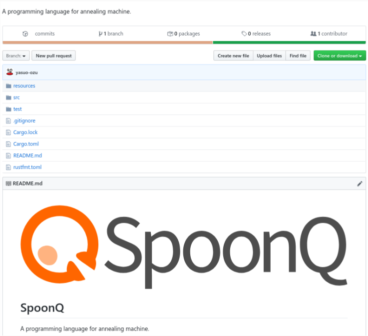
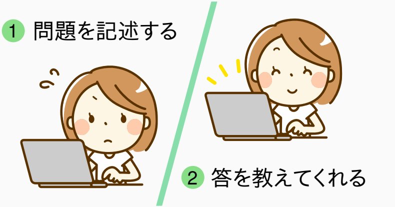
昨年度のプロジェクト

2019 年度のプロジェクト・・・「SpoonQ」





初心者のユーザーがアニーリングに関する専門的な技術を学ぶこと無く、問題をアニーリングによって解くことができる

本年度プロジェクト

2020 年度のプロジェクト

アニーリングを用いた効率的な

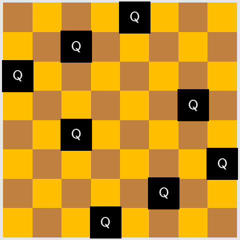
制約充足問題ソルバの実装

制約充足問題

制約充足問題

変数と制約関数が与えられた時、制約関数を満たすような変数の値を求める問題。

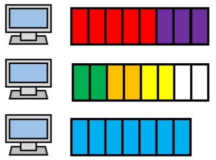
N クイーン問題



グラフ彩色問題

(Wikipedia より)

スケジュール問題



制約充足問題の解き方

制約充足問題の解き方 (SAT ソルバを使う場合)

$\boxed{\text{制約充足問題}}$

$\Downarrow$ 制約記述言語処理系

(Sugar など)

$\boxed{\text{CNF 形式の充足可能性問題}}$

SAT Solver(minisat など)

$\Downarrow$ 今回のプロジェクトで作成

$\boxed{\text{解}}$

SAT ソルバとは

CNF(積和標準形) で書かれた充足可能性問題において、各節 (行) がすべて True となる値 x_i の組を求めるシステム。

積和標準形の例

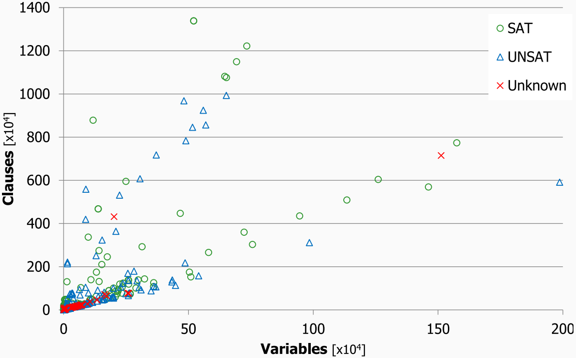
$$x_1, \ldots, x_n \in \{True, False\}$$

