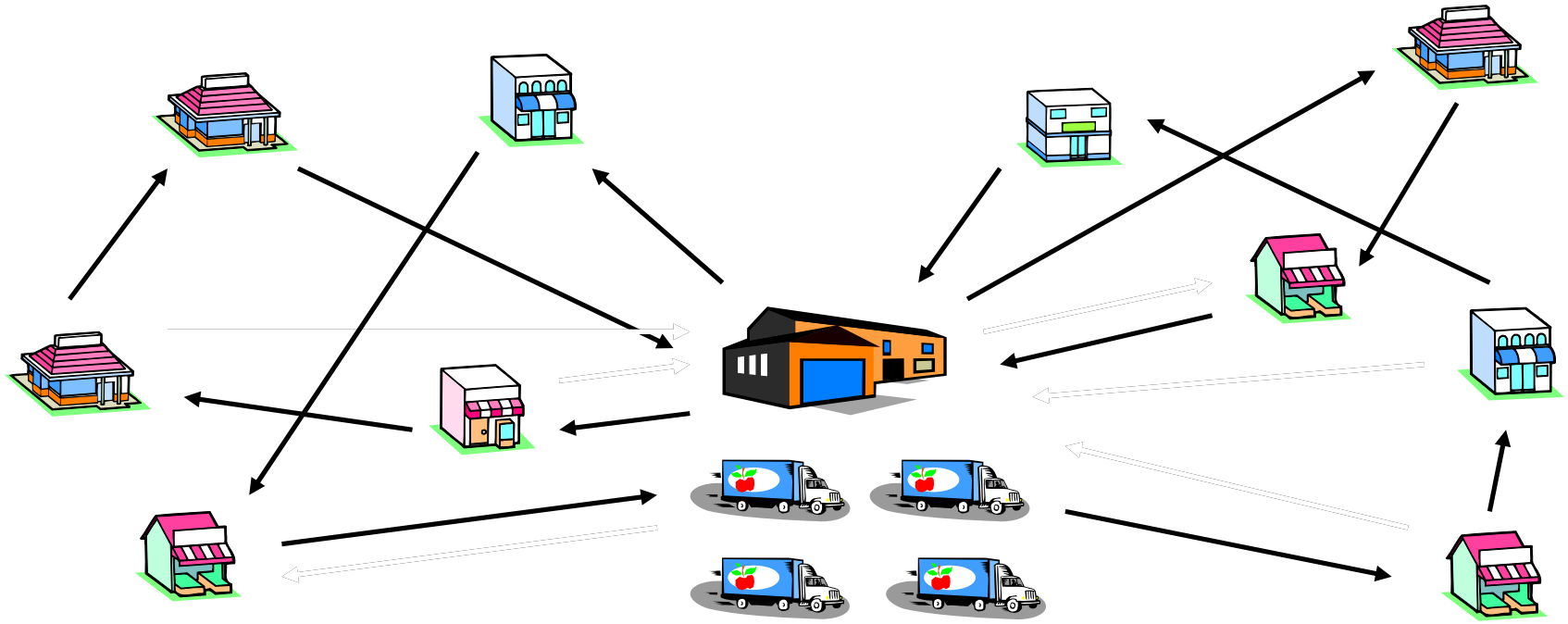
- 人間がこういった問題を解くときは、トライ＆エラーを繰り返す
  - ➔ 近いものから順に選んでみたり、
  - ➔ 1つずつ追加してみたり、
  - ➔ うまく行かない部分をちょこちょこ変更してみたり
- 人間は、じっくりやれば、かなり品質の高い解を得る
- このような操作をコンピュータにやらせてみよう



# セービング法

- 最初、すべてのトラックのルートを空にする
- 顧客を1つずつ、どこかのルートに挿入する

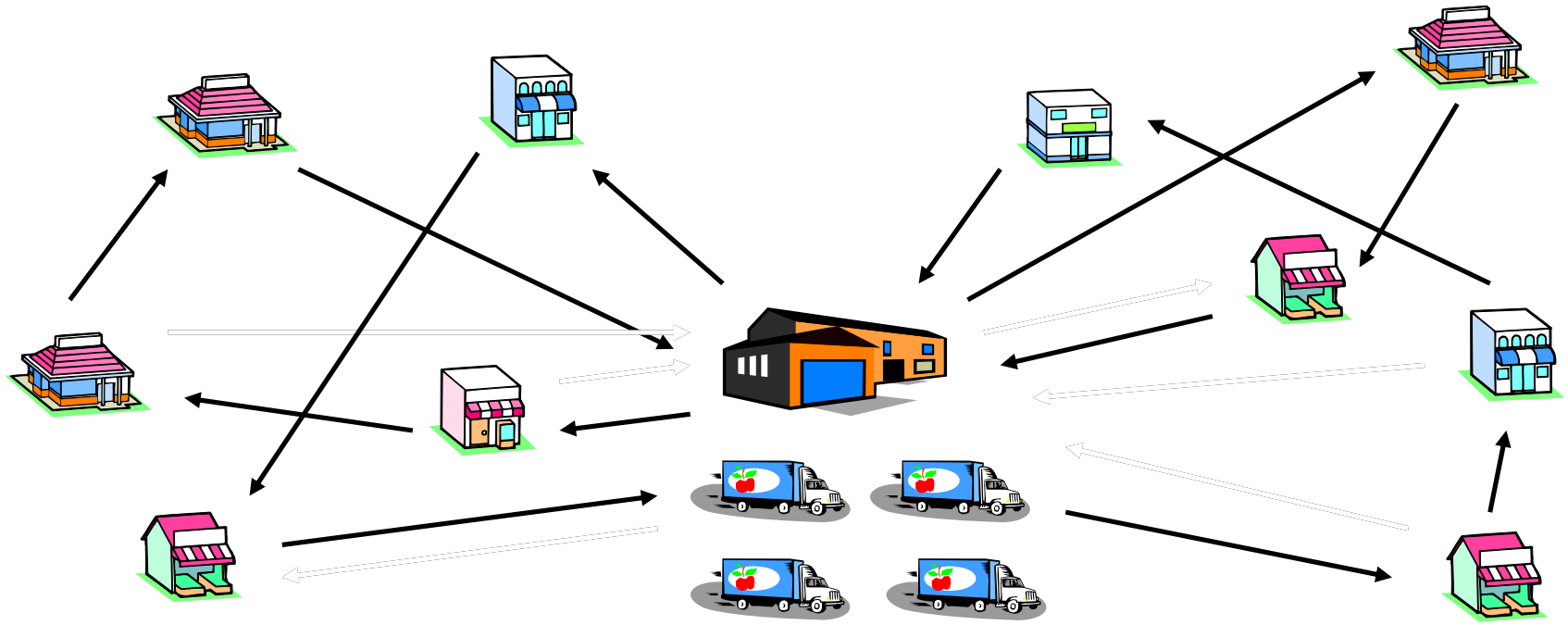
ただし、最大積載量や労働時間の制約を破らず、最も移動距離が増えない場所に挿入する



**平均的に、最も良い解からの誤差が、10-20%くらいになる**

# セービング法の計算時間

- セービング法は、お客を一人ずつ挿入する
- **k**番目の客を入れる場所は、高々 **2k** 通り
- 調べる候補は、全部で高々 $2+4+\ldots+2n \fallingdotseq n^2$ 個



**お客が1万人でも候補は 1億個、これなら解ける**

# どうしてうまくいくのか？

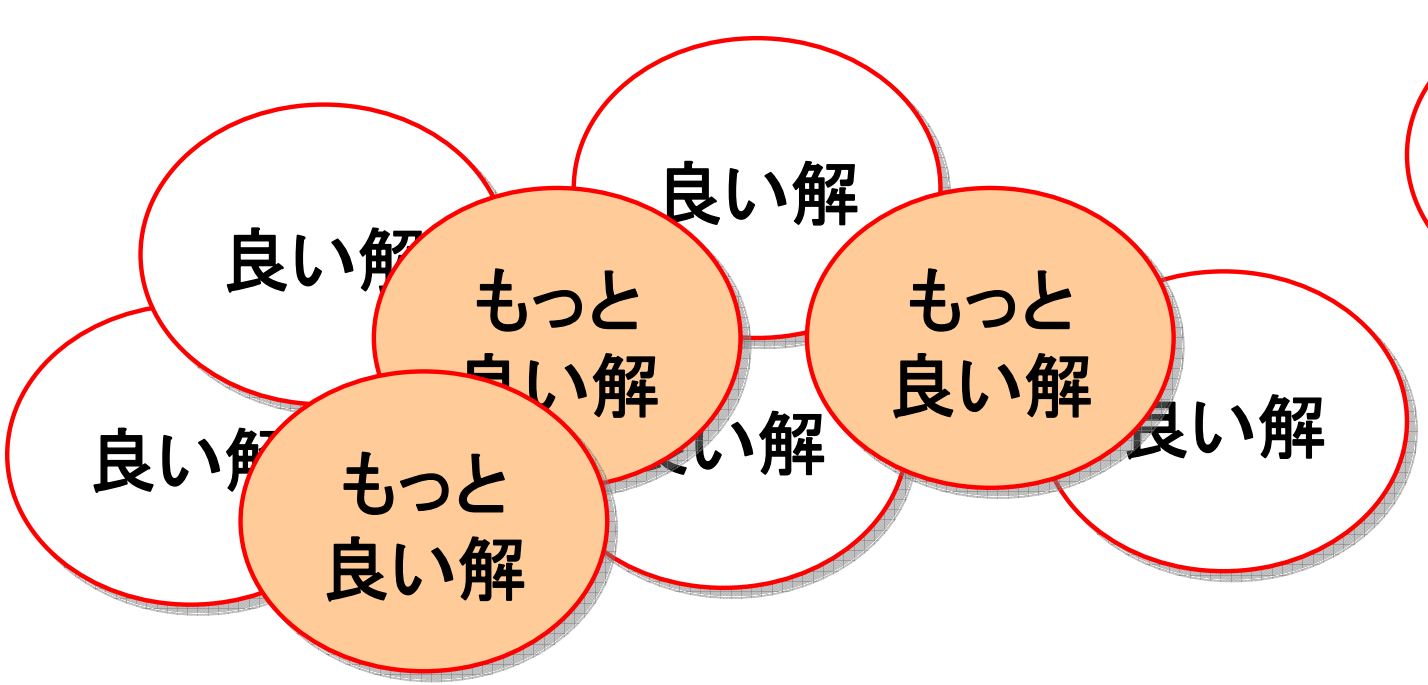
- 「距離」「荷物の重さ」「労働時間」、これら全て、局所的な変化が全体に影響しない
- そのため、配送計画は

