

アニーリングを用いた効率的な 制約充足問題ソルバの実装

小津 泰生 (応用・実用化枠)

担当 PM: 棚橋耕太郎

2021年2月11日

発表の流れ

1 制約充足問題の解き方について

本プロジェクトの成果物「SpoonQ」の位置づけについて(メインプロジェクト)

2 プログラマーの方向け情報

Rust 言語から最適化問題及びアニーリングを利用するライブラリについて(サブプロジェクト)

3 皆様へのメッセージ

今後、アニーリングがどのように発展してゆくか、について

公式サイト(スライド公開中)

<https://spoonq.github.io/>

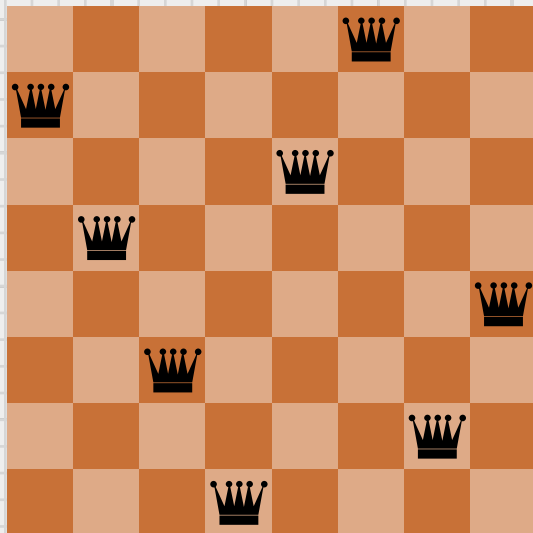


制約充足問題とは

与えられた条件すべてを満たす**実行可能解**を求める問題。



塗り分け問題



8-Queen 問題

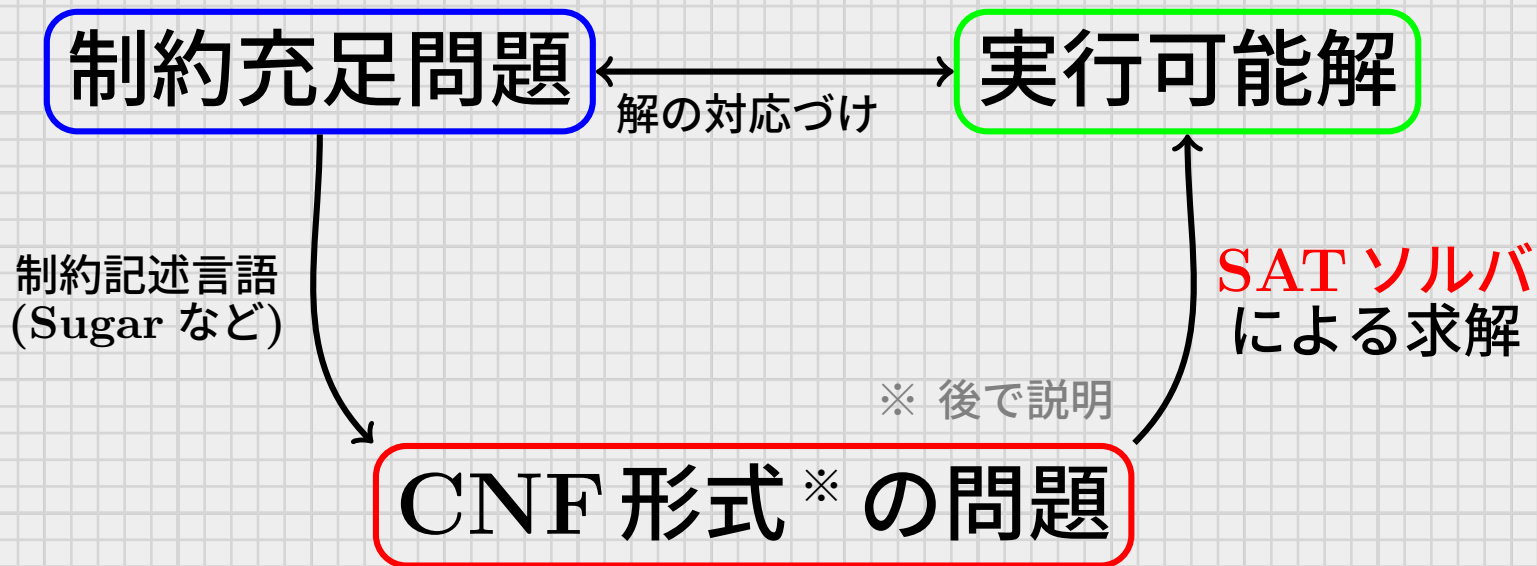
5	3			7				
6			1	9	5			
	9	8					6	
8				6			3	
4			8		3		1	
7				2			6	
	6					2	8	
			4	1	9		5	
				8			7	9

