2019 年度未踏ターゲット事業

数式処理を用いたアニーリングマシン向けプログラミング言語およびその処理系の開発
— 専門的な知識がないユーザーにもアニーリングを —

小津 泰生 (名古屋大)

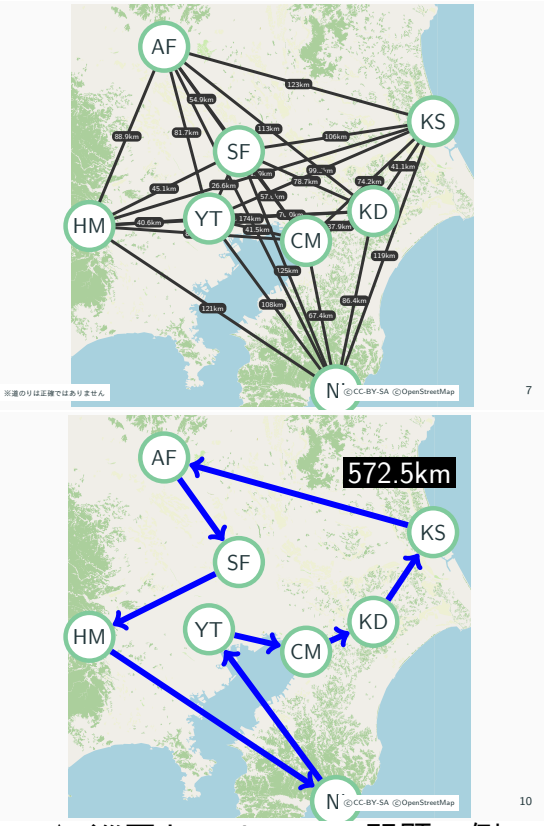
近年、組合せ最適化問題を効率よく解くための方法としてアニーリングマシンの利用が注目されている。

しかしアニーリングマシンを利用して問題を解くためには、問題をハミルトニアンの形式で記述し、ハミルトニアンを高々2次の式に変形し、QUBOを生成する必要がある。

さらに、アニーリング後も解が問題の条件を満たすことを確認し、満たさない場合はパラメタサーチを行ってアニーリングをやり直す必要がある。このように複雑な操作が必要になるため、専門的な知識のない人にとってアニーリングマシンを利用することの敷居が高くなっている。

$$H = A_1 \sum_{v\in V}\left(1-\sum_{j=1}^{N} x_{v,j}\right)^2 + A_2 \sum_{j=1}^{N}\left(1-\sum_{v\in V} x_{v,j}\right)^2 + A_3 \sum_{(u,v)\notin E}\sum_{j=1}^{N} x_{u,j}x_{v,j+1} + A_4 \sum_{(uv)\in E} W_{uv}\sum_{j=1}^{N} x_{u,j}x_{v,j+1}$$

▲ 巡回セールスマン問題のハミルトニアン



▲ 巡回セールスマン問題の例

2019 年度未踏ターゲット事業

数式処理を用いたアニーリングマシン向けプログラミング言語およびその処理系の開発
— 専門的な知識がないユーザーにもアニーリングを —

小津 泰生 (名古屋大)

```
const cities := [@NS, @KD, @NI, @CM, @HM, ...]
type order := 1 .. len(cities)
order cities
define distance(@A, @B) := (@A <-> @B || (@A == 1,
    @B == len(cities)) || (@A == len(cities), @B == 1))
different(cities)
minimize {
    41.1 : distance @KS @KD
     119 : distance @KS @NI
    74.2 : distance @KS @CM
     139 : distance @KS @HM
    86.4 : distance @KD @NI
    37.9 : distance @KD @CM
     110 : distance @KD @HM
    67.4 : distance @NI @CM
     121 : distance @NI @HM
    81.1 : distance @CM @HM ...
}
solve(cities)
print(cities)
```

cities の定義

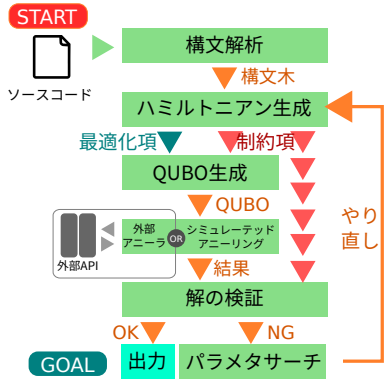
distance の定義

最小化

※最新版とは仕様が異なります

9

▲ 巡回セールスマン問題の記述例

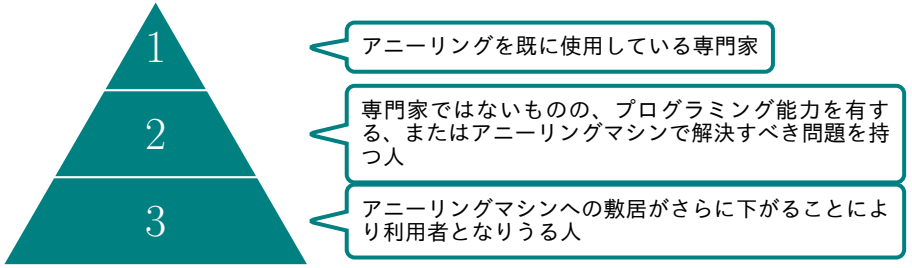


▲ 処理系の内部構造

本プロジェクトでは、このような煩雑な処理を自動で行う処理系を実装した。これによりアニーリングマシンを用いたアプリケーションを開発する敷居が下がり、アニーリングに関する専門知識がなくとも、プログラミングの経験があればアニーリングマシンを活用できるようになった。

また、アニーリングマシンを活用する人が増えることにより、アニーリングコミュニティが活性化し、さらなる技術革新が進むことが期待される。

本プロジェクトの成果物は、https://github.com/SpoonQ で公開している。



▲ アニーリングマシンの潜在的利用者