「良い解は良い部分構造を持つ」

「良い解のそばには、より良い解がある」

という性質を持つ

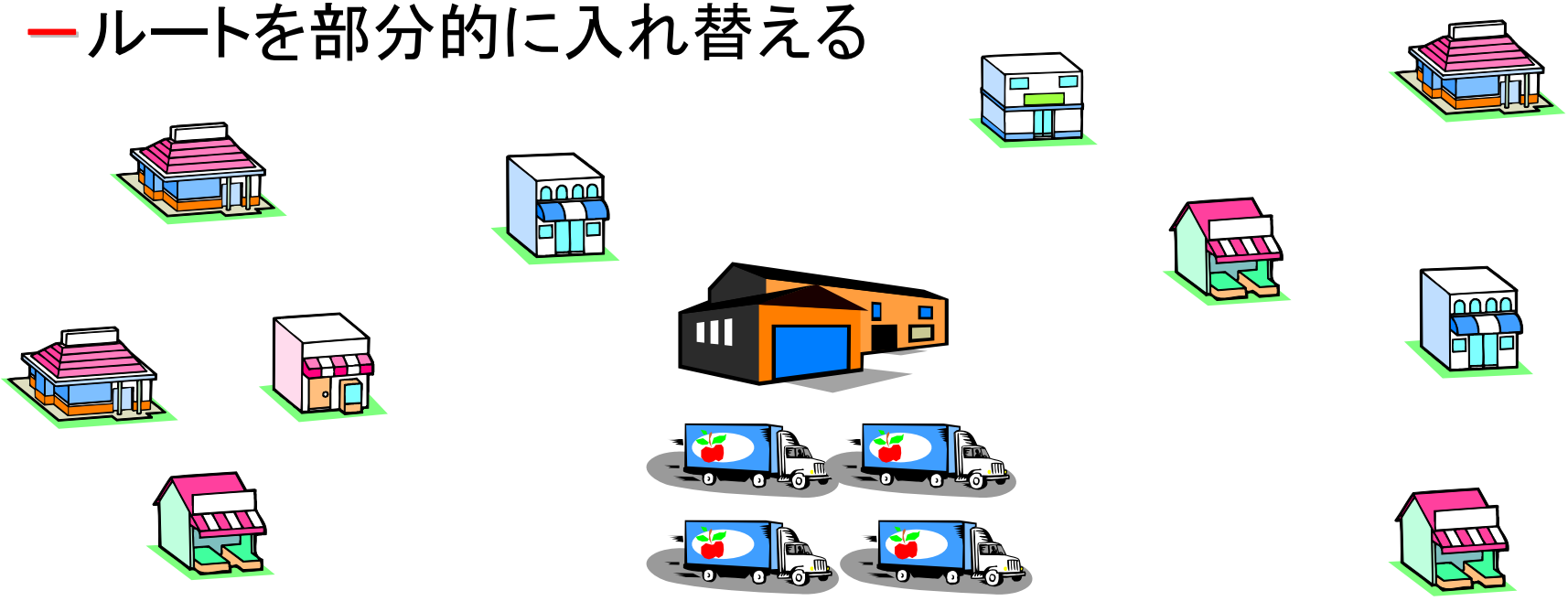
- そのため、良い方へ良い方へと進むと、解が改善される



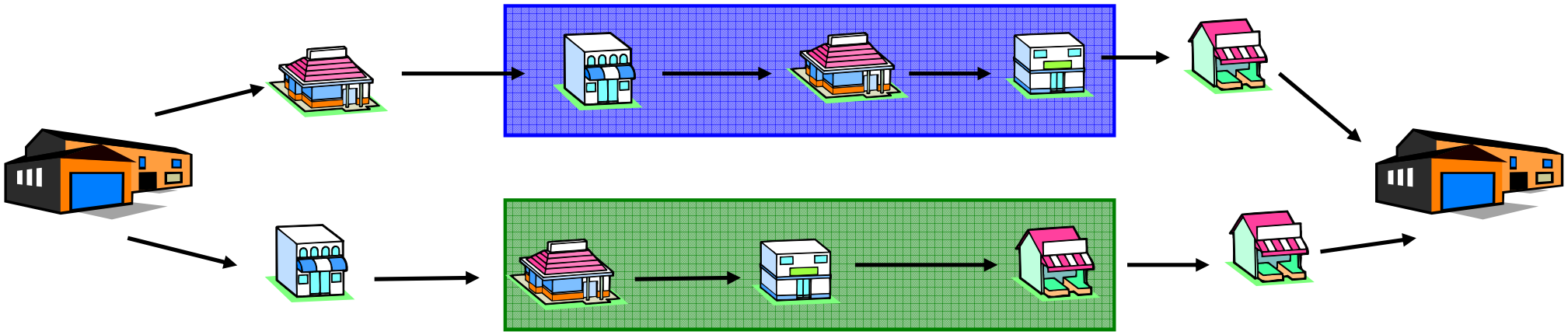


# 解の改善

- 人間は「うまくいっていない場所」を見つけて、改善する
  この作業をコンピュータにやらせてみよう
- うまくない場所を見分けるのは大変なので、全部の場所をちょこちょこ変えて改良することにする
  ➔ 試してみて、今よりよくなったら採用する
- 「ちょこちょこ変える」の部分は、変更のルールを作ろう
  ー ルートから1つ抜いて他の部分に挿入
  ールートを部分的に入れ替える



