The 33rd Annual Conference of the Japanese Society for Artificial Intelligence, 2019

ハイブリッド制約処理系 HyLaGI における 分枝限定法を用いた離散変化時刻導出手法

A Branch-and-bound Algorithm for Determining Discrete Changes for Hybrid Constraint Solver HyLaGI

佐藤柾史　　上田和紀
Masashi SATO　　Kazunori UEDA

*1早稲田大学大学院基幹理工学研究科情報理工・情報通信専攻
Graduate School of Fundamental Science and Engineering, Waseda University

*2早稲田大学理工学術院情報理工学科
Faculty of Science and Engineering, Waseda University

Hybrid systems are dynamical systems involving continuous and discrete changes. Various models such as cyber-physical systems and control systems can be discribed as hybrid systems. We are developing HydLa, a modeling language of hybrid system and its symbolic simulator HyLaGI. HyLaGI performs exhaustive search to find time points of discrete changes, but its computation becomes a bottleneck for some programs having a large number of guarded constraints. In this research, we propose a efficient search method using a branch-and-bound algorithm and implement a prototype. The result of simple experiment shows that our approach reduces most of the search cost in the motivating example.

1. はじめに

ハイブリッドシステム [4] は時間進行に伴って連続変化と離散変化を繰り返す系であり, 物理系やサイバーフィジカルシステム等に広く適用できる概念である.

ハイブリッドシステムのシミュレーションとそれによる検証のための処理系として, HyLaGI[2] がある. HyLaGI は, HydLa[1] で記述されたプログラムを入力として, 不確定性をはらんだハイブリッドシステムの記号実行を実現する. HydLa では, ハイブリッドシステムを制約 (時相論理式や等式/不等式, 微分方程式) を用いてモデリングする. HyLaGI は記述された制約を HydLa の意味論に従って解くが, 記号を残した誤差の無いシミュレーションには様々な困難がある.

本研究は, HyLaGI における離散変化の要因が多数ある問題のシミュレーションの高速化を目的とする. 現在の HyLaGI では, 離散変化発生の原因をしらみつぶしに探索するため, その数に比例して実行時間が増大することが分かっている. 制約で記述された多数の離散変化条件を包含する緩和問題を作成し, 分枝限定法を用いた効率的な解探索をする手法の提案とプロトタイプ実装を行い, 目的とする例題での実行時間の抑制を実現した.

2. HyLaGI における離散変化時刻導出手続き

HydLa における離散変化は, 論理包含で記述された制約の前件 (ガード条件と呼ぶ) が成立するようになる, または成立しなくなることで発生する. これは, 時間の関数である変数の式とそれを参照 *1 するガード条件の元で時刻を最小化する連続最適化問題に帰着できる. HyLaGI では, 記述された各ガード条件についての最小化問題の作成と Mathematica の組み込み関数である Minimize を用いた求解を行うことで, 離散変化時刻を導出している. この手続きは軽いものではなく, ガード条件が多数ある場合はしらみつぶしに探索するため, 実行における明確なボトルネックとなることがある.

連絡先: 佐藤柾史 早稲田大学基幹理工学研究科情報理工・情報通信専攻, 〒 169-8555 新宿区大久保 3-4-1 63 号館 5 階 02 号, 03-5286-3340, masashi(at)ueda.info.waseda.ac.jp

*1 並行制約プログラミングでは ask と呼ばれる

```
1 // #define N 6
2
3 INIT_X(v) <=> (x = 0 & x'=v).
4 X_MOVE <=> [](x'' = 0).
5 INIT_Y(h) <=> (y = h & y' = 0).
6 FALL(g) <=> [](y'' = -g).
7 WALL(w) <=> [](x- = w & 0<=y-<=2*N => x' = -x'-).
8
9 BOUNCE_ON_STEP_HOR(cornerx) <=>
10   [](  (y- = N - cornerx-) & (cornerx- <= x- < cornerx- + 1)
11     => (y' = -9/10 * y'-) ).
12 BOUNCE_ON_STEP_VER(cornerx) <=>
13   [](  (x- = cornerx-) & (N - cornerx- < y- <= N - cornerx- + 1)
14     => (x' = -x'-) ).
15 BOUNCE_HOR := { BOUNCE_ON_STEP_HOR(i) | i in {0..N} }.
16 BOUNCE_VER := { BOUNCE_ON_STEP_VER(i) | i in {0..N} }.
17
18 INIT_X(1), INIT_Y(N + 3),
19 (FALL(9.8) << BOUNCE_HOR),
20 (X_MOVE << BOUNCE_VER << WALL(0)).
```

