Chapter 5.
Optimization

- 최적화 기법
  - 여러 변수 존재하고 다양한 해법 있는 문제
  - 변수들의 조합에 따라 산출물이 크게 달라지는 문제
  - 물리학의 분자역학 연구, 유전학의 단백질 구조 예측, 컴퓨터과학의 알고리즘 최악 실행 시간 추정 등
- 예제 : 단체 여행 문제
  - 다양한 입력 변수 존재
    - 각 사람의 여행 일정
    - 차 렌트 기간?
    - 어떤 공항 편리?
    - 전체 비용, 공항 체류 기간, 휴가 기간 등 고려해야
  - 최적화 문제

1. 단체 여행

- 여러 다른 장소에서 출발하여 같은 목적지를 찾아가는 단체 여행
- optimization.py
  - 가족 구성원은 전국에 퍼져있고 뉴욕에서 만날 계획
  - 같은 날 출발, 같은 날 도착
  - 가능하면 공항 이용 운임 공유
  - 다양한 항공 요금, 항공편

```
import time
import random
import math

people = [('Seymour','BOS'),
          ('Franny','DAL'),
          ('Zooey','CAK'),
          ('Walt','MIA'),
          ('Buddy','ORD'),
          ('Les','OMA')]
# Laguardia 공항
destination='LGA'
```

- 비행편 데이터 샘플 파일
  - schedule.txt
  - 비행편별 출발지(origin), 도착지(dest), 출발시간, 도착시간, 가격정보가 콤마로 분리되어 저장

```
LGA,OMA,6:19,8:13,239
OMA,LGA,6:11,8:31,249
LGA,OMA,8:04,10:59,136
OMA,LGA,7:39,10:24,219
```

  - origin과 dest를 키로 하고, 나머지 비행편 정보를 값으로 하여 사전 생성

```
flights={}
#
for line in file('schedule.txt'):
  origin,dest,depart,arrive,price=line.strip().split(',')
  flights.setdefault((origin,dest),[])

  # Add details to the list of possible flights
  flights[(origin,dest)].append((depart,arrive,int(price)))
```

- Getminutes 함수
  - 특정 일시에서 경과한 시간을 분단위로 계산
  - 비행시간, 대기시간 계산

```
def getminutes(t):
    x=time.strptime(t,'%H:%M')
    return x[3]*60+x[4]
```

2. 해답 표현하기

- 숫자리스트
  - 가장 흔한 표현방식
  - 예: [1,4,3,2,7,3,6,3,2,4,5,3]
    - 각 사람마다 출발/도착 비행편 의미
      - 리스트 길이 : 사람 수의 2배
    - Seymour가 보스턴에서 라구아디아으로 가는 2번째 비행편을 타고 가서 다시 그날 보스턴으로 돌아오는 5번째 비행편을 타는 것을 의미

– Printschedule 함수

• 선택한 탑승편을 테이블 형식으로 출력

```
def printschedule(r):
  for d in range(len(r)/2):
    name=people[d][0]
    origin=people[d][1]
    out=flights[(origin,destination)][int(r[d])]
    ret=flights[(destination,origin)][int(r[d+1])]
    print '%10s%10s %5s-%5s $%3s %5s-%5s $%3s' % (name,origin,
                                                  out[0],out[1],out[2],
                                                  ret[0],ret[1],ret[2])
```

```
>>> from optimization import *
>>> s=[1,4,3,2,7,3,6,3,2,4,5,3]
>>> printschedule(s)
   Seymour        BOS  8:04-10:11 $ 95 12:08-14:05 $142
    Franny        DAL 12:19-15:25 $342 10:51-14:16 $256
     Zooey        CAK 10:53-13:36 $189  9:58-12:56 $249
      Walt        MIA  9:15-12:29 $225 16:50-19:26 $304
     Buddy        ORD 16:43-19:00 $246 10:33-13:11 $132
       Les        OMA 11:08-13:07 $175 15:07-17:21 $129
```