# 入れ替えの例



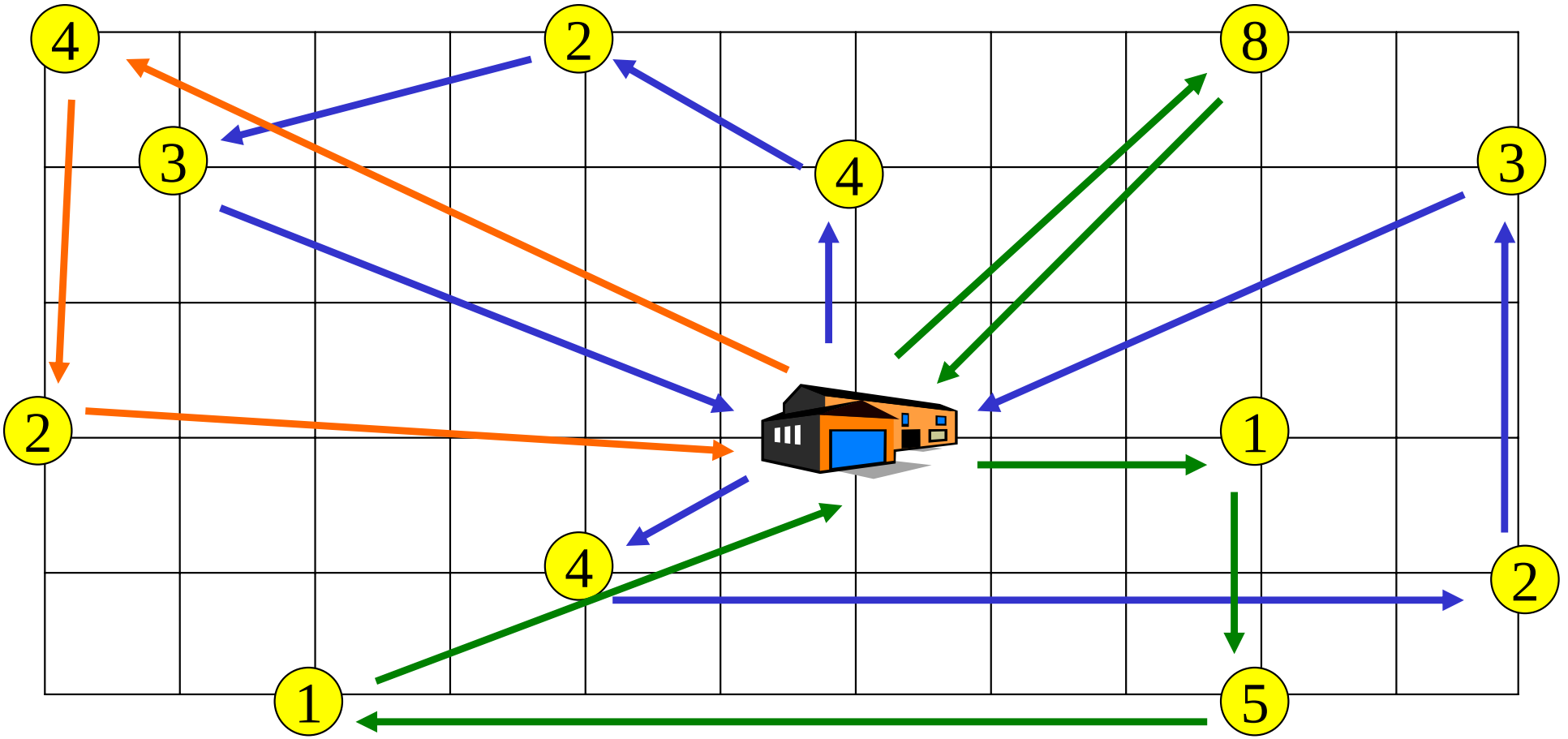
- ２つのルートから部分ルートを１つずつ選び、その部分を交換

① ちょこちょこ変更してみて、改善されたら採用する
② 変更してもいい解が得られなくなったら終了
　という仕組みだけでも、かなり良い解が得られる

**到着時間の制約がなければ、誤差10%くらいに落ち着くそうだ**

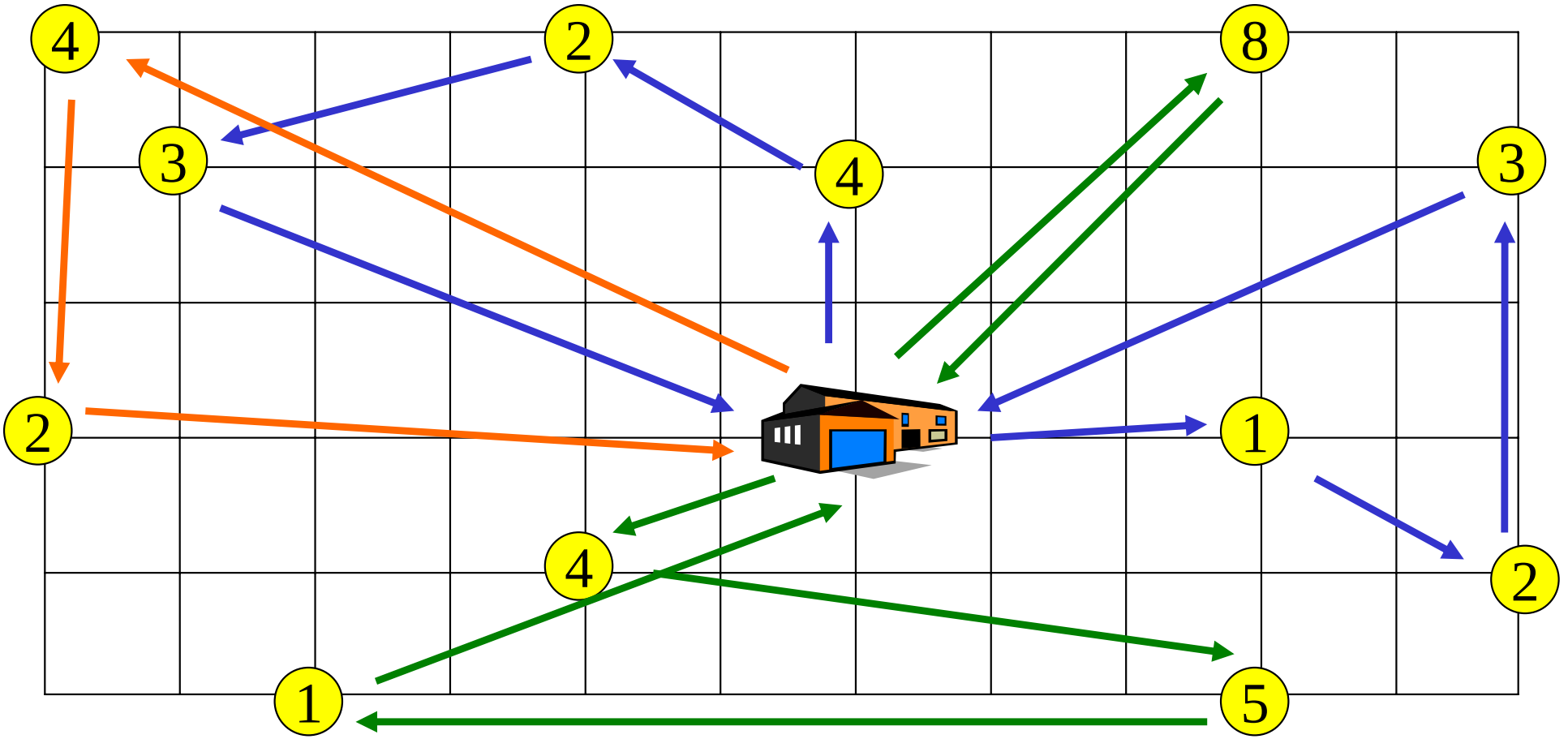
# ちょこちょこ、の例題

- 距離は縦横に移動した数、最大積載量は10、最大移動距離は 20



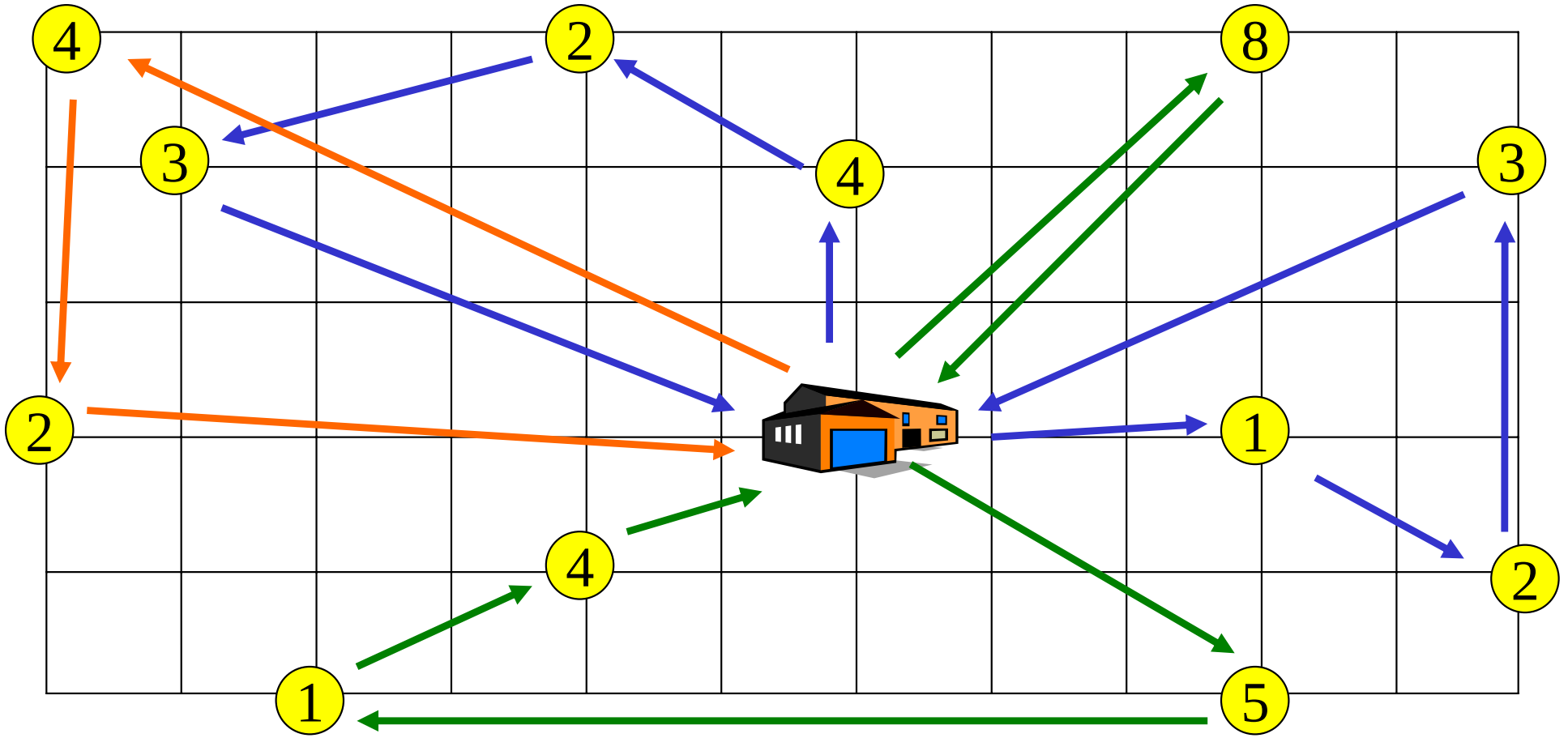
# ちょこちょこ、の例題

- 距離は縦横に移動した数、最大積載量は10、最大移動距離は 20



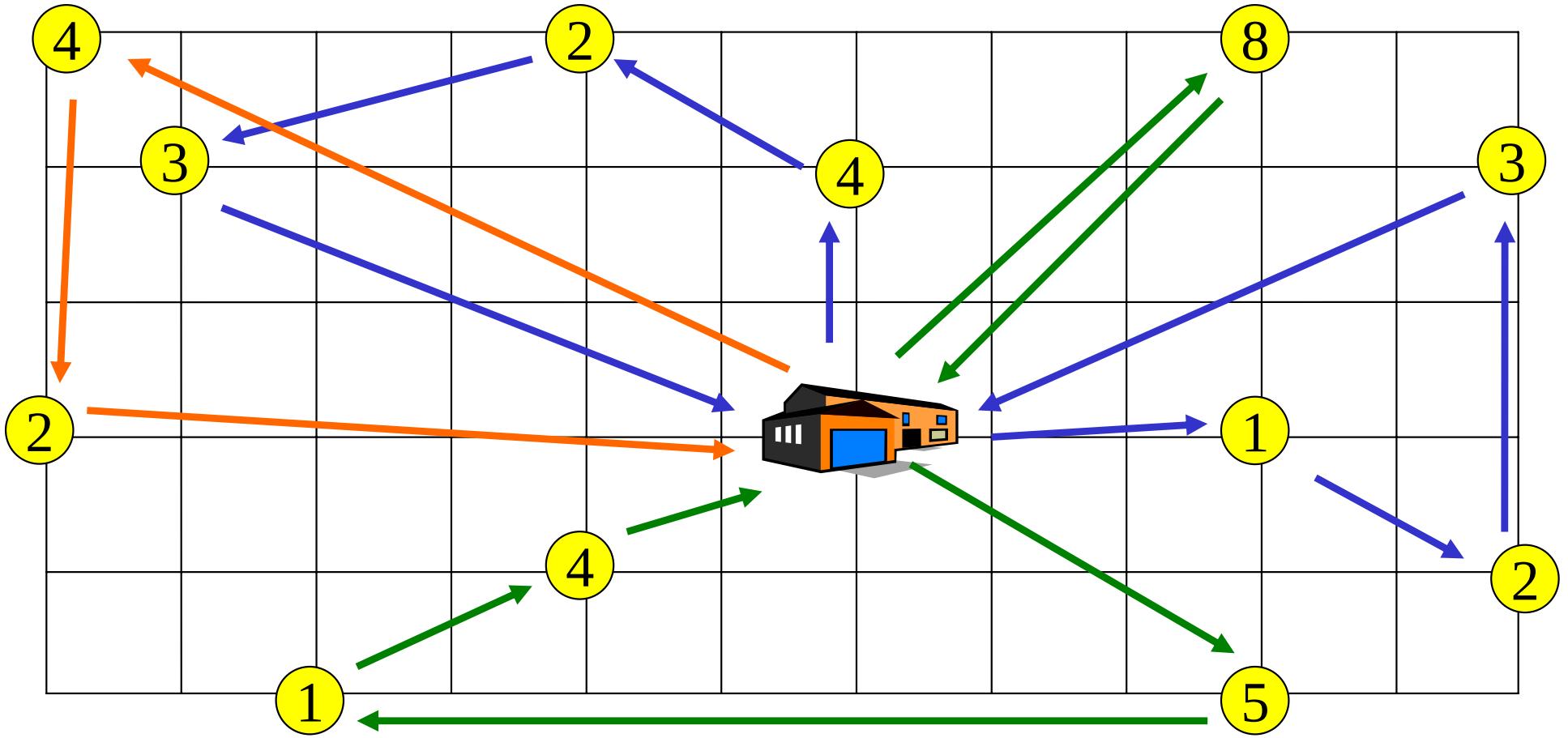
# ちょこちょこ、の例題

- 距離は縦横に移動した数、最大積載量は10、最大移動距離は 20



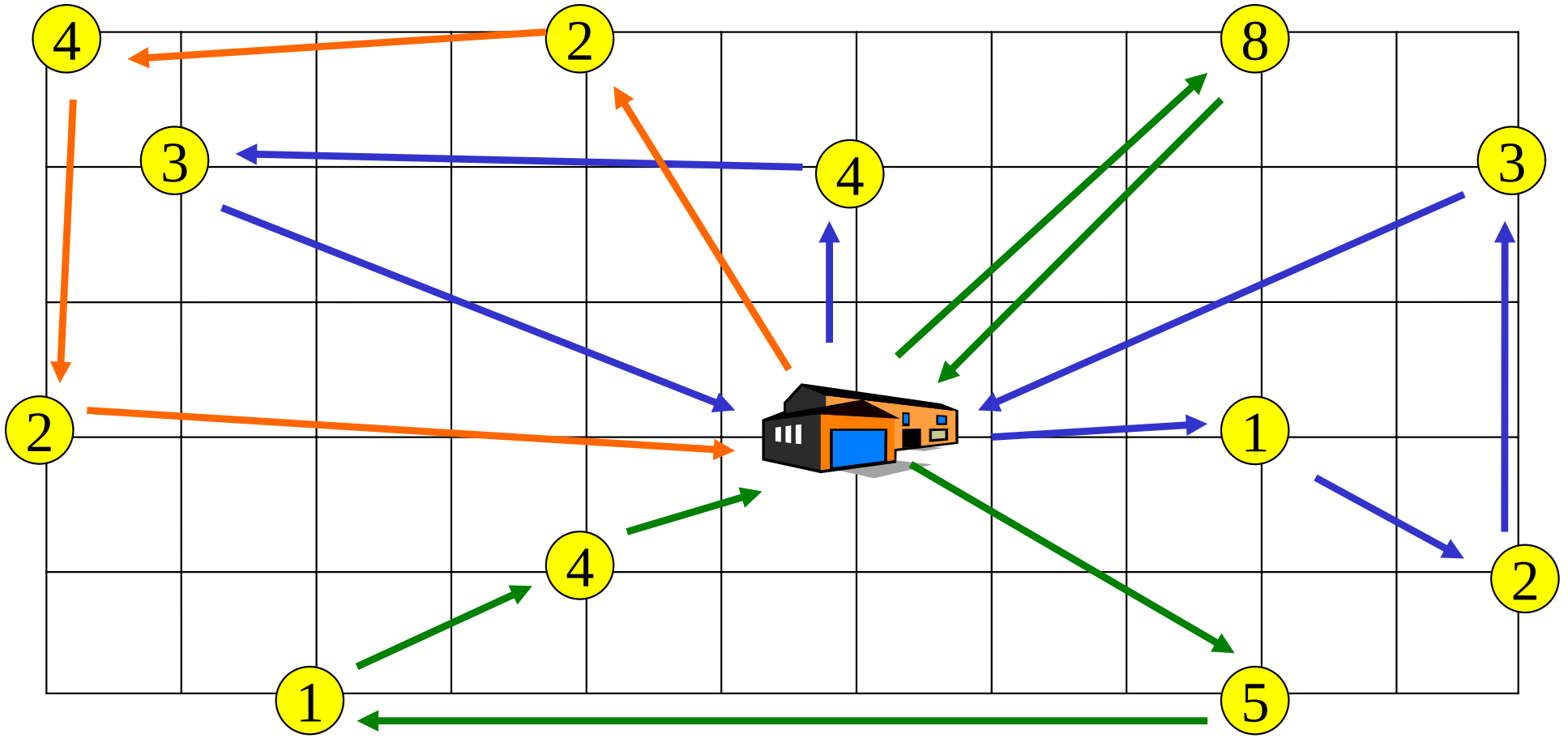
# ちょこちょこ、の例題

- 距離は縦横に移動した数、最大積載量は10、最大移動距離は 20



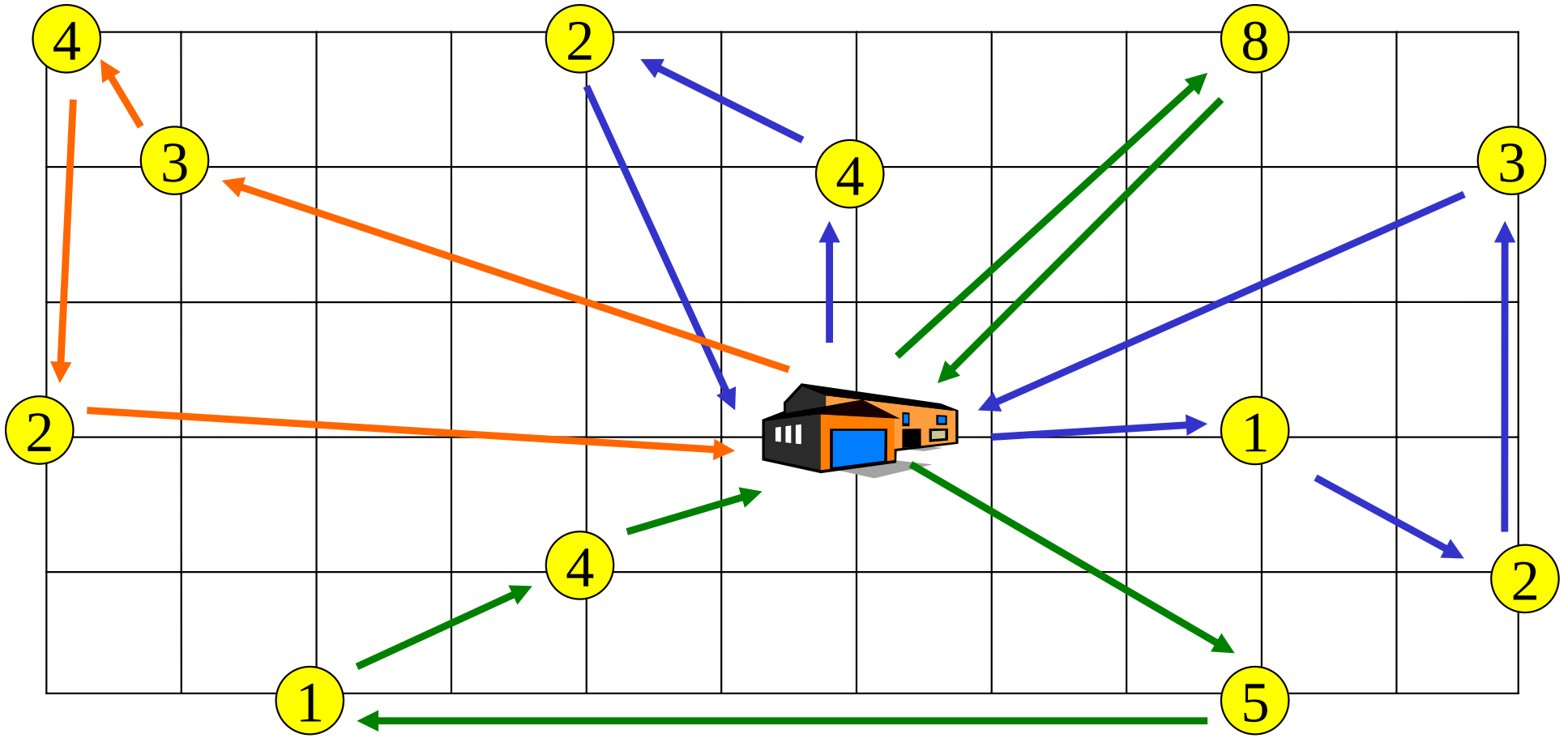
# ちょこちょこ、の例題

- 距離は縦横に移動した数、最大積載量は10、最大移動距離は 20



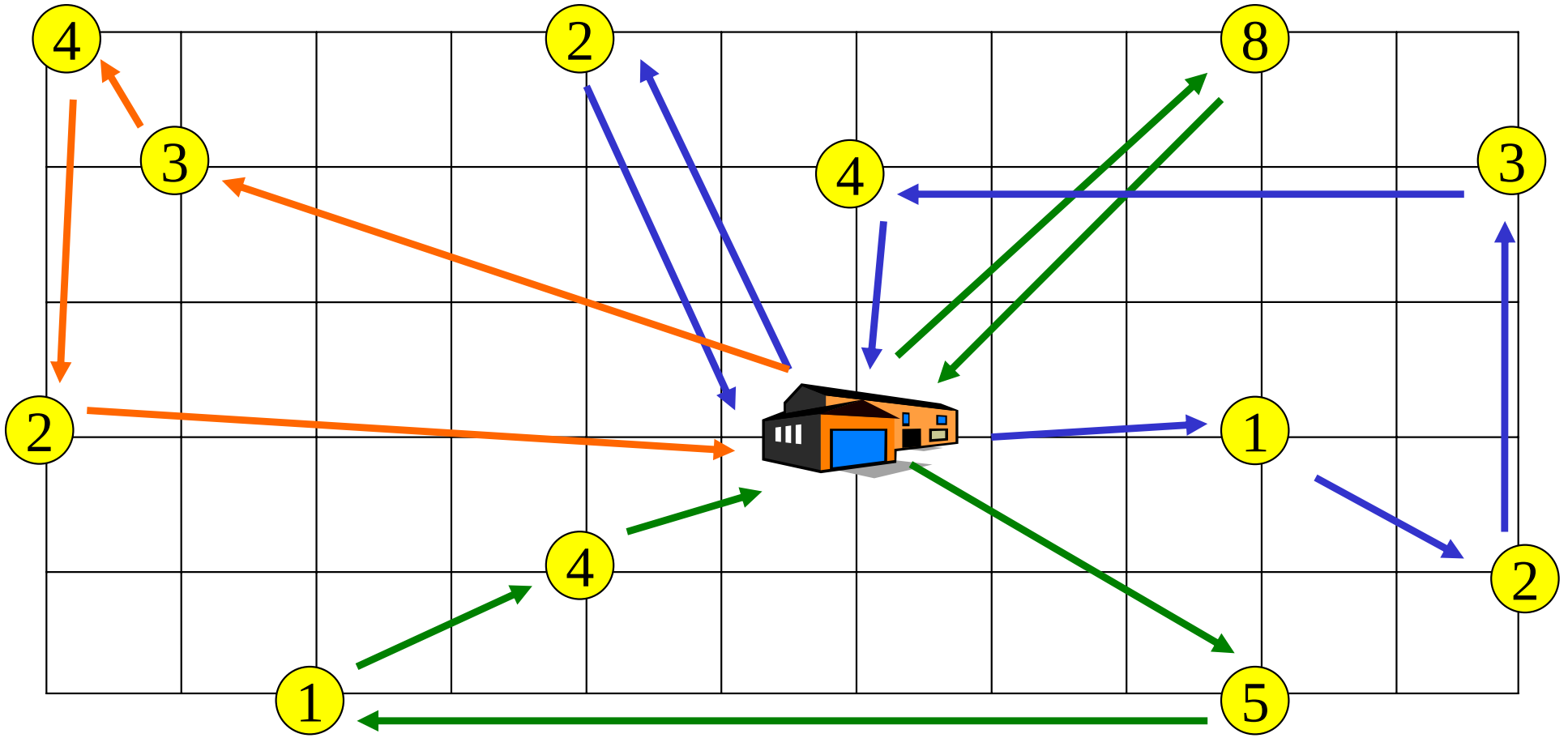
# ちょこちょこ、の例題

- 距離は縦横に移動した数、最大積載量は10、最大移動距離は 20