図 1: 階段を跳ねる質点の HydLa プログラム

2.1 最適化の対象とする例題

図 1 は, 階段を跳ねる質点をモデリングした HydLa プログラムである. HydLa プログラムの詳細は [1] を参照してほしい. 10, 13 行目が段の横, 縦を記述した制約である. 1 行目のマクロにより, 階段の段数を指定する.

図 2 は, このプログラムの階段の段数 N を変えながら HyLaGI で実行した際の階段の段数と実行時間の関係である. 質点が 10 回跳ねるまでにかかる時間を測定している. まず, 離散変化時刻の導出が全実行時間の殆どを占めていることが分かる. また, 離散変化時刻の導出にかかる時間は階段の段数に比例して増加することが分かる.

3. 緩和問題と分枝限定法

分枝限定法 [3] とは, ナップサック問題や巡回セールスマン問題といった離散最適化において, 効率的な枝刈りを行うための探索アルゴリズムである. 本章では, 分枝限定法の詳細を述

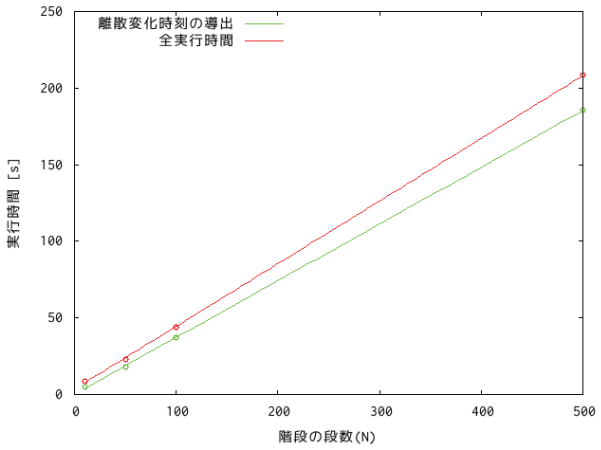

図 2: 階段を跳ねる質点のプログラムにおける段数と実行時間の関係

べる.

3.1 緩和問題

緩和問題とは, 最適化問題において目的関数を変えずに制約条件を緩めた問題のことである. 緩めた制約条件は, 元の問題の制約条件を包含する. 緩和問題について定理 1 が成立する. なお, 以降の記述において, 制約をそれを満たす集合で表す場合があることに注意してほしい.

定理 1. 制約条件 $\boldsymbol{x} \in X_p$ 下で $f(\boldsymbol{x})$ を最小化する問題 P と, 制約条件 $\boldsymbol{x} \in X_{rp}$ 下で $f(\boldsymbol{x})$ を最小化する問題 RP において, $X_p \subset X_{rp}$ が成立するならば, P の最適値 z_p* と RP の最適値 $z_{rp}*$ について, $z_p* \geq z_{rp}*$.

証明. $f(x_p*) = z_p*$ を満たす P の最適解 x_p* について, $x_p* \in X_p \subset X_{rp}$ より, x_p* は RP の実行可能解.
$\therefore z_p* \geq z_{rp}$ □

3.2 分枝限定法

分枝限定法では離散最適化問題 (最小化問題について説明) を, 部分問題への分割操作 (分枝) と部分問題の下界を計算して枝刈りを行う操作 (限定) を繰り返すことで最適値の探索を行う.

分枝操作では, $f(x)$ を $x \in X$ の元で最小化する問題において, X をいくつかの部分集合 $X_1, X_2, \ldots, X_n$ に分割して各々の元で $f(x)$ を最小化する問題に変換する. そして, 各部分問題の最適値を比較して, その中で最小の値, すなわち $min(f(x_1* \in X_1), f(x_2* \in X_2), \ldots, f(x_n* \in X_n))$ が元の問題の最適値となる.