3. 비용 함수

- Cost function
  - 최적화를 사용해서 문제를 푸는 핵심
  - 선택하기 가장 어려운 부분
  - 최적화 알고리즘은 비용 함수의 값이 가장 작은 입력 집합 선택하는 것이 목적
- 단체 여행에서 측정될 수 있는 지표
  - 가격 : 비행 티켓 가격
  - 여행시간 : 비행기 안에서 머문 총 시간
  - 대기시간 : 다른 동반자가 도착하기를 기다린 시간
  - 출발시간 : 이른 비행편을 놓칠 경우 추가 비용 부담
  - 자동차 렌탈기간 : 당일 반납하면 적은 비용

- 중요한 요소가 무엇인지 판단해야
  - 비용 부가하는 변수들 선택했으면, 그들을 결합해서 하나의 숫자로 만드는 법 결정
- Schedulecost 함수
  - 여행의 전체 비용과 가족 구성원들이 공항에서 체류한 시간, 자동차 렌탈 예상 시간보다 늦게 반납하는 경우 추가 비용 부담 등 고려
  - 다른 비용을 추가하거나, 돈/시간의 상대 중요도 미세 조정

```
>>> schedulecost(s)
5285
```

  - 이론상 모든 조합을 시도해 보면 정답을 얻을 수 있지만, 아주 오랜 시간 걸림

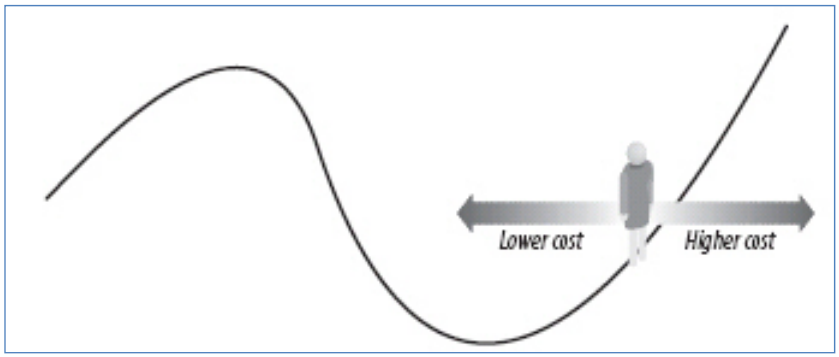
4. 무작위 검색 (Random Searching)

- Random scheduling
  - 좋은 최적화 기법은 아님
  - 다른 알고리즘이 잘 동작하는지 기준선으로 사용
- Randomoptimize 함수
  - Domain 인자
    - 튜플의 리스트: 각 변수의 최소/최대값
    - 모든 사람에 대해 출발/도착 비행편 각각 9개
  - Costf 인자
    - 비용 함수 : schedulecost 함수 사용
  - 무작위로 1000개 추정값 생성
    - 최소값 리턴
    - 최고는 아니지만 만족스러운 성능

```
>>> from optimization import *
>>> domain = [(0,8)]*(len(people)*2)
>>> domain
[(0, 8), (0, 8), (0, 8), (0, 8), (0, 8), (0, 8), (0, 8), (0, 8),
(0, 8), (0, 8), (0, 8), (0, 8)]
>>> s = randomoptimize(domain, schedulecost)
>>> s
[5.0, 3.0, 2.0, 3.0, 4.0, 4.0, 5.0, 0.0, 0.0, 5.0, 4.0, 7.0]
>>> s = randomoptimize(domain, schedulecost)
>>> s
[5.0, 4.0, 4.0, 4.0, 4.0, 4.0, 5.0, 8.0, 7.0, 5.0, 0.0, 7.0]
>>> schedulecost(s)
3222
>>> printschedule(s)
   Seymour       BOS 13:40-15:37 $138 12:08-14:05 $142
    Franny       DAL 12:19-15:25 $342 12:20-16:34 $500
     Zooey       CAK 12:08-14:59 $149 12:01-13:41 $267
      Walt       MIA 12:05-15:30 $330 12:37-15:05 $170
     Buddy       ORD 12:44-14:17 $134 12:08-14:47 $231
       Les       OMA 12:18-14:56 $172 14:05-15:47 $226
```

5. 언덕등반 (Hill Climbing)