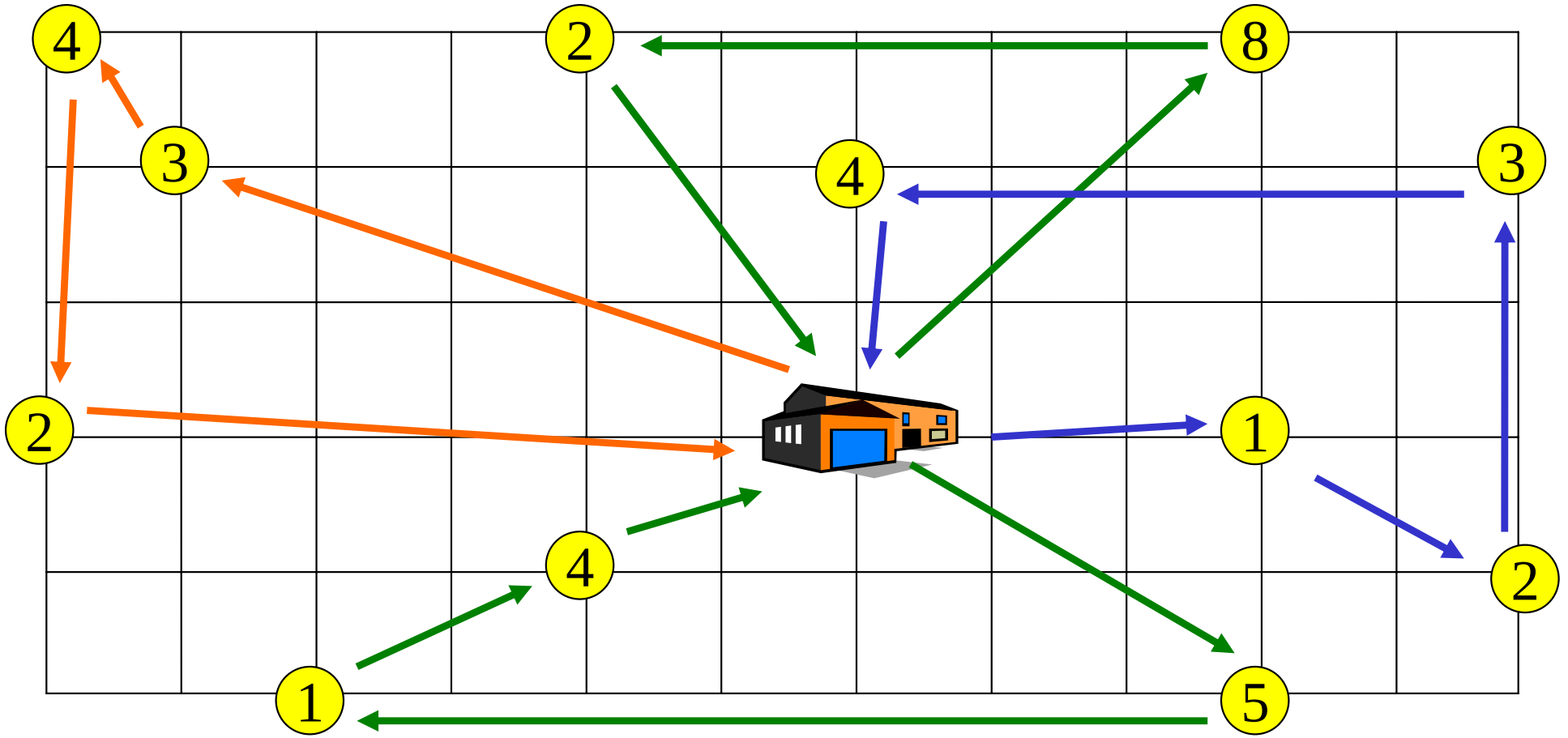
# ちょこちょこ、の例題

- 距離は縦横に移動した数、最大積載量は10、最大移動距離は 20



# ちょこちょこ、の例題

- 距離は縦横に移動した数、最大積載量は10、最大移動距離は 20

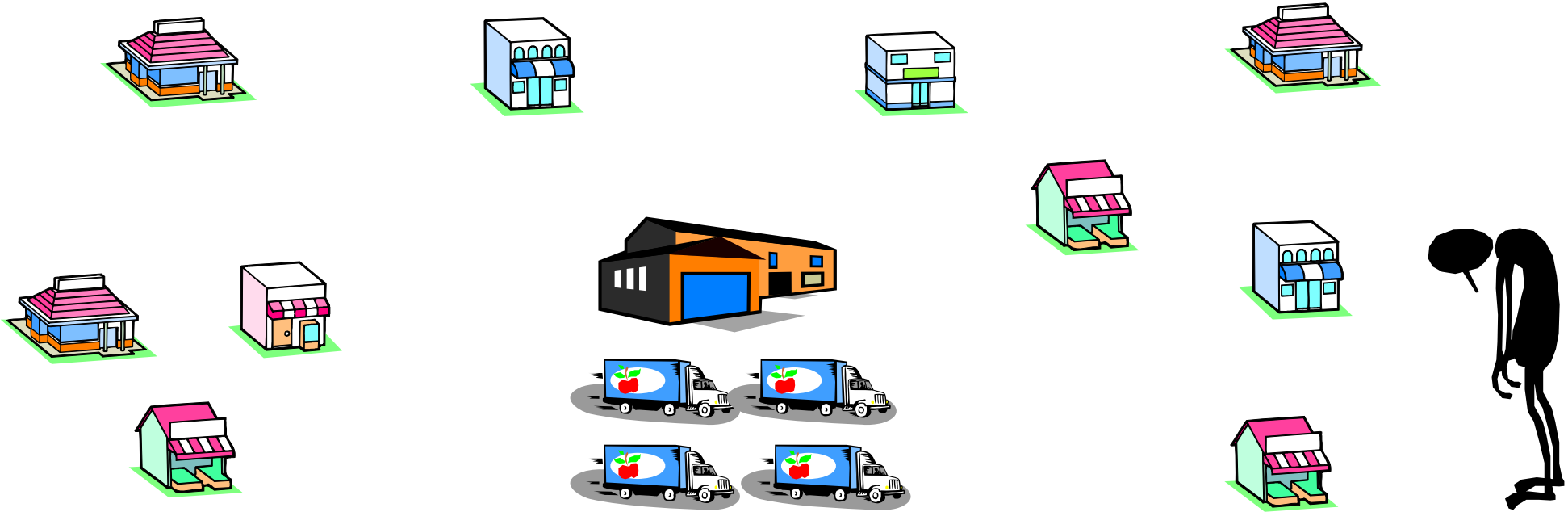


# 候補の数

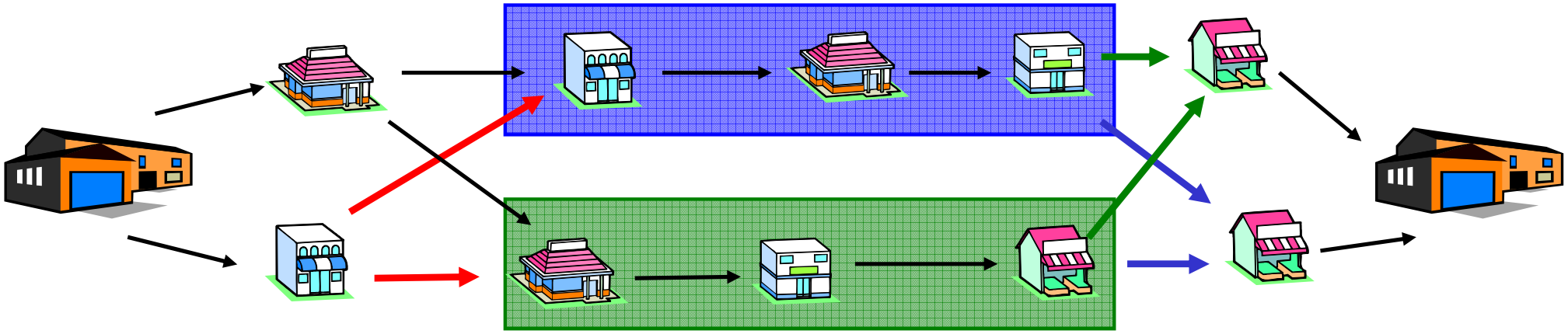
- **候補の数　≒　（客の数）$^2$（ルートの長さ）$^2$**

  - お客が100人、1ルート10人配送、と思うと 100万程度
  - お客が1000人、1ルート30人配送、と思うと 10億程度

➔ 何回も改善していては、とても短時間で解けない



# 候補の絞込み



- 部分ルートを入れ替えて、長さの合計が短くなるためには、少なくともどこかひとつの変化で改善がなければいけない

- 「つなぎかえると短くなるところ」のみ調べればいい

**候補の数が大幅に減る　（客の数×少々、くらい）**

# 最適解の運用

- ナビゲーションシステムと同じように

「出てきたルートはまあまあいいが、実際には道の通りやすさなど他の要素もあるので、少々変更したほうがよりよくなる」

➔ そのまま指示に従ってもよいし、参考にしても良い

- 各ドライバーが他人に影響を与えることなく、独立してルート変更できるので、運用しやすい

# 配送計画、まとめると

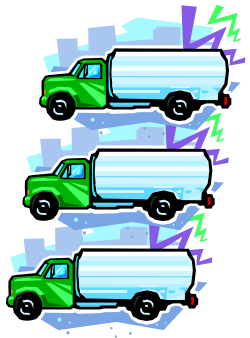
- 目的がコストなので、扱いやすい
- 決めるものは、ルートと配送先
- 制約も、数理的に表現しやすい　➔ モデル化がうまく行く