分枝操作を再帰的に適用して元の問題の最適解を探索するのだが, 限定操作では, 各部分問題の最適値の下界 *2 を求める. 導出した下界となる値は, 以下のような探索の枝刈りに用いる.

1. 探索における暫定的な最適値 $z*$ が求まっている時, ある部分問題の最適値の下界 z_{min} が導出された. この時, $z* \leq z_{min}$ を満たすならば, その部分問題の最適値は暫定解を更新し得ないため探索を終了する.

2. ある部分問題 (制約は X) の最適値の下界 $f(x_{min}) = z_{min}$ が導出された. この時, $x_{min} \in X$ を満たすならば, その部分問題の解は x_{min} であるため, 解を保存して探索を終了する.

*2 必要ならば上界も求める

部分問題の最適値の下界は, 例えば部分問題を緩和問題に変換してそれを解くことで導出できる. 緩和問題の定理 1 より, 緩和問題の最適値は元の問題の下界となる. 探索の効率化という観点でいうと, 緩和問題の性質として, 元の問題と比べて十分に解きやすいことや下界が元の問題の解に近くなりやすい (あるいは一致しやすい) ことが要求される.

4. HyLaGI の離散変化時刻導出手続きにおける分枝限定法の適用

HydLa プログラムに出現する変数にかかる時間変化の等式制約 $S \Leftrightarrow x_1 = f_1(t) \wedge x_2 = f_2(t) \wedge \cdots \wedge x_n = f_n(t)$ とそれらを参照するガード条件の集合 $G = (g_1, g_2, \ldots, g_m)$ を用いて, HyLaGI の離散変化時刻の導出問題は以下の最小化問題として記述できる.

$$\text{Minimize } t \text{ such that } S \wedge (g_1 \vee g_2 \vee \cdots \vee g_m)$$

HyLaGI では論理和を分解して全てのガード条件についての最小化問題の求解と各々の最適値の比較を行っているが, この問題について分枝限定法を適用して離散変化時刻 (言い換えると, 原因となるガード条件) を効率よく探索することを考える.

4.1 分枝限定法を用いた離散変化時刻導出アルゴリズム

アルゴリズム 1 は, 分枝限定法を用いて離散変化時刻を導出する非決定アルゴリズムである. この関数は, ガード条件の集合と変数の時間変化の等式, 記号パラメータを受け取り, 記号パラメータに対応する離散変化時刻を返す. ここで, 記号パラメータは変数の持つ不確実性を保存しており, パラメータの取る値によって離散変化の原因が異なるかもしれないことを考慮する必要がある.

4.2 節で具体例を用いた説明を行うが, まずは一般的なアルゴリズムについて説明する. 15 行目の *CalculateRelaxedGuards* 関数は, ガード条件の集合を入力してそれらの論理和を論理包含する制約を返す. すなわち, アルゴリズム上の記号 $(g_1, g_2, \ldots, g_n) = G_{curt}$ と出力 $(h_1, h_2, \ldots, h_m) = H$ について, $\bigvee_{i=1}^{n} g_i \Rightarrow \bigvee_{j=1}^{m} h_j$ が成立する. この操作は分枝限定法の分枝, および限定操作のための緩和問題の作成に対応する. 多数のガード条件を幾つかのグループに振り分けて (分枝), 各々のグループに所属するガード条件を包含する新しい制約の作成を行う. 探索対象の多数のガード条件をひとまとめにすることが目的のため, n より m がずっと小さいことが望ましい. 16 行目のループにおいて, 緩和した制約下で t を最小化する問題 (部分問題) を解く. 17 行目の *FindMinTime* は作成した各々の緩和問題の求解を行う関数で, 部分問題の下界となる時刻を求める手続き (限定操作) に対応する. パラメータによる非決定性があるため, パラメタの取る値に対応した複数の解を返す場合がある. 19-20 行目の *CheckGuardSat* 関数は, 17 行目で導出した緩和問題の解が元の問題の実行可能解であるかの確認を行う. 具体的には, 部分問題に所属するガード条件を満たすかをチェックし, 満たされていたらその部分問題の最適解が求まることになる. この手続きも非決定性を伴うため, パラメタの取る値に対応してガードが満たされるか否かを返す. 7 行目, 10 行目は, 分枝限定法における枝刈りである. 7 行目は, 下界が暫定値を上回った部分問題の探索を打ち切る. 10 行目は, 19-20 行目で最適解が判明した場合に, パラメタに対応する最適値が見つかったことになり *3, 探索を打ち切る.