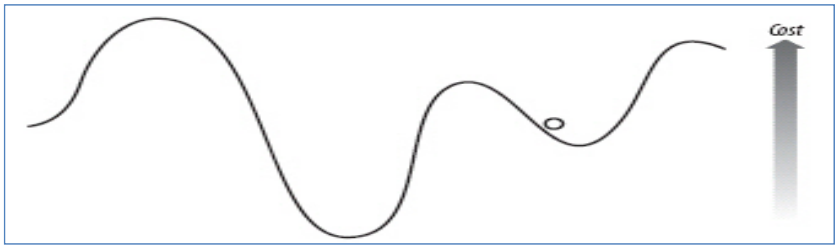
- 무작위 검색
  - 이미 발견했던 좋은 해법들의 장점 활용 못함
- 언덕등반
  - 무작위 해법으로 시작해서 더 좋은 해답을 찾아 이웃 해답들을 살펴보는 기법
  - 언덕에서 최저점 찾기



- 예제에 적용
  - 무작위 일정에서 시작해서 근처의 모든 일정을 찾음
  - 즉, 바로 이전이나 이후 비행편 검색
- Hillclimb 함수
  - 주어진 도메인에서 무작위로 초기 해답 생성
  - 현재 해답의 모든 이웃 찾음
    - 1만큼 늘린 항목 / 1만큼 줄인 항목 비교
    - 이 중 더 좋은 것이 새로운 해답
    - 변화 없을 때까지 반복

```
>>> s = hillclimb(domain, schedulecost)
>>> schedulecost(s)
2789
```

- 무작위 검색 방법에 비해
  - 빠르게 수행
  - 보통 더 좋은 해답 생성
- 언덕등반 방식의 문제점
  - 국소 최저(local minimum) 함정에 빠질 가능성 존재
  - 주변에서는 좋은 해답이지만 전체적으로 해답(global minimum)은 아님 → random-restart hill climbing



6. Simulated Annealing

- 물리학에서 영감을 받은 최적화 기법
  - Annealing
    - 합금을 가열한 후 천천히 냉각하는 과정
- 국소 최소점 문제 회피하려는 시도
- Annealingoptimize 함수
  - 무작위 해답부터 시작
  - 온도를 표현하는 변수 사용: 고온부터 시작해서 점차로 낮춰감
  - 좀 더 좋은 방향으로 항상 이동해야
  - 시작 과정 근처에서는 더 좋은 해답을 얻기 위해 더 나쁜 해답을 선택할 수도
  - 큰 비용을 가진 해답을 선택할 확률

$$p=e^{((-highcost-lowcost)/temperature)}$$

- 온도가 거의 0이 될때까지 매번 온도에 냉각 속도를 곱하면서 루프 돔
- 옵션 함수인자
  - T : 온도, cool : 냉각속도, step : 변경할 방향

```
>>> s = annealingoptimize(domain, schedulecost)
>>> schedulecost(s)
2912
```

- 줄어든 비용을 유지하면서 전체 대기 시간을 줄이는 결과
- 실행할 때마다 다른 결과
  - 더 나쁜 결과 나올 수도

7. 유전자 알고리즘

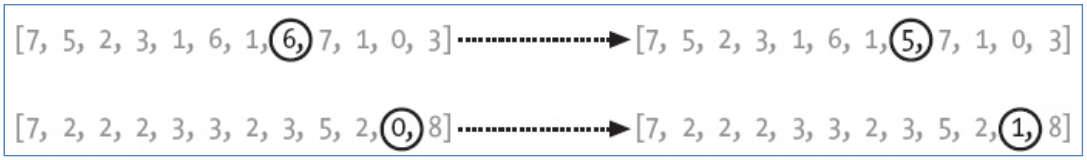
- Genetic Algorithms
  - 자연의 섭리 본뜸
  ① 개체군(population)이란 무작위 해답들 생성
  ② 최적화 단계마다 전체 개체군별 비용 함수 계산하여 해답 순위 목록 만듦
  ③ 다음세대(next generation)로 알려진 새로운 개체군 만듦
    ① 현재 개체군의 최상위 해답(비율)을 새로운 개체군에 추가
      - 엘리트 선발(elitism)
    ② 새로운 개체군의 나머지는 최고 해법을 수정하여 완전히 새로운 해법 구성
  ④ 이 과정을 정해진 횟수만큼 또는 개선이 없을 때까지 반복