- 局所的な変更が全体に影響を与えない
  - ➔ 良い解のそばに良い解がある
  - ➔ 近視眼的に改善すると、比較的良い解が得られる

（到着時間の制約が入ると、こう簡単にはいかないが、
工夫するとうまくいく）

- 短時間で計算するためには、候補を絞込む必要がある

（改善の可能性のあるところだけ調べる）



# 最短路問題

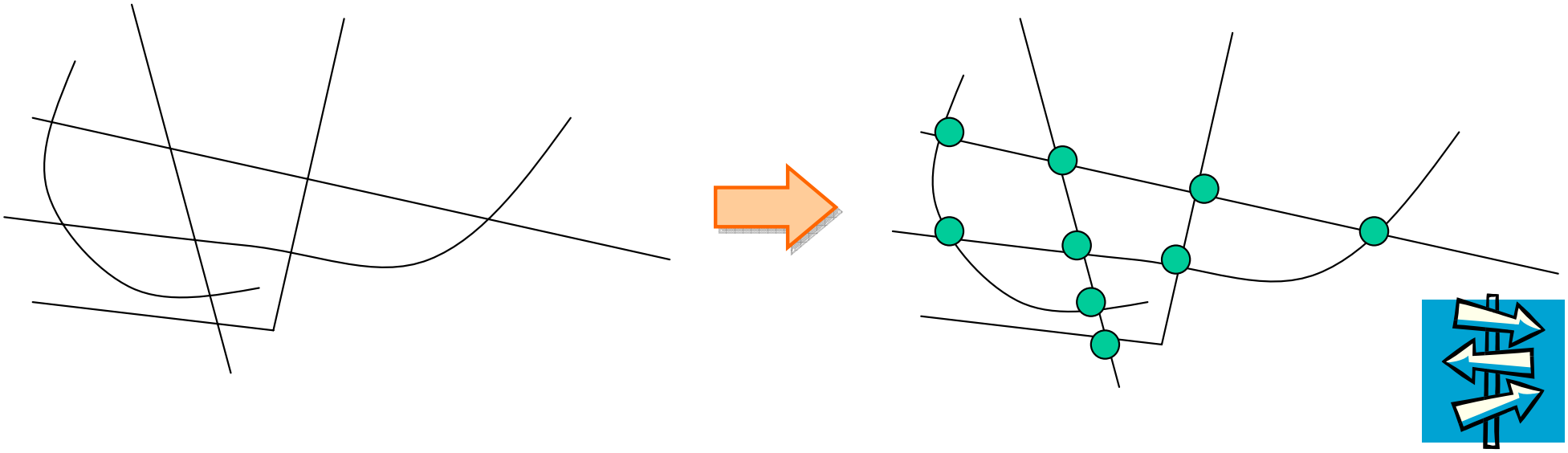
# 最短路問題

- 地図などのネットワークで、ある地点からある地点（駅から駅）への最も短い（安い）経路を見つける問題
  - 最短のドライブルートが知りたい
  - 旅行先まで最も安く行く電車バスの乗り方が知りたい


# 何を見つける問題か？

- 電車の経路は、乗換駅を出せばわかる
- 地図などの幾何学的なデータも、道案内をする立場からは、
  交差点ごとにどちらに行くかを示せばいい
- 地図も交差点の隣接関係のみに注目して、抽象化できる

（交差点から交差点へ必ず進み、Uターンはしない）



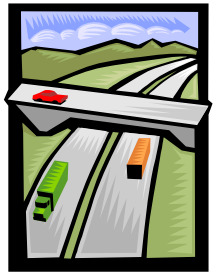
**つまり「交差点・乗換駅ごとに、どちらに行くべきかを決める」問題**

# 最適化の問題としてまとめると


**目的**：現在地から目的地へ向かう最短（最安）の経路

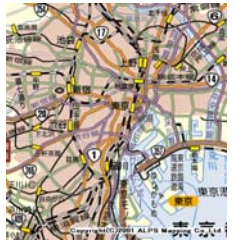
**決めるもの**：交差点ごとに、どちらに進むか

**制約**：（高速道路・特急を使うか、値段はいくらまで、など）



**解法**：ボタンを押したらすぐに答えが出てほしい

（ダイクストラ法を使うとうまく解ける）

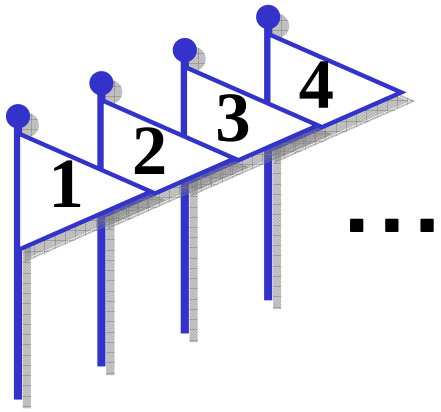
**データの収集**：地図・路線図の電子的なデータを使う

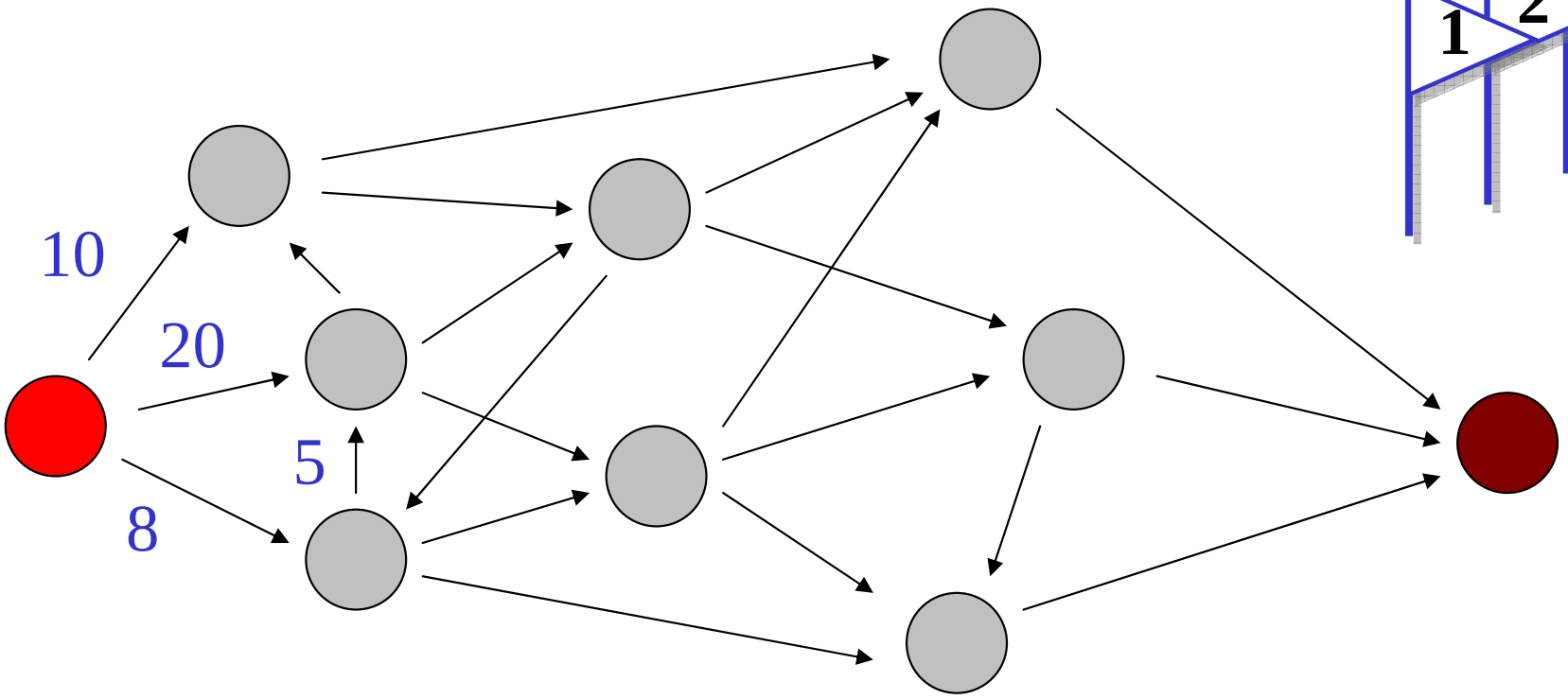
**運用**：教えられた通りそのままは知ってもいいし、参考程度にしてもよい

ダイクストラ法がどのようなものか、見てみましょう

# ダイクストラ法

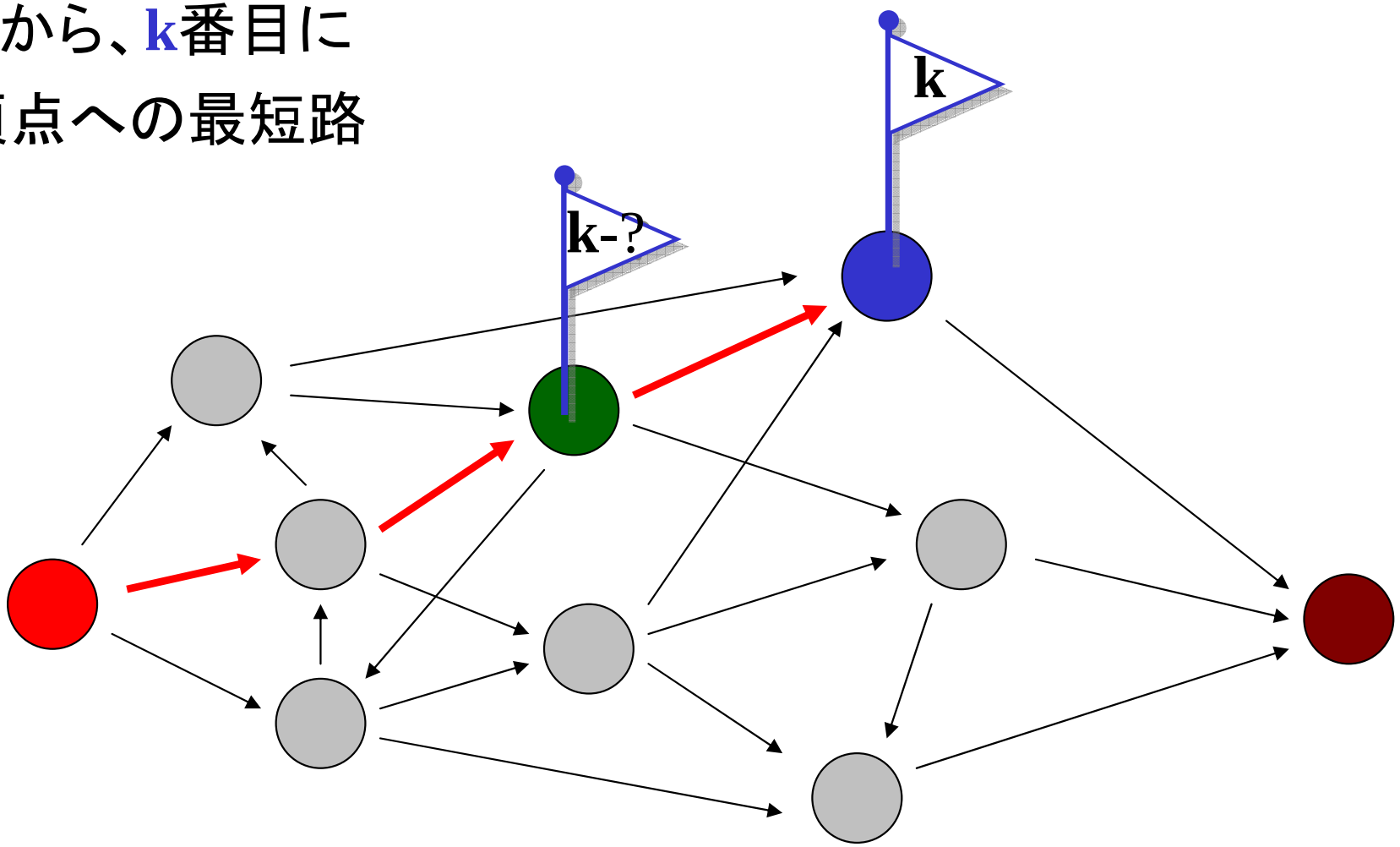
- ダイクストラ法のアイディアはとても簡単
- 始点に一番近い頂点、2番目に近い頂点、と順々に見つけていく、というもの