*3 最適値の下界でソートされた優先キューを用いた探索のため, 最初に求まった解がそのパラメタに対応する最も良い解である

アルゴリズム 1 分枝限定法を用いた離散変化時刻導出問題の非決定アルゴリズム

input:
G : ガード条件
S : 制約条件
P : 記号定数の条件
output:
最小離散変化時刻
記号定数の条件

```
1: function FindMinTimeUsingBranchAndBound(G, S, P)
2:     PQ := PriorityQueue()
3:     Sol := {}
4:     PQ.push(⟨0, G, P, false⟩)
5:     repeat
6:         ⟨T_curt, G_curt, P_curt, tf_curt⟩ := PQ.pop()
7:         if P_curt = false then
8:             continue
9:         end if
10:        if tf_curt = true then
11:            Sol := Sol ∪ {⟨T_curt, P_curt⟩}
12:            PQ := ⋃_{⟨T,g,p,tf⟩∈PQ} {⟨T, g, ¬P_curt ∧ p, tf⟩}
13:            continue
14:        end if
15:        H := CalculateRelaxedGuards(G_curt)
16:        for h ∈ H do
17:            MinResult := FindMinTime(Subst(h, S), P_curt)
18:            for ⟨T_min, p⟩ ∈ MinResult do
19:                CGSResult :=
20:                CheckGuardSat(Subst(S, t = T_min), G_curt, p)
21:                for ⟨tf, p⟩ ∈ CGSResult do
22:                    PQ.push(
23:                        ⟨T_min, {g | g ∈ G_curt ∧ g ⇒ h}, p, tf⟩)
24:                end for
25:            end for
26:        end for
27:    until PQ = {}
28:    return GetElement(Sol)
29: end function
```

4.2 アルゴリズムの具体化

アルゴリズム 1 は, あらゆるガード条件の入力に対応した一般的なアルゴリズムである. 一般的に分枝限定法が有効に働くかどうかは分枝と限定の方法に依り, このアルゴリズムでは 15 行目の *CalculateRelaxedGuards* の実装に強く依存する. ここでは, ガード条件をどのようにグルーピングし, どのように緩和するかについて一切言及しておらず, 実装にあたっては具体化が必要である. HydLa の制約として記述できる (不) 等式のクラスに制限はなく [1], あらゆるガード条件の入力に対して有効に働く *CalculateRelaxedGuards* 関数を実装することは現実的ではない. そこで, 本研究では 2 次元の幾何問題を記述した HydLa プログラムを動機とし, それに対して有効に働くアルゴリズムの具体化を試みる. 以下に実装の方針を列挙する. アルゴリズムの詳細は省略する.

- 入力されるガード条件は, ある変数 x, y のどちらかまたは両方を参照した, 一次方程式/一次不等式に制限する.
- 分枝操作では, 変数の時間変化の式 $x = f(t)$ における $x = f(0)$ を超平面として x-y 座標平面を分割し, それぞれに包含されるガード条件を同じグループとする. ガード条件が境界面をまたぐ場合は, どちらかに所属させるか独立させる (後述).
- 制約の緩和においては, 同じグループに所属するガード条件を凸包で包含することで実現する.

以上の方針に従って, 図 1 のプログラムにおける離散変化時刻の導出をアルゴリズム 1 で実行した際の探索の様子を図 3 に示す. なお, プログラムの質点の初期位置は階段の左端の壁だが, 図解を分かりやすくするために, 数フェーズ経過して質点が階段の上にある状況から次に跳ねる段を探索する場合を考える.