Solution	Cost
[7, 5, 2, 3, 1, 6, 1, 6, 7, 1, 0, 3]	4394
[7, 2, 2, 2, 3, 3, 2, 3, 5, 2, 0, 8]	4661
...	...
[0, 4, 0, 3, 8, 8, 4, 4, 8, 5, 6, 1]	7845
[5, 8, 0, 2, 8, 8, 8, 2, 1, 6, 6, 8]	8088

– 해답 변경 방식

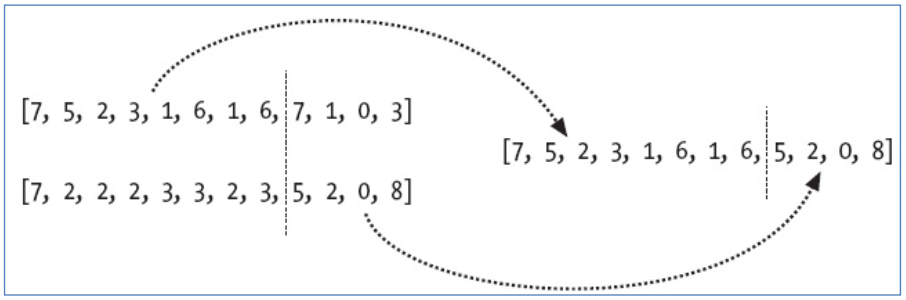
1. 돌연변이(mutation)

- 소량의 간단하고 무작위적인 변화를 기존 해답에 적용



2. 교배(crossover)와 번식(breeding)

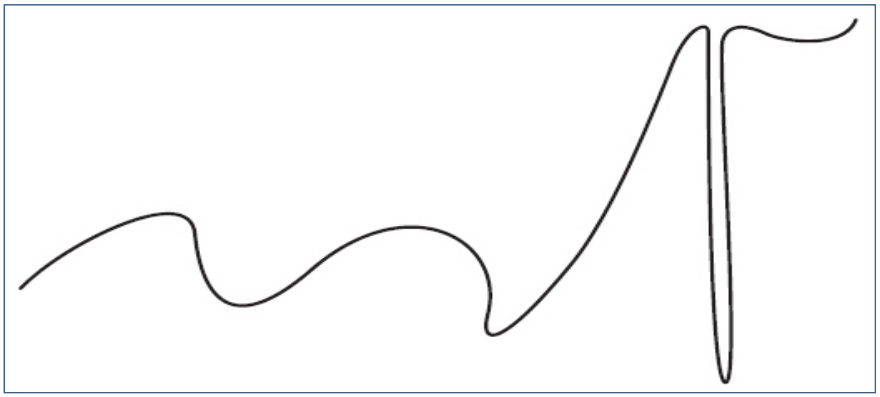
- 최적 해답 2개를 골라 어떤 방식으로 그 둘을 결합
- 교배 : 한 해답에서 무작위로 몇 개 요소 선택하고, 다른 해답에서 나머지 선택



- Geneticoptimize 함수
  - 함수 인자
    - Popsize : 개체군의 크기
    - Mutprob : 개체군의 새로운 멤버에 교배보다 돌연변이가 발생할 확률
    - Elite : 좋은 해답으로 간주되어 다음세대로 전달될 개체군의 비율
    - Maxiter : 진화시킬 세대 수

```
>>> from optimization import *
>>> domain = [(0, 8)]*(len(people)*2)
>>> s = geneticoptimize(domain, schedulecost)
4034
4034
3842
3802
3620
  ⋮
2804
2804
2804
>>> printschedule(s)
    Seymour        BOS   8:04-10:11 $ 95   6:39- 8:09 $ 86
     Franny        DAL   6:12-10:22 $230   7:57-11:15 $347
       Zooey       CAK   8:27-10:45 $139   8:19-11:16 $122
        Walt       MIA   7:34- 9:40 $324   8:23-11:07 $143
       Buddy       ORD   8:25-10:34 $157   7:50-10:08 $164
         Les       OMA   7:39-10:24 $219   8:04-10:59 $136
```