# ダイクストラ法で使う性質 (1)

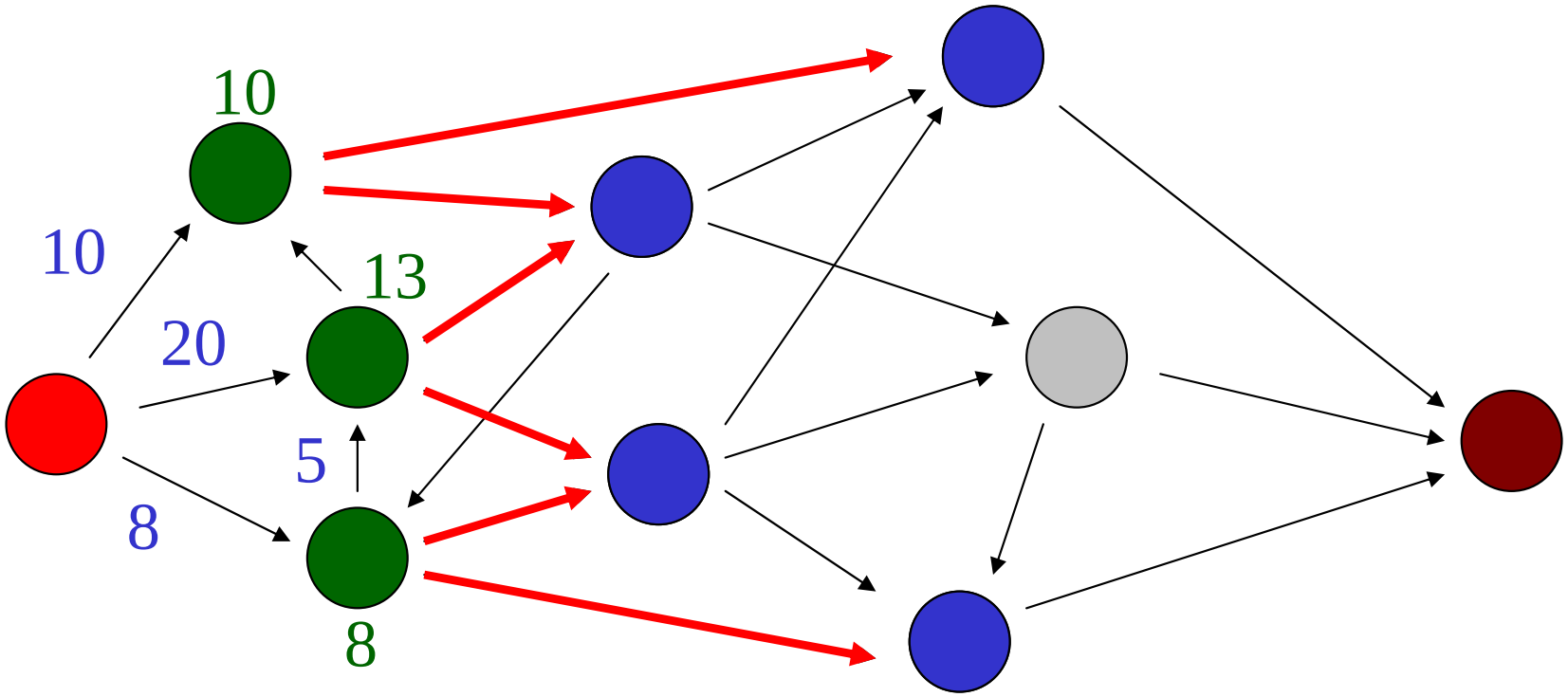
- 始点から、k番目に
近い頂点への最短路




- 緑の頂点は、k-1番目以下に近い

# ダイクストラ法で使う性質 (2)

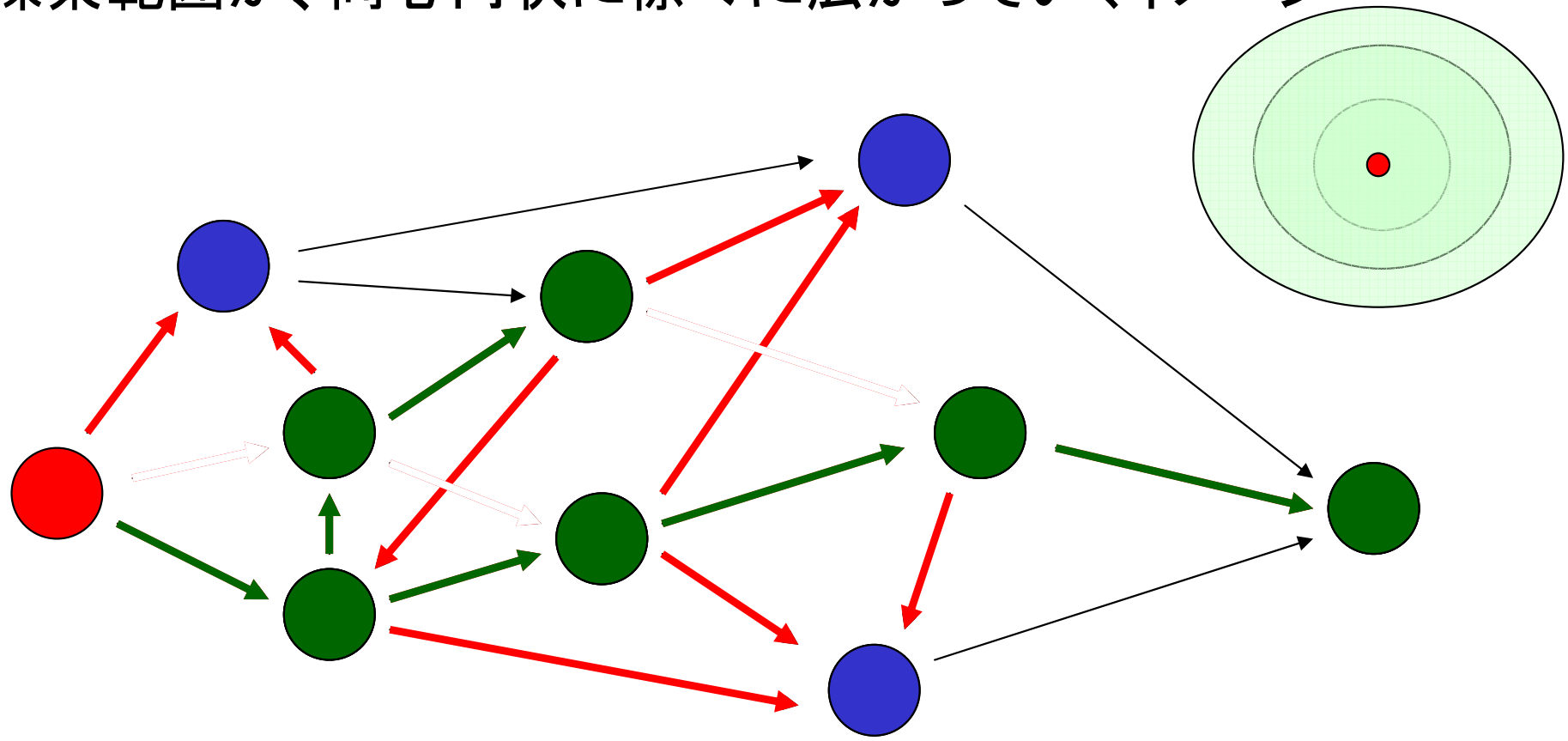
- 距離 **k-1** 番目までの頂点・距離がわかっている
- 距離 **k** 番目の頂点はどこにあるでしょう？




**青のどれか。赤線を使った経路の中で最短なものを見つける**

# ダイクストラ法

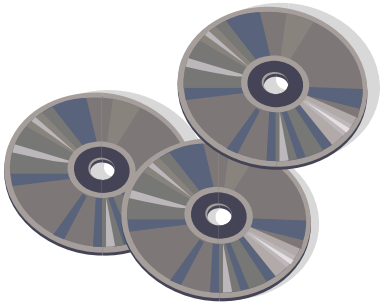
- 以下同様に、始点に一番近い頂点、2番目に近い頂点、と順々に見つけていく
- 探索範囲が、同心円状に徐々に広がっていくイメージ