$$x_1 \vee x_2 \vee x_3$$

$$\neg x_2 \vee x_4$$

古典コンピュータにおける minisat などのソフトウェアが知られている。

SAT ソルバの現状



SAT 2011 競技会 Application 部門 300 問

http://www-erato.ist.hokudai.ac.jp/docs/seminar/nabeshima.pdf

本プロジェクトによる SAT ソルバの処理

CNF 形式入力

↓ QUBO の生成

QUBO

↓ アニーリング

アニーリングの解

↓

正しい解

古典コンピュータ上の SAT ソルバと異なる部分

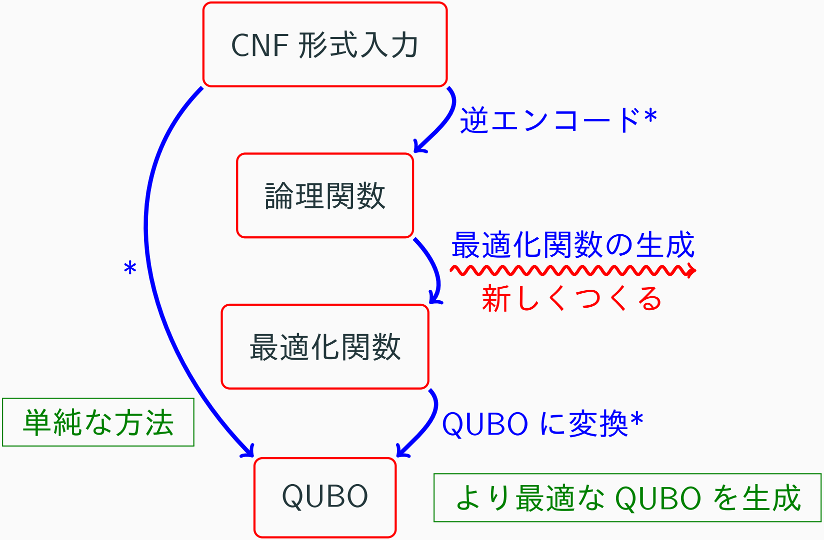
⇒ 解探索の時間を短縮

古典コンピュータによる解探索

（既存のアルゴリズムを使用）

⇑ 非常に時間のかかる処理

CNF から QUBO を生成



*:既に研究されている

最適化関数の生成方法

論理関数

引数, 戻り値 $\in \{True, False\}$ を、

バイナリ最適化関数

引数 $\in \{0, 1\}$, 戻り値: 実数 に変換する

最適化関数の生成方法

論理関数を実数値最適化関数に変換する。

論理変数 $x_i \in \{True, False\}$ を実数変数 $x_i^{(A)}, x_i^{(B)}, \ldots \in \mathbb{R}$ に変換する際に以下の戦略をとるとする。

戦略	$x =$*True*	$x =$*False*
A	$x^{(A)} = 0$	$x^{(A)} = 1$
$\overline{A}$	$x^{(\overline{A})} = 1$	$x^{(\overline{A})} = 0$
B	$x^{(B)} = 0$	$x^{(B)} \neq 0$
C	$x^{(C)} = 0$	$x^{(C)} > 0$

論理関数 $y = f(x_1, x_2)$ は y, x_1, x_2 の戦略に応じて複数通り (戦略が N 個なら最大 N^3 通り) の最適化関数で表される。

最適化関数の生成例

論理関数 $f(x_1, x_2) = x_1 \wedge x_2,\ g(x_1, x_2) = x_1 \vee x_2$ を変換する。最適化関数は

$$
\begin{aligned}
&f^{(\overline{A})} = x_1^{(\overline{A})} x_2^{(\overline{A})} = (1 - x_1^{(A)})(1 - x_2^{(A)}) \\
&f^{(A)} = 1 - x_1^{(\overline{A})} x_2^{(\overline{A})} = 1 - (1 - x_1^{(A)})(1 - x_2^{(A)}) \\
&f^{(\overline{C})} = x_1^{(\overline{C})} x_2^{(\overline{C})}, \quad f^{(\overline{B})} = x_1^{(\overline{B})} x_2^{(\overline{B})} \\
&g^{(\overline{C})} = x_1^{(\overline{A})} + x_2^{(\overline{A})} = x_1^{(\overline{C})} + x_2^{(\overline{C})}
\end{aligned}
$$

等が生成できる。

古典コンピュータによる解の探索

アニーリングによって求まる「仮の解」は、問題の制約を満たしているとは限らない。よって、後処理として古典コンピュータによる解の探索を行う。

DPLL アルゴリズム (例)

- 文字が一個のみの節がある場合は、その文字を True とみなす
- 全節の中に肯定と否定の両方が含まれない文字がある場合、それを True とみなす
- 適当な文字を選択し、True の場合と False の場合でそれぞれ探索する
- 以上を、解もしくは矛盾がみつかるまで繰り返す

ここまでのまとめ

本プロジェクトで開発する処理系では

- 複数の最適化関数から、最適な最適化関数を選ぶ
- アニーリングで前処理を行い、効率的に解の探索を行う

これらの工夫により、従来よりも効率的な SAT ソルバを目指す。また、従来の SpoonQ よりも大きな (多変数, 多制約式) 問題に対応できることが期待される。

本プロジェクトの位置づけ

プロジェクトの対象

対象	2019 年度	本プロジェクト
問題の規模	小さい	大きい
ユーザー	新たに組合せ最適化問題を解く初心者ユーザー	既存の手段を活用して制約充足問題を解いているユーザー
アニーリングの知識	不要	不要

$\Rightarrow$ より多くの人をアニーリングの世界に引き込むことができる

今後の予定

	2020							2021	
	6	7	8	9	10	11	12	1	2
初期実装・環境構築	–	–							
文献調査	–	–	–	–	–				
アルゴリズムの検討	–	–	–	–	–				
開発実施		–	–	–	–	–	–		
評価・ドキュメント化					–	–	–		

今後の予定

- 初期実装・環境構築
  ⇒ デジタルアニーラの動作確認など
- 文献調査
  ⇒ 関連実装, 関連研究 (今回利用する既存アルゴリズム等) の調査
- アルゴリズムの検討
  ⇒ 制約関数の評価方法等の検討
- 開発実施
  ⇒Rust 言語 (予定) を用いた提案アルゴリズム、SpoonQ との連携機能等の実装
- 評価及びドキュメント化
  ⇒ 提案アルゴリズム等のドキュメント化、既存の問題に対する性能評価

効果と展望

期待される効果

- SAT Solver としてのアニーリングマシンの活用推進
  既存の SAT ソルバ利用者をアニーリングに引き込む
- 成果を SpoonQ に取り込む →SpoonQ の性能向上

課題

- 最適化関数の評価方法
  どの最適化関数が最も効率的に求解できるか?