ペンシルパズル

⇒ 世の中の様々な問題は**制約充足問題**で表される

SAT ソルバについて

制約充足問題を解く方法...SAT ソルバを用いる



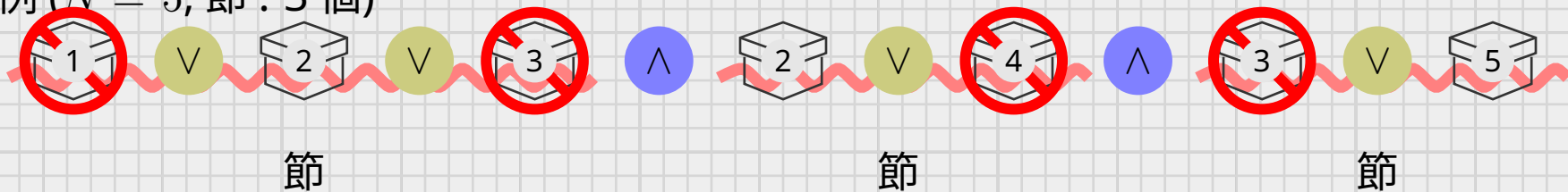
CNF 形式とは

決定すべき二値変数 $x_1 \sim x_N$ が存在する。 ($x_i \in \{ \text{true}, \text{false} \}$)

これらの変数 x_i もしくはその否定 $\neg x_i$ (項という) の

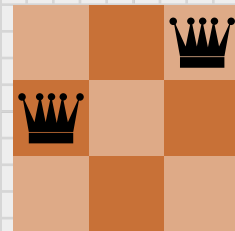
- \vee : OR (論理和) をとったもの (節という) の
- \wedge : AND (論理積) をとったもの

例 ($N = 5$, 節: 3 個)