- 최적화 기법들은 문제에 따라 얼마나 잘 동작할지가 달라짐
  - 시뮬레이티드 어닐링, 유전자 최적화 등은
    - 최적 해답이 다른 좋은 해답 근처에 있는 경우에 잘 동작
    - 최적화에 나쁜 예 (아래 그림)
      - 대부분 그림의 왼쪽에 있는 지역 최소점들 하나에 정착



8. 비행편 검색 실제

- 앞에서 사용한 최적화 방법을 실제 비행편 데이터로 시도
- 카약(Kayak) 사이트
  - 비행편 검색 API 제공
    - 하루 검색 횟수 제한
  - 샘플 데이터와 차이점
    - 주요 도시 간 비행편 9편 이상 존재
  - 카약 API
    - 개발자 키 신청
      - http://www.kayak.com/labs/api/search/
      - 계정 만들면 자동으로 생성

- 카약용 파이썬 API는 없지만,
- urllib2와 xml.dom.minidom 표준 파이썬 패키지 이용하여 검색
- Minidom 패키지
  - 표준 파이썬 배포판에 포함
  - 많은 웹 사이트들이 XML 인터페이스를 통해 정보를 외부에 제공
  - XML 문서를 객체 트리로 다루는 DOM 인터페이스 구현
    - XML을 담은 문자열이나 열린 파일을 받아서 쉽게 정보 추출 가능한 객체를 리턴
    - getElementsByTagName(name)
      - 전체 문서에서 name과 일치하는 태그 요소들을 찾아 DOM 노드 리스트 리턴
    - firstChild
      - 이 객체의 1번째 자식 노드 리턴
    - data
      - 이 객체와 연계된 데이터 리턴

```
>>> from xml.dom.minidom import *
>>> dom = parseString('<data><rec>Hello!</rec></data>')
>>> dom
<xml.dom.minidom.Document instance at 0x0137FDA0>
>>> r = dom.getElementsByTagName('rec')
>>> r
[<DOM Element: rec at 0x11e1990>]
>>> r[0].firstChild
<DOM Text node "Hello!">
>>> r[0].firstChild.data
u'Hello!'
```

- 비행편 검색
  - Getkayaksession 함수
    - 개발자 키를 사용해서 세션 ID를 얻음
  - Flightsearch 함수
    - 비행편 검색
  - Flightsearchresults 함수
    - 더 이상 결과가 없을 때가지 요청

```
>>> from kayak import *
>>> sid = getkayaksession()
<?xml version="1.0" ?><ident>
  <uid>42021984</uid>
  <sid>20-CEP$oTPdmKUTDhpgSur6</sid>
  <token>tzVqc4JZ6Fm4fPbU_UTECQ</token>
  <error/>
</ident>
>>> searchid = flightsearch(sid, 'BOS', 'LGA', '12/01/2009')
<?xml version="1.0" ?><search>
  <url>![CDATA[http://www.kayak.com/s/basic?apimode=1&amp;searchid=Ti4IBt&amp;c=
10]]</url>
  <searchid>Ti4IBt</searchid>
</search>
>>> f = flightsearchresults(sid, searchid)
>>> len(f)
302
>>> f[0:3]
[(u'14:00', u'19:30', 15.0), (u'05:50', u'11:57', 15.0), (u'05:50', u'13:57', 15
.0)]
```

- 비행편이 가격순으로 리턴됨
  - 같은 가격인 경우에는 시간 순으로 리턴
- createschedule 함수
  - 비행 스케줄 생성

```
>>> f = createschedule(people[0:2], 'LGA', '12/01/2009', '12/03/2009')
                    ⋮
>>> flights = f
>>> domain = [(0, 30)] * len(f)
>>> s = geneticoptimize(domain, schedulecost)
```

9. 선호도 최적화

- 최적화 기법으로 풀기 위한 조건
  - 문제가 정의된 비용 함수를 가지는가?
  - 유사한 해답이 유사 결과를 내는가?
- 적용 가능 예
  - 제한된 자원을 그에 대해 선호를 표현한 사람들에게 분배하여 가능한 모두 행복하게 하는 방법