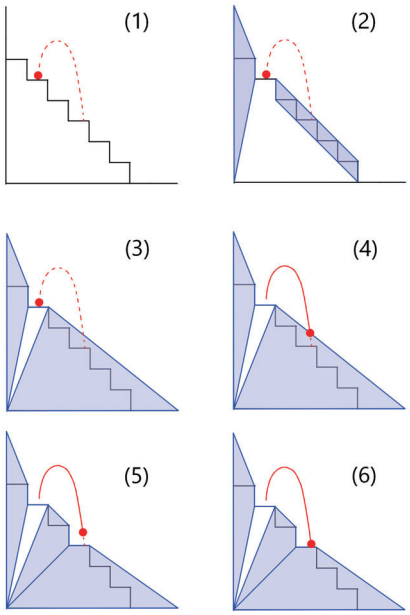

図 3: 階段を跳ねる質点のプログラムに提案アルゴリズムを適用した解探索の流れ

図 3 の (2)-(3) は, アルゴリズムの 15 行目の関数に対応する. まず, 質点の初期位置の x 座標の左右で空間を 2 分し, それらの領域に包含されるガード条件を同一のグループとして 2 分割する. 次に, それぞれのグループに属するガード条件を内包する凸包 (凸多角形) を作成する (2). ここで, 左右の空間にまたがるガード条件について考える. 図では, 床と質点が乗っている段を表す制約は, 左右の空間にまたがる. これらはどちらかの凸包にマージを試みるが, この際に凸包が質点の初期位置を包含しない場合に限りマージする. 凸包が包含する領域でガード条件を緩和するが, 質点の初期位置を包含してしまうと緩和問題の制約が最初から満たされることになり, 導出される時刻の下界が真の最適値と極端に離れてしまい役に立たなくなってしまうので, これを防ぐために行う. なお, マージの結果初期位置が凸包の境界になる場合も, 質点の移動する方向によっては同じ問題が発生するため, 失敗とする. また, マージに失敗した場合は独立した凸包にする. この場合, 床は右の凸包にマージできるが, 質点が乗る辺は独立させる. その結果, (3) のようになる. 左右の青い領域が示す凸包とマージに失敗した質点が乗るガード条件の, 3 つに対応する制約が関数の戻り値として返される.

	元の実装 [s]	提案手法 [s]
1 回目の離散変化時刻の探索 (N=100)	1.4072	0.15838
10 回目の離散変化時刻の探索 (N=100)	7.6225	0.808
1 回目の離散変化時刻の探索 (N=500)	6.5186	0.56365
10 回目の離散変化時刻の探索 (N=500)	38.597	1.2795

表 1: 提案手法を図 1 のプログラムに適用した実験結果

(4) は, アルゴリズムの 17-20 行目に対応する. それぞれの凸包が示す領域と変数の時間変化の式の元で時刻を最小化する問題を解く (17 行目). 右の凸包の元での最小化問題のみに最適値がある. 凸包は包含するガード条件を緩和した制約のため, この最適値は凸包が包含するガード条件の元で解いた最小化問題の最適値の下界となる. また, 導出された最適解が凸包が包含しているガード条件を満たすかを確認する (19-20 行目). (4) より, ガード条件を包含した際の近似誤差が原因で最適値が求まっているため, この最適値は元の問題の最適値と一致しない. そのため, 探索を続行する.

(5), (6) は, 今までの操作を右の凸包が包含するガード条件に対して再帰的に行っている. (6) では, 求めた最適解がガード条件に含まれることが図から分かり, 導出された下界が元の問題の最適値と一致する. 従って, 最適値を更新して探索を終了する.

5. 実装と例題を用いた実験

アルゴリズム 1 と, その 15 行目を具体化した節 4.2 のアルゴリズムを Mathematica でプロトタイプ実装をした. そして, 図 1 のプログラムにおいて離散変化時刻を特定するまでにかかる時間を元の HyLaGI の実装と比較した. 表 1 はその結果である.

表の各行はそれぞれ, 階段の段数が 100・500 段で 1・10 回目の離散変化時刻の導出にかかる時間を比較したものである. 階段の段数を比較したのは提案手法の実行にかかる時間と階段の段数の関係を確認するためである. また, ハイブリッドシステムの記号シミュレーションでは時間経過に伴って微分方程式の初期値が複雑化していく傾向があり, 跳ねるボールのように変数の時間変化の式が 2 次である場合もそれに当てはまる. 式の複雑化は離散変化時刻の導出にかかるコストにも強く影響するため, 離散変化の回数に対しても比較を行っている.