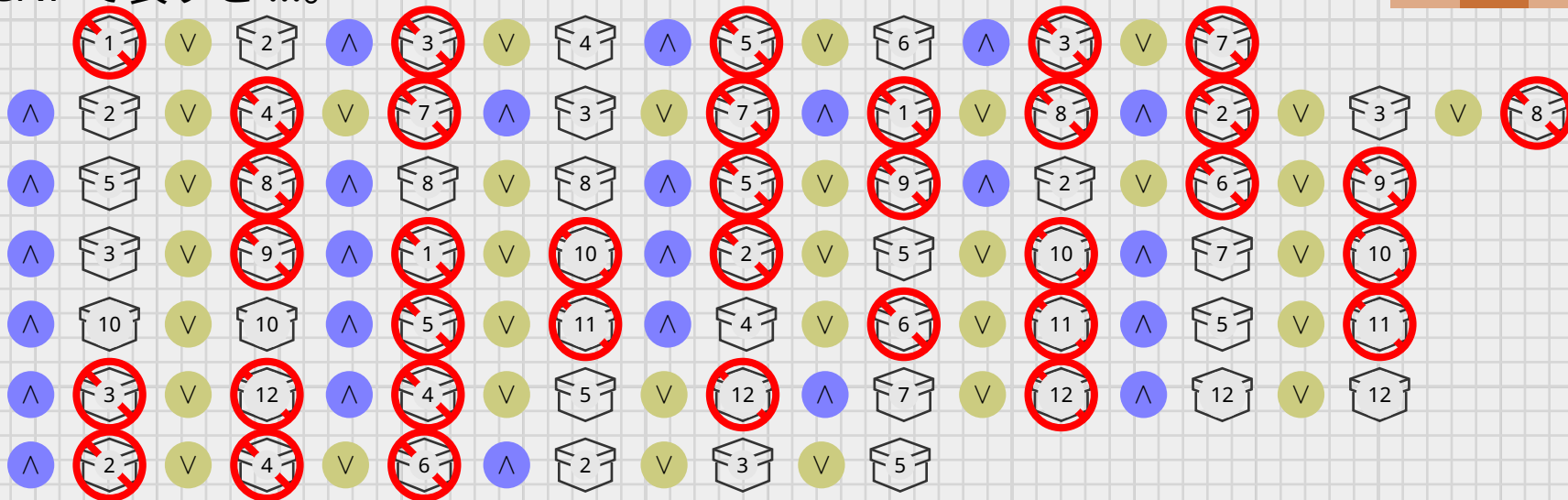


例：3-Queen 問題

3つの Queen を 3×3 の盤に配置する。ただし、同じ行・列・対角線に複数の Queen を配置してはならない。



CNF で表すと ...。



SAT ソルバによる解法



1. **選択** ↑



2. **推論** ↑

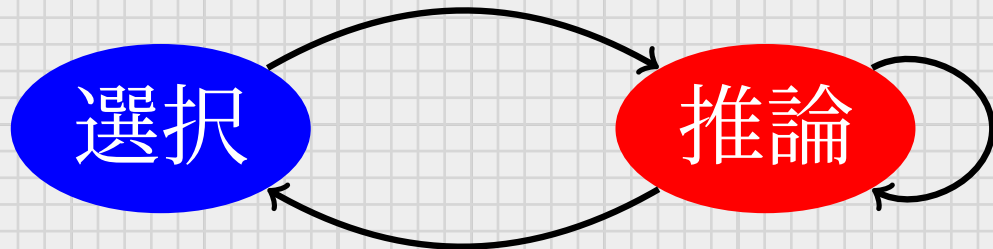


3. **推論** ↑






矛盾を検出すると、直前の**選択**まで**巻き戻す** (巻き戻せなければ解なし)

SAT ソルバのアルゴリズム



SAT ソルバは**すべての解を探索**するため、(存在する場合) 必ず解を見つけ出す反面、本質的に**非常に時間がかかる**

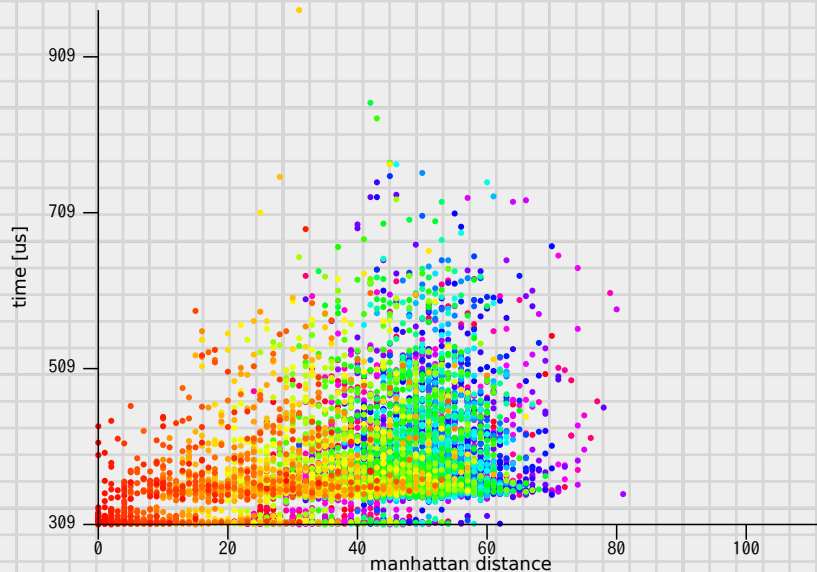
効率を上げるため、**選択**フェーズにおいて

- どの未確定変数  を選択するか (→ **選択の回数を減らす**)
- その変数に  と  のどちらを代入するか (→ **巻き戻しを減らす**)