- 학생 기숙사 최적화
  - 학생들의 1, 2지망을 반영해서 기숙사 배치하는 문제
  - 유사 문제
    - 집안일 가족 할당, 프로젝트에서 버그를 개발자에 할당 등
  - 조건 : 학생 10명, 5개 기숙사, 기숙사별 2개 공간
  - Dorm.py

```
import random
import math

# The dorms, each of which has two available spaces
dorms=['Zeus','Athena','Hercules','Bacchus','Pluto']

# People, along with their first and second choices
prefs=[('Toby', ('Bacchus', 'Hercules')),
       ('Steve', ('Zeus', 'Pluto')),
       ('Karen', ('Athena', 'Zeus')),
       ('Sarah', ('Zeus', 'Pluto')),
       ('Dave', ('Athena', 'Bacchus')),
       ('Jeff', ('Hercules', 'Pluto')),
       ('Fred', ('Pluto', 'Athena')),
       ('Suzie', ('Bacchus', 'Hercules')),
       ('Laura', ('Bacchus', 'Hercules')),
       ('James', ('Hercules', 'Athena'))]
```

  - 이 예제는 대략 100,000개의 가능한 해답 존재
  - 만약 기숙사 방이 4개라면 수조 개의 해답 존재
  - 해답 표현 방법
    - 숫자 리스트
      - 각 학생이 들어갈 기숙사 번호로 리스트 구성하는 방법
      - 해답이 각 기숙사에 2명의 학생만 들어가도록 강제할 수 없음
    - 순서대로 학생에게 방 할당
      - 첫 학생은 10개 중 한 방
      - 두 번째 학생은 9개 중 한 방
      - 검색을 위한 domain에 위 제약 반영

```
# [(0,9),(0,8),(0,7),(0,6),...,(0,0)]
domain=[(0,(len(dorms)*2)-i-1) for i in range(0,len(dorms)*2)]
```

– Printsolution 함수

```
>>> printsolution([0,0,0,0,0,0,0,0,0,0])
Toby Zeus
Steve Zeus
Karen Athena
Sarah Athena
Dave Hercules
Jeff Hercules
Fred Bacchus
Suzie Bacchus
Laura Pluto
James Pluto
>>> printsolution([2,0,0,0,0,0,0,0,0,0])
Toby Athena
Steve Zeus
Karen Zeus
Sarah Athena
Dave Hercules
Jeff Hercules
Fred Bacchus
Suzie Bacchus
Laura Pluto
James Pluto
```

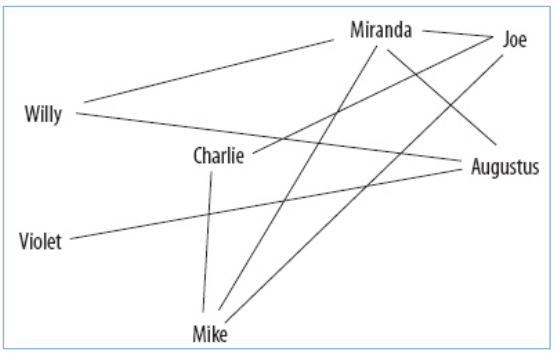
- 비용 함수
  - 비용이 0인 답이 완벽한 해답
    - 즉 얼마나 정답에 가까운지를 알 수 있다는 뜻
    - 최적화 알고리즘 적용 가능
  - Dormcost 함수
    - 첫 번째로 선택한 기숙사 배정시 : 0
    - 두 번째로 선택한 기숙사 배정시 : 1 증가
    - 선택하지 않은 기숙사 배정시 : 3 증가