どの場合でも, 一桁の高速化が実現された. 階段の段数が増えたり, シミュレーションのフェーズが進行したりしたほうがコストが増加するのは同じであるが, その関係は元の実装ほど単純でない. そこで, アルゴリズム 1 の主要な関数に費やした累計時間の内訳を図 4 に示す. まず, 凸包作成の初期化処理 *4 では階段を表す制約のソートを行うため, 時間計算量は O(NlogN) である. また, 凸包は逐次添加法で作成するため, O(N) である. 図 4 のグラフは概ねそれに倣っている. 次にグラフから読み取れる重要な特性は, *FindMinTime* にかかる時間が階段の段数に依らない点である. 元の実装では, 最適化問題を作成して解く *FindMinTime* 関数が離散変化時刻導出における実行時間の内訳の全てであり, そのコストは段数に比例していたが, 分枝限定法が良く機能すれば *FindMinTime* の回数は階段の段数に対して定数オーダーで収めることができる. アルゴリズムが良く機能するか否かは, 不要な部分問題をいかに枝刈りできるかや, 最適解をいかに早く見つけて部分問題の探索を打ち切ることができるかに依る. ガード条件が一次でありそれらを凸包で包含する手法では, 凸包の辺とガード条件の境界が重なり, そこが最適解となった場合に探索を打ち切ることができる. 本実験では図 3 のように早々と最適解を求めることができるため, *FindMinTime* のコストは元の問題と比べて十分に小さく, そのために緩和問題の作成と求解のコストを加味しても高速化が実現できた.

*4 シミュレーションで一度行えば良い

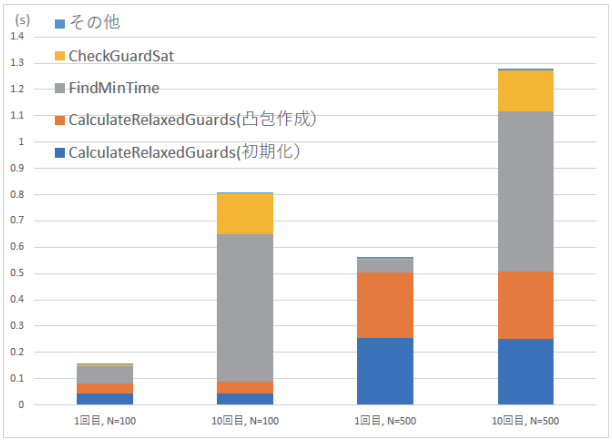

図 4: 実験結果の内訳

6. まとめと今後の課題

HyLaGI の離散変化時刻の導出手続きに分枝限定法を適用することで, 動機となる例題において一桁の高速化を実現した.

今後の課題としては, まず実験例題を増やして対象とする多くの例題で有効に機能するかどうかを確認する必要がある. モデルが変われば探索の様子も大きく変わるため, 一つの例題で有効性を主張することはできない. 加えて, 4.2 節の分枝の方針は単純であり, 部分問題の大きさが偏ってしまう場合がある. 探索木が偏り, 探索が葉まで到達する最悪のケースでは, ガード条件を全探索することになるだけでなく余計な緩和問題を解くことになるため, 元の実装より悪化することは明白である. より広いモデルに適用するためには, 部分問題の大きさをバランスさせる工夫が必要である.

参考文献

[1] 上田和紀, 石井大輔, 細部博史 : 制約概念に基づくハイブリッドシステムモデリング言語 HydLa, SSV2008(第 5 回システム検証の科学技術シンポジウム), 2008.

[2] Matsumoto, Shota et al. HyLaGI: Symbolic Implementation of a Hybrid Constraint Language HydLa. Electr. Notes Theor. Comput. Sci. 317 (2015): 109-115.

[3] Land, Ailsa H., and Alison G. Doig. An automatic method of solving discrete programming problems. Econometrica: Journal of the Econometric Society (1960): 497-520.

[4] Lunze, J : Handbook of Hybrid Systems Control : Theory, Tools, Applications, Cambridge University Press, 2009.