(**ヒューリスティクス**) が重要

SAT ソルバの前処理の必要性

SAT ソルバのヒューリスティクスとして、正しい解から距離 d (横軸) の状態を与えたときの**実行時間**



理想的なヒューリスティクスを与えることで、
SAT ソルバの実行時間が短くなる

SpoonQ とは

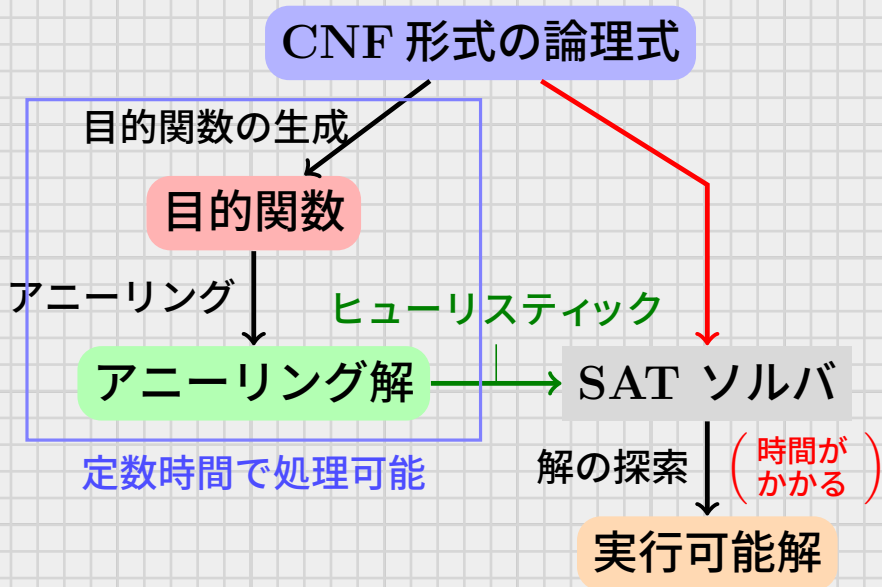
本プロジェクトの目的

アニーリングを用いた効率的な 制約充足問題ソルバの実装

ここでは SAT ソルバを扱う

アニーリングと SAT ソルバのアイデアを組み合わせる

SpoonQ のアルゴリズム



アニーリングは**定数時間**で実行できる反面、解が**実行可能解**であることが**保証されない**。

SAT ソルバは (存在する場合) 実行可能解を確実に見つけ出す。

⇒ アニーリングの結果を**ヒューリスティック**として与えることで SAT ソルバによる求解の**効率化**を目指す

SpoonQ の紹介、デモ

未踏 TG 2020

A programming language for annealing machine.


commits 1 branch 0 packages 0 releases 1 contributor

Branch: New pull request Create new file Upload files Find file Clone or download

yasuo-ozu

- resources
- src
- test
- gitignore
- Cargo.lock
- Cargo.toml
- README.md
- rustfmt.toml

README.md

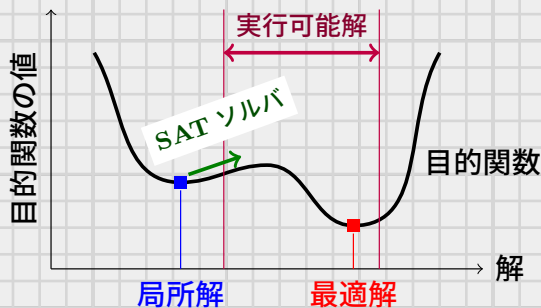


SpoonQ

A programming language for annealing machine.