- 최적화 실행

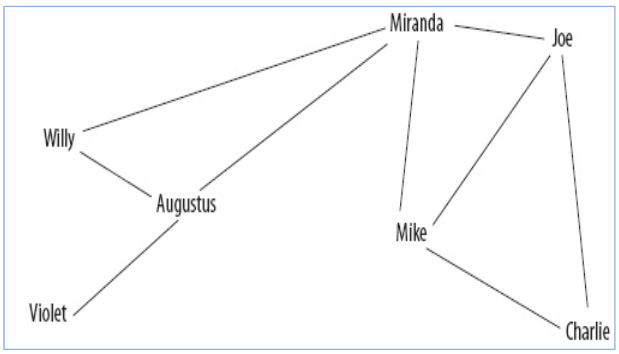
```
>>> from optimization import *
>>> from dorm import *
>>> domain=[(0,(len(dorms)*2)-i-1) for i in range(0,len(dorms)*2)]
>>> s = randomoptimize(domain, dormcost)
>>> dormcost(s)
7
>>> geneticoptimize(domain, dormcost)
9


>>> printsolution(s)
Toby Bacchus
Steve Pluto
Karen Athena
Sarah Zeus
Dave Bacchus
Jeff Hercules
Fred Pluto
Suzie Hercules
Laura Zeus
James Athena
```

10. 네트워크 시각화

- 소셜 네트워크
  - 예) 마이스페이스(MySpace), 페이스북(FaceBook), 링크드인(LinkedIn)
- 배치문제
  - 많은 사람들과 그들 간의 연결을 시각화하여 네트워크를 그릴 때 사람(이름)을 어디에 놓을 것인지 결정하는 문제



혼란스런 네트워크 배치



깔끔한 네트워크 배치

- Socialnetwork.py
  - 소셜 네트워크 정보 표시
    - 누가 누구의 친구인지에 관한 사실정보 목록

```
people=['Charlie','Augustus','Veruca','Violet','Mike','Joe','Willy','Miranda']

links=[('Augustus', 'Willy'),
       ('Mike', 'Joe'),
       ('Miranda', 'Mike'),
       ('Violet', 'Augustus'),
       ('Miranda', 'Willy'),
       ('Charlie', 'Mike'),
       ('Veruca', 'Joe'),
       ('Miranda', 'Augustus'),
       ('Willy', 'Augustus'),
       ('Joe', 'Charlie'),
       ('Veruca', 'Augustus'),
       ('Miranda', 'Joe')]
```

  - Mass & Spring 알고리즘
    - 해석이 편한 네트워크 도표 생성
    - 각 노드는 다른 노드에 대해 미는 힘을 지속적으로 발휘하여 떨어뜨리려 하고,
    - 링크는 연결된 노드를 가깝게 끌어당기려 하는 물리학을 모방한 것
    - 선 교차를 방지할 방법은 없음

- 교차선 세기
  - 모드 노드가 x, y축 가지므로 모든 노드에 대한 좌표로 해답 목록 구성
  - 예
    - Sol = [120, 200, 250, 125, …]
      - Charlie는 (120, 200)에, Augustus는 (250, 125)에 …
  - 비용 함수
    - 교차하는 선의 수를 계산
    - Crosscount 함수
  - Domain
    - 각 좌표의 범위

```
>>> from optimization import *
>>> from socialnetwork import *
>>> domain=[(10,370)]*(len(people)*2)
>>> sol = randomoptimize(domain, crosscount)
>>> crosscount(sol)
0
>>> sol
[64.0, 71.0, 255.0, 296.0, 257.0, 191.0, 355.0, 161.0, 130.0, 80.0, 193.0, 40.0,
 128.0, 164.0, 33.0, 306.0]
>>> sol = randomoptimize(domain, crosscount)
>>> crosscount(sol)
0
>>> sol = annealingoptimize(domain,crosscount,step=50,cool=0.99)
>>> crosscount(sol)
4
>>> sol
[300, 306, 203, 271, 370, 253.0, 145.0, 236, 356.0, 307, 132.0, 10, 30, 211.0, 1
36.0, 334]
```

- 네트워크 그리기
  - Drawnetwork 함수
    - 3장에서 사용한 파이썬 이미지 라이브러리 필요
    - 이미지 생성
    - 사람들 간 연결 그리고, 마지막 사람 이름 출력

```
>>> drawnetwork(sol)
```

    - 선의 각도나 근접한 노드와 같은 것에 대한 벌점 고려는 하지 않았음
      - Crosscount 함수 끝부분에 관련 코드 추가하여 구현

11. 다른 가능성들

- 이 장에서 가장 중요한 단계
  - 해답 표현
  - 비용 함수 결정
- 위 단계 가능하면 최적화 기법 적용 가능
- 응용분야는?