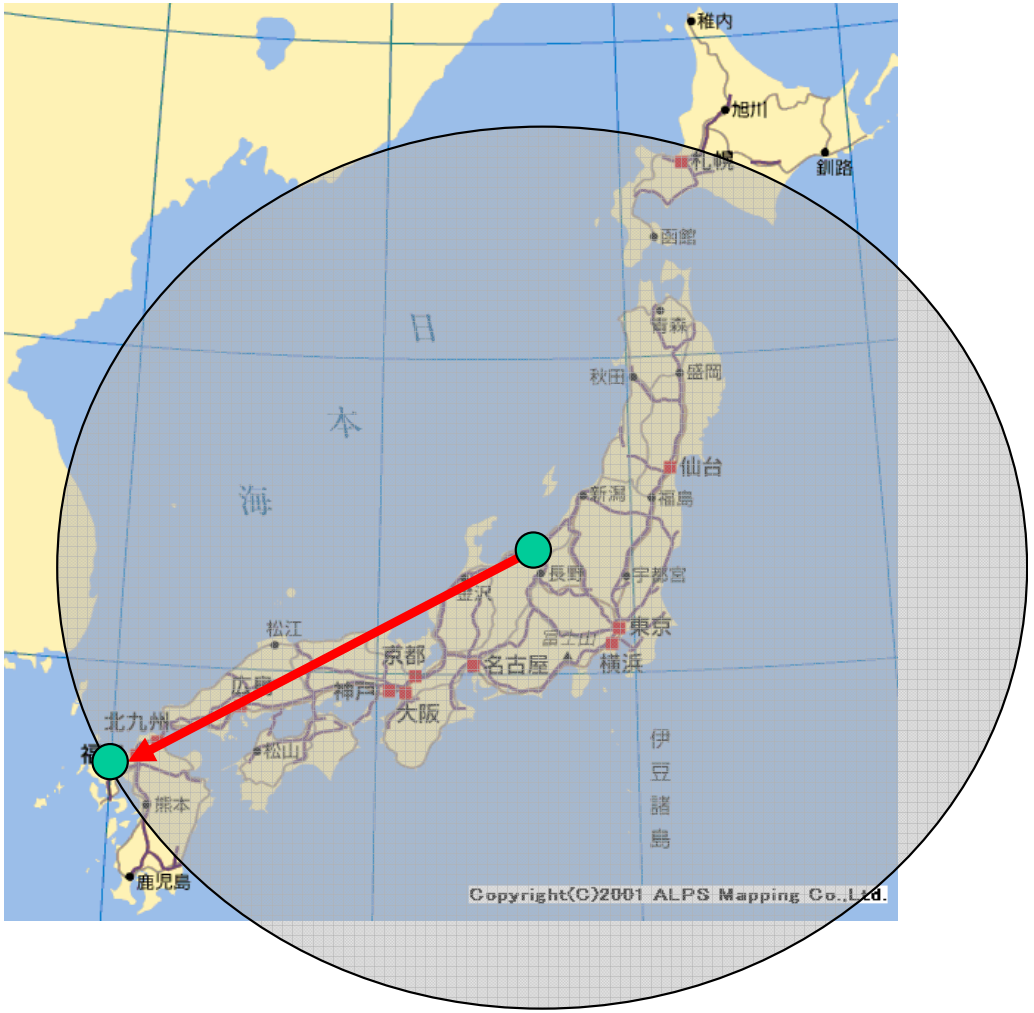


# 問題点

- ナビゲーションシステムで、実際使う場合を考える
- 離れたところへの最短経路を求めると、非常にたくさんのデータを読む

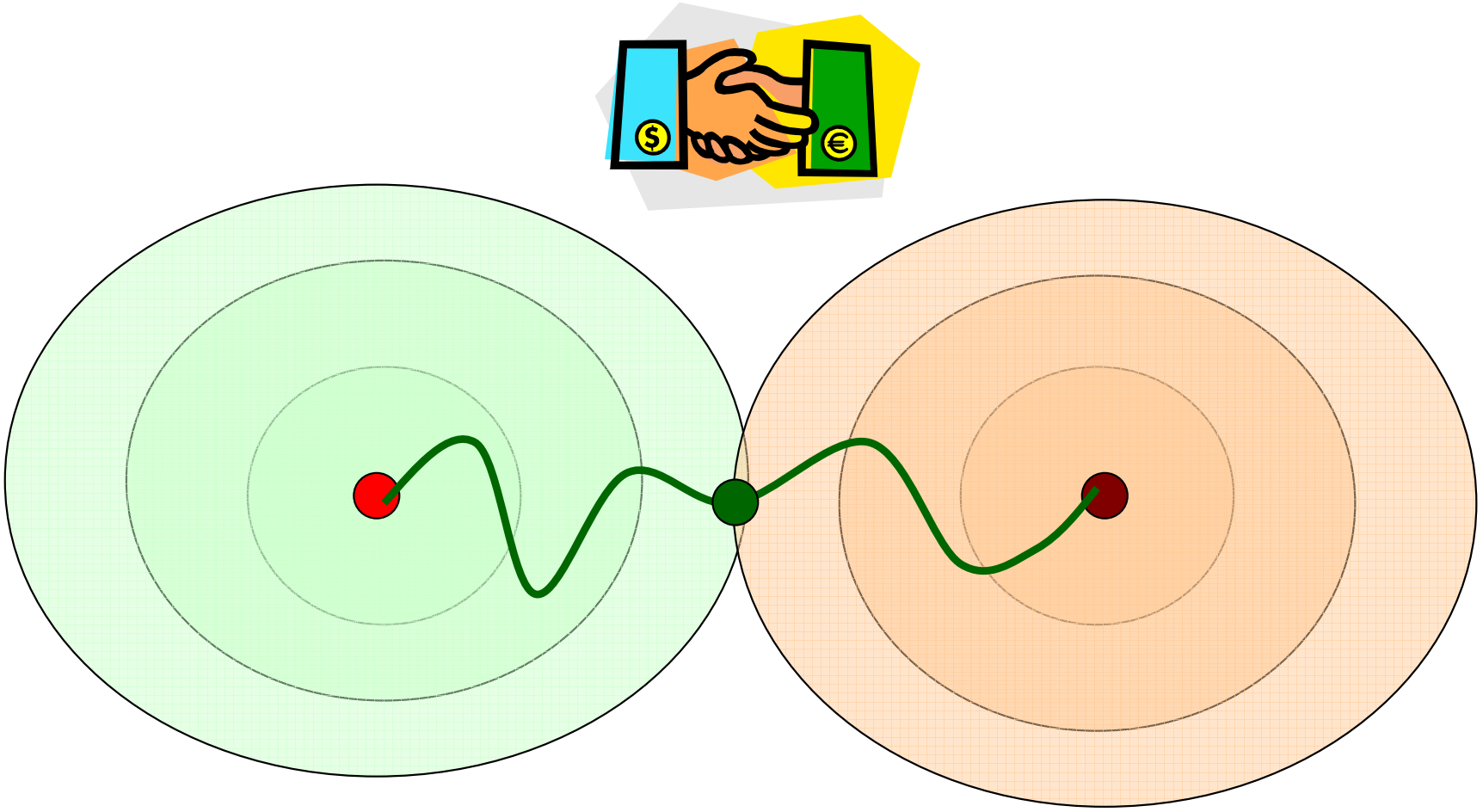
← ダイクストラ法は目的地より近い所は、全て探索する





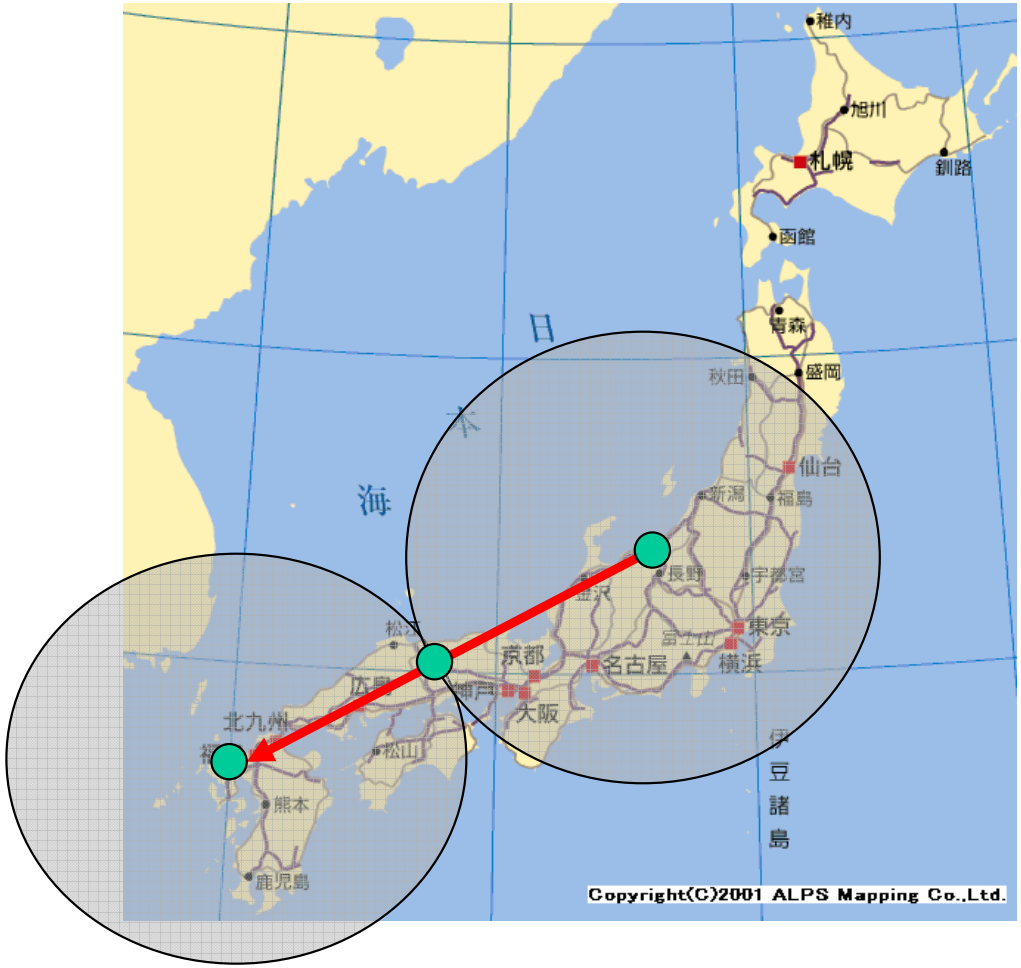
# 始点終点両方から探索

- 始点と終点、両方から同時にダイクストラ法を動かす
- 出会った所で最短路が得られる



# 少々改良

- 読み込む範囲が多少、少なくなる



# 幾何学的な情報の利用

- ダイクストラ法は、問題の持つ幾何学的な情報をまったく利用していない
  - ← 普通、目的地から遠ざかる方向へは探索しない

- 目的地に近いところから優先的に、探索の手を伸ばしていきたい
- ダイクストラ法をベースにするには、出発点からの距離が短い順に探索をしたい

→ 間を取って、

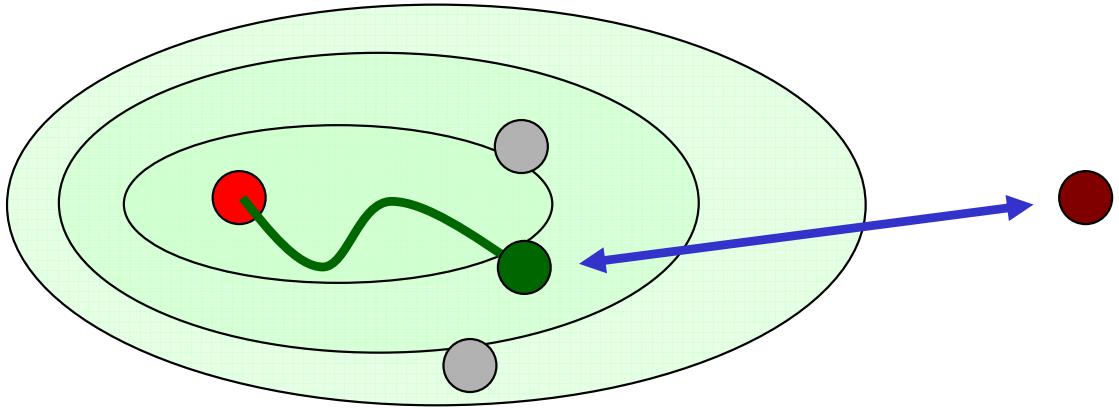
（始点からの距離）＋（終点までの直線距離）

が小さい頂点から順に見つけていこう

# A*アルゴリズム

- （始点からの距離）＋（終点までの直線距離）
  が小さい頂点から順に見つけていこう
  （探索範囲が、目的地方向に伸びやすくなる）

- 実際の距離　≧　直線距離
  が成り立つので、これを利用すると、最適性が証明できる
（実際には、「実際の距離」の下界を用いることができる）

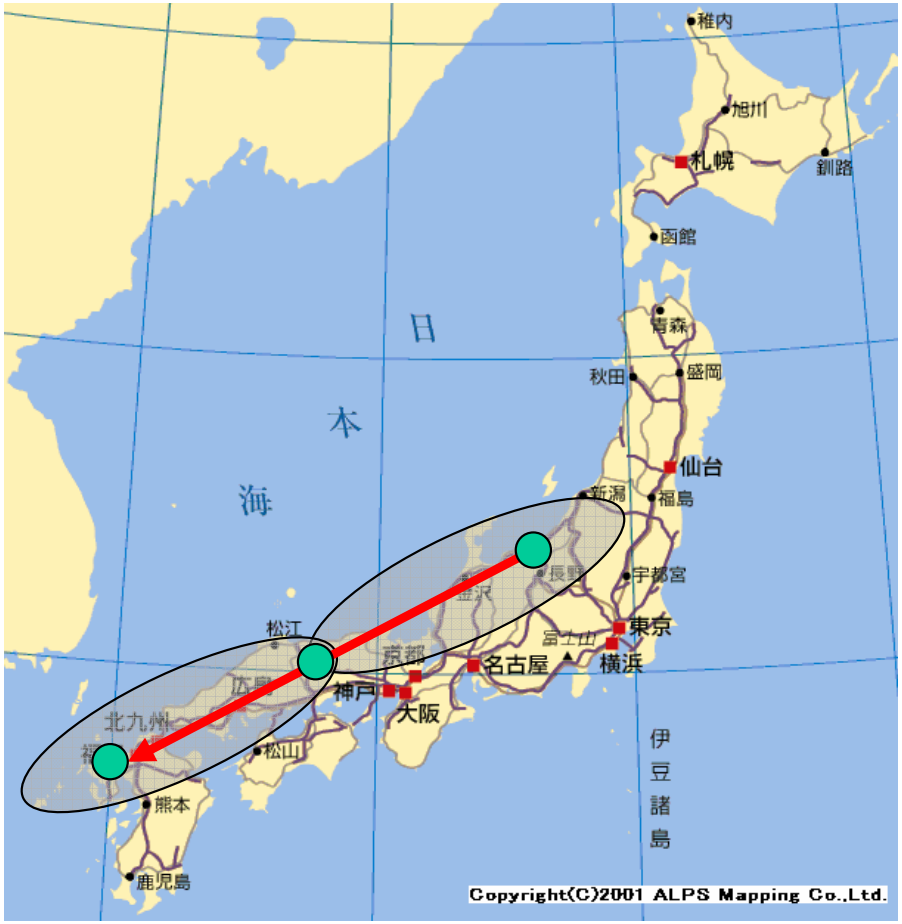


# だいぶ改良される

- 読み込む範囲がだいぶ、少なくなる

- しかし、読み込む範囲に関しては、細い道まで含め、全部のデータを読む

- 通常、遠距離の移動には、細い道は使わないので、これは明らかに無駄



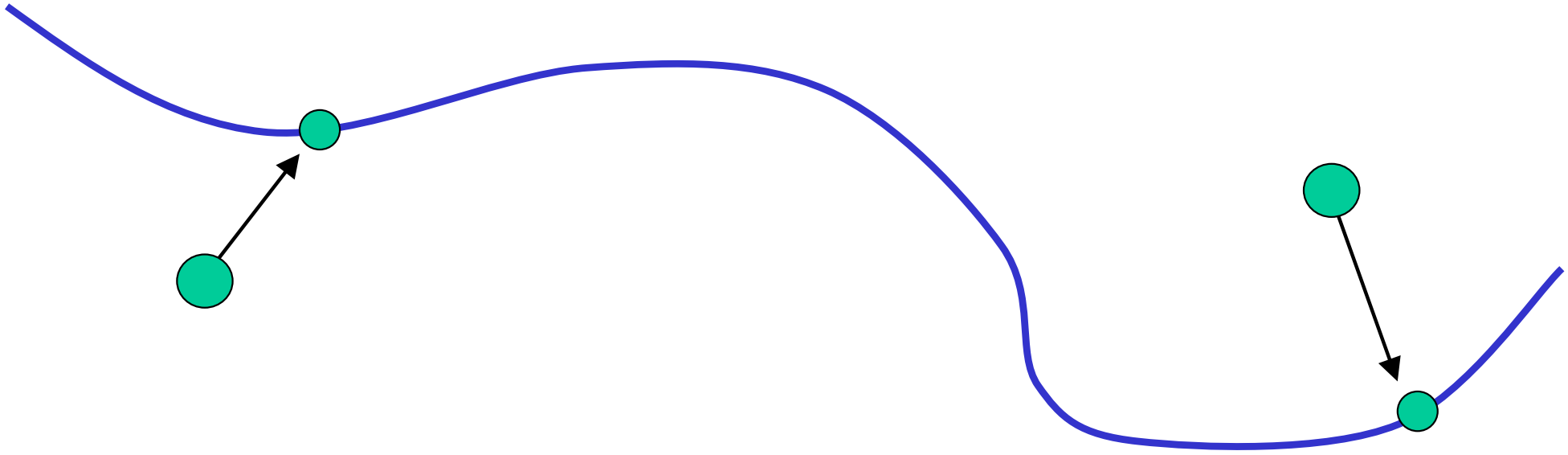
# 2つのレイヤーに分ける (2)

- では、2つのネットワーク(**レイヤー**)に分けましょう
  - ① 高速道路(＋太い道)の**レイヤー**
  - ② 全ての道の**レイヤー**

**2つのレイヤーをどのようにつなぐか**


# レイヤーの使い方

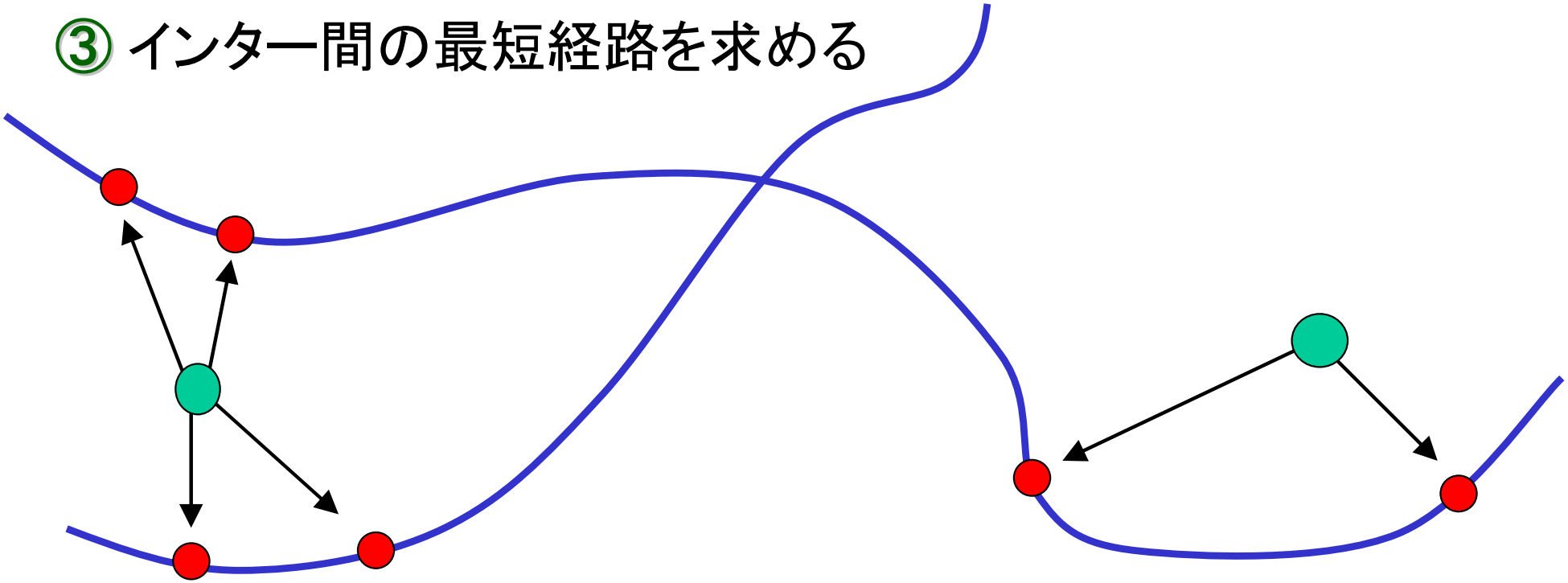
① 現在地から、最寄のインターを見つける
② 目的から最寄のインターを見つける
③ インター間の最短経路を求める