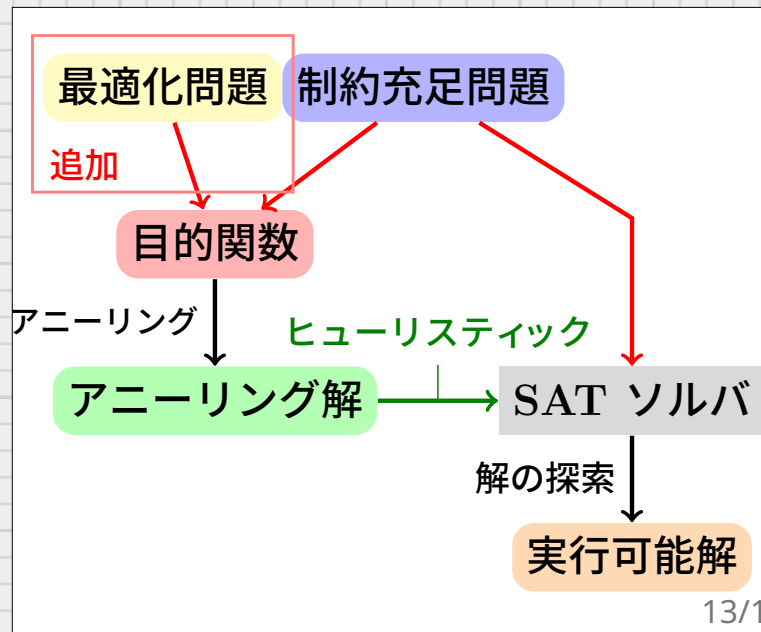
制約充足問題 → 組合せ最適化問題への拡張

組合せ最適化問題 = **制約充足問題** + **最適化問題**
 (今年度のプロジェクト)

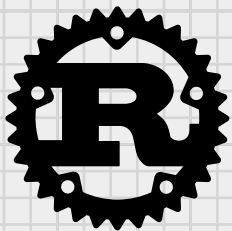


最適化問題 ... 目的関数の値が**最小値**をとる
最適解を求める問題

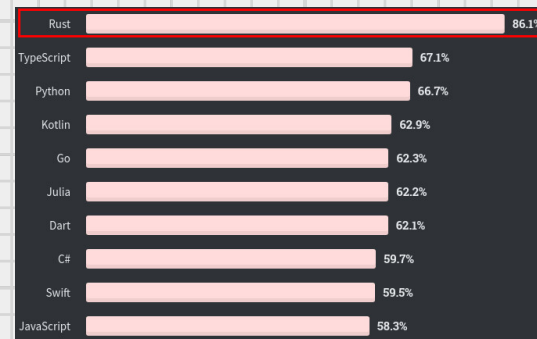
アニーリングでは近似的に**局所解**が求まる
 ⇒ **SAT ソルバ**を用いて**実行可能解**を求める



Rust 言語について



Rust 言語 ... 新しい (2010 年) 汎用プログラミング言語。C++ を置き換える存在
 採用例: AWS, Firefox, Dropbox, Fastly, npmjs.com, Cookpad, Dwango, ...



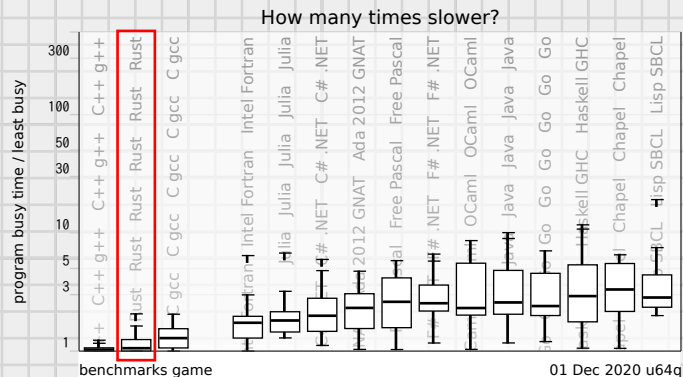
プログラマーに**最も愛されている**言語 (2020 年 Stack Overflow 調べ)

Python ライブラリも多く移植 (?) されている
 numpy, scipy, sympy, matplotlib, Jupyter, TensorFlow, ...

安全かつ最も高速な言語 (Tier 1) の一つ

C++ より遅いわけではない

図の出典: The Computer Language Benchmarks Game



現在開発中のライブラリ

Rust 言語向けの以下のライブラリを**オープンソース**で開発している。(これらは現在の **SpoonQ** でも使用している)

RustQUBO

- **Rust 言語**上で**目的関数**を記述する
- **目的関数**から **QUBO** を生成する
- **annealers** を用いて **QUBO** を解く
- 与えられた**制約**が満たされているか確認する
- 制約が満たされない場合、自動で**パラメタチューニング**を行う





類似のライブラリ



- PyQUBO (Python), ThreeQ.jl(Julia)





annealers


- **シミュレーテッドアニーリング**又は**アニーリング API**を用いてアニーリングを行う
- 複数の**アニーリングマシン**を同じインターフェースで扱える
- D-Wave, Hitachi (Annealing Cloud Web), Fujitsu, Toshiba 等に対応予定
- **問題分割**をサポート (予定)



