しかし、最寄インターを使うルートが最適とは限らない

# レイヤーの使い方 (2)

① 現在地から最寄のインター(複数)までの
最短経路を求める
② 目的地から最寄のインター(複数)までの
最短経路を求める
③ インター間の最短経路を求める



出てきた経路の中の最短経路を選べばよい

# 最短路問題のまとめ

- 目的が距離や値段の減少なので、扱いやすい
- 決めるものは、交差点の曲がり方
- 制約は、ゴールに到着すこと。簡単

- 現在地から近い順番に見つけいくと、必ず最短経路が

見つかる、という良い性質がある

➔ ダイクストラ法ですばやく解ける

（例えば、値段が××以内で最も速いルート、というような検索をさせたいときには、こうはいかない）

- （データが巨大なため）短時間で計算するためには、なるべく必要な部分しか見ないように、解法を工夫する必要がある

# そのほかの問題

# 生産計画

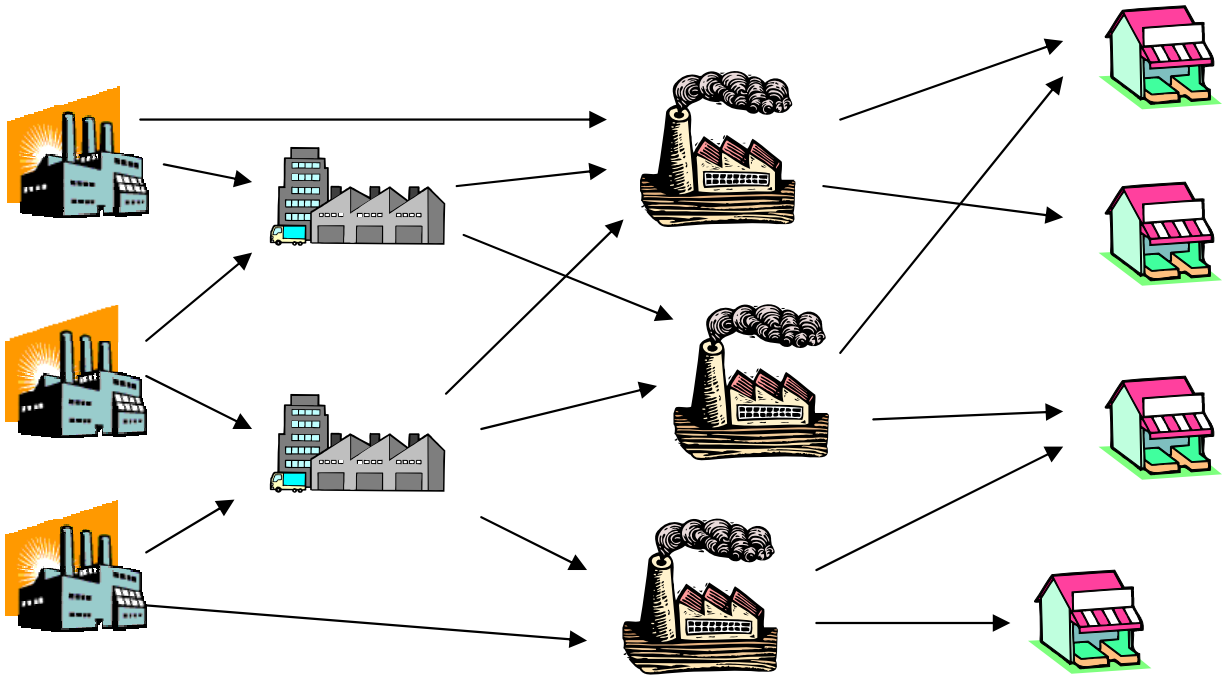
（サプライチェーンの輸送・在庫コスト最適化）

- いくつかの部品工場を持つメーカーを考える
- 各工場は、他の工場から部品を持ってきて、加工・組立てし、できた部品・製品を他の工場へ送り出す（あるいは在庫として蓄える）
- 輸送＆在庫コストを最小化する

**決めるもの**：
工場間の輸送量と時期

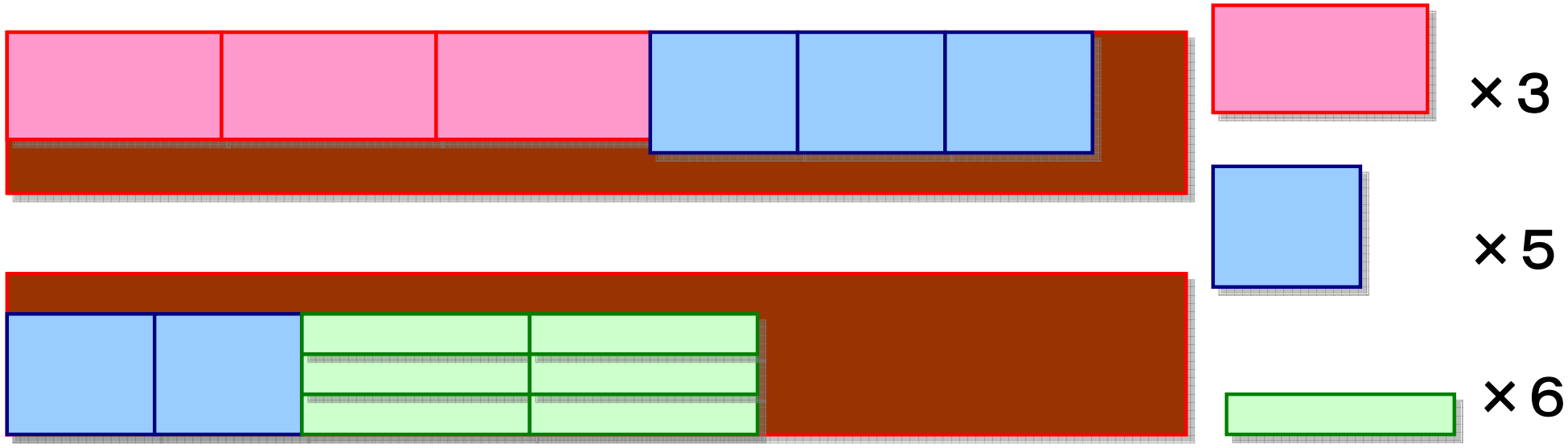
**制約**：必要な部品が足りていること

**解法**：**凸性**という、近視眼的に進むと必ず最適解に行くといういい性質があり、それを用いた**線形計画**で解ける



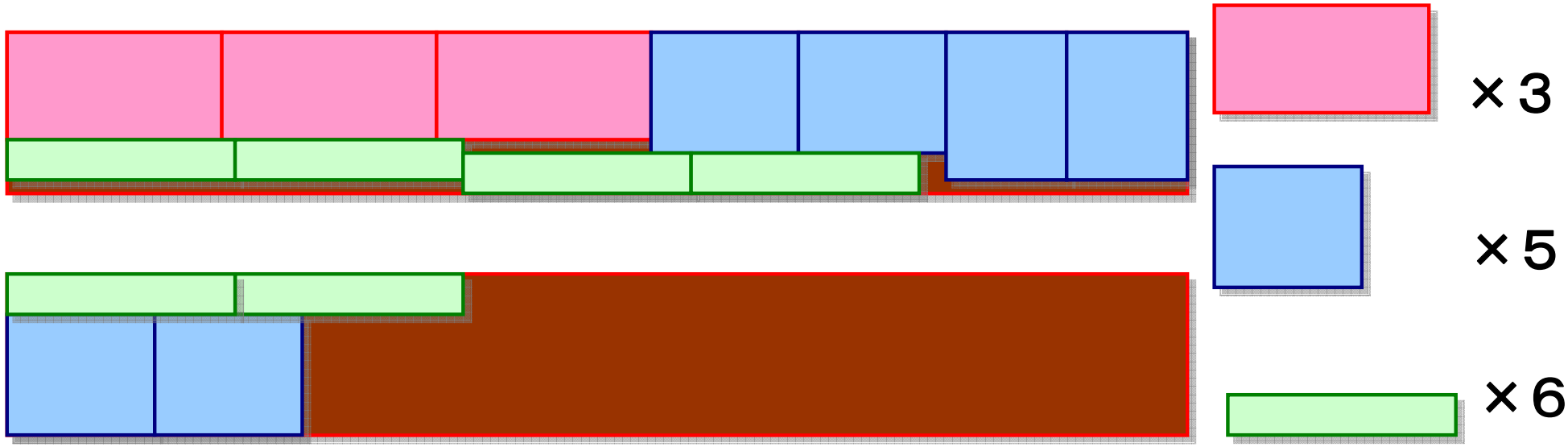
# 板取計画最適化

- 製鉄所の鉄板、あるいは布地を作る工程の最適化
- 作る鉄板・布地の大きさは一定。ここから、顧客が注文した大きさの、あるいは布地を切り取る
- なるべく無駄なく切り取るよう、切り取り方を最適化する



# 板取計画最適化

- 製鉄所の鉄板、あるいは布地を作る工程の最適化
- 作る鉄板・布地の大きさは一定。ここから、顧客が注文した大きさの、あるいは布地を切り取る
- なるべく無駄なく切り取るよう、切り取り方を最適化する
- 「良い解のそばにより良い解がある」が成り立つので、近視眼的な解法でうまく解ける。運用もやり易い



# スタッフのスケジューリング

**問題**: パイロットや客室乗務員、あるいは看護師の勤務表を作る

**目的**: 各個人の負担・休日の希望をなるべくたくさんかなえたい

**制約**: 複雑な制約が数多くある

- 遠くに行ったら、その日はそこに宿泊しなければいけない
- 電車で帰ってきても良い
- 各人の宿泊は多くなりすぎない
- 各人に土日の休日がある
- 夜勤や準夜勤の数がたくさんになるといけない

. . .

- 1ヶ月に一回解く程度、じっくり時間をかけて良いものを作りたい

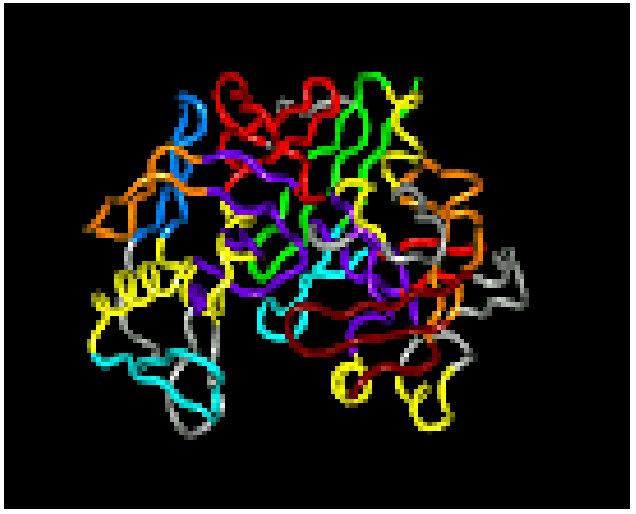
- いい解のそばにいい解がある、という性質があまり成り立たない
- 実際に解を求めるのは大変。そもそも制約を満たす解が見つからない
- 技術の粋を集めると、何とか解ける

# たんぱく質の形状

- たんぱく質はアミノ酸がつながった鎖状の物質
- 鎖が折れ曲がって、電気的に安定した状態になる。
- たんぱく質の機能は形で決まるので、形の解析はとても重要
- しかし、アミノ酸のつながりは解析しやすいが、形の解析は難しい

➔ 最適化で何とかしよう

**目的**:たんぱく質のなるべく安定した形状を求める
**決めるもの**:鎖の各点での折れ曲がりの角度
**制約**:原子がぶつからないこと
**データ**: 実験で入手
**運用**: 機能の推測＆参考にする



- データが3次元なので、扱いづらい
- 1部の変更が全体に影響を及ぼす
- 最適化は、非常に難しい

# 最適化のまとめ

- 最適化とは、変数の値を決めることで、最も良い状態をつくる方法のこと