特徴... QUBO 生成アルゴリズムに**石川の方法**を使用している

例:     (4 次の積) を高々 2 次に変換

既存の方法 (PyQUBO 等)... 追加のビット:  ,  , 制約: **2** 個

$$0 = \min_{\substack{u \\ v}} \left(2 - 2 \text{  } - 2 \text{  } \right) = \min_v \left(3 - 2 \text{  } - 2 \text{  } \right)$$

石川の方法 (RustQUBO)... 追加のビット:  , 制約: **0** 個

$$\begin{aligned} & -2 \text{  } \left(\text{  } + \text{  } + \text{  } + \text{  } \right) + 3 \text{  } + \text{  } \text{  } \\ & + \text{  } \text{  } + \text{  } \text{  } + \text{  } \text{  } + \text{  } \text{  } + \text{  } \text{  } \end{aligned}$$

- アルゴリズムが制約を増やさない → **アニーリングのやり直しが減る**
- 追加のビット (Ancilla) が少なくて済む → **アニーリングマシンに乗せやすい**

RustQUBO の使用例

```
1 let exp = -10_i32 * Expr::Binary(1) + 5_i32 * Expr::Binary(2) + 12_i32;  
2 let compiled = exp.compile();  
3 let solver = SimpleSolver::new(&compiled);  
4 let (c, sol) = solver.solve().unwrap();  
5 println!("{}", sol); // 2, {1: true, 2: false}
```

実数型

Rust 標準のすべての実数型 (i8, i32, i128, f32, f64 など) が利用できる。
使用する Solver (Annealer) の **ビット深度** に対応

変数

- Expr::Binary(...)
- Expr::Spin(...)

が使える。... 部 (ラベル) には数値や文字列等、好きな型を入れられる。

その他

- 解のエネルギー、部分場の計算
- Constraint
- WithPenalty
- Placeholder

annealers について

複数のアニーリングマシンを**同じインターフェース**で扱えるようにするため、アニーリングマシンや解くべき問題を一般化する。

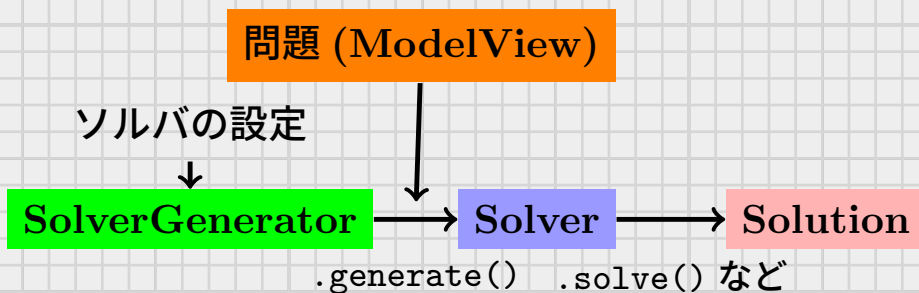
問題について

Order

- Quadric... 高々 2 次
- HighOrder... N 次

Node

- DiscreteNode... 3 つ以上の値を取る変数
- SingleNode... 真偽 2 つの値をとる変数
- Spin / Binary / TwoVal



Real

ビット深度を表す。i8, i32, f64 など

ソルバについて

- ClassicalSolver... 古典コンピュータ上で動作する。与えられた乱数ジェネレータを用いて処理を行う
- AsyncSolver... 他のコンピュータと通信して処理を行う。
- UnstructuredSolver... 全結合かつ変数の番号の欠番がないソルバ
- UnsizedSolver... 変数の数に制限がないソルバ

メッセージ

プログラマーの方へ

アニーリング向けツールはまだ成熟し切っていない

⇒ **研究・実装の両面で参入する余地がある**

共にアニーリング業界を盛り上げましょう！

アニーリングを利用している (or 興味がある) 方へ

開発中の **RustQUBO** や **annealers** の採用を検討いただければ幸いです

SAT ソルバに興味がある方へ

ぜひ **SpoonQ** を使ってみてください