情報工学コース卒業研究報告

解集合プログラミングを用いた時間割問題の解法に関する考察

2020年2月

桑原 和也

概要

時間割問題は, 求解困難な組合せ最適化問題の一種である．カリキュラムベース・コース時間割 (Curriculum-based Course Timetabling; CB-CTT) 問題は, 大学等の 1 週間の講義スケジュールを求める問題であり, 最も研究が盛んな教育時間割問題の一つである．近年, 解集合プログラミング (Answer Set Programming; ASP) を用いた CB-CTT 問題の解法が提案され, 成功を収めている．ASP は系統的探索であることを活かして, 未解決問題の最適値を決定するなど優れた性能を示している．しかし, その一方で, 制約が多く含まれるような問題集において，局所的探索を用いた解法より性能が劣っている場合が見られる．

この問題を解決するために，系統的探索と局所的探索を組み合わせた Large Neighborhood Prioritized Search (LNPS) が提案されている．LNPS は，暫定解に含まれる変数の値割り当ての一部をランダムに選んで取り消し，他の値割り当てをなるべく維持したままで解を再構築する反復法の一種である．LNPS の性能は，暫定解の一部をランダムに選んで取り消す destroy 演算子に依存するが，十分な研究がなされてない．

本論文では，LNPS を用いた CB-CTT 問題の解法について述べる．CB-CTT に対する既存研究を応用して，3 種類の destroy 演算子 (random, day-period, day-room) を ASP 上に実装した．random が問題の性質をまったく利用しないのに対し，day-period と day-room は CB-CTT のソフト制約を考慮して暫定解の一部をランダムに選んで取り消す点が特長である．提案手法の有効性を評価するために，国際時間割競技会 ITC-2007 のベンチマーク問題 (21 問) を用いて実行実験を行った．その結果，多くの問題に対して，day-period と day-room が既存 ASP 解法より良い解を生成し，提案手法の有効性が確認できた．

目 次

図 目 次

表 目 次

コード目次

第1章　緒論

時間割問題 (Timetabling Problem) は，求解困難な組合せ最適化問題の一種である．この問題には，ハード制約とソフト制約が存在し，ソフト制約に違反するとペナルティが与えられる．必ず満たすべきハード制約を満たしながら，ペナルティの総和を最小にするような解を求めることが目的である．現状では，質の高い時間割を編成するために多くの人間の労力が費やされている．このような背景から，時間割に関する国際会議 (Practice and Theory of Automated Timetabling; PATAT) や国際時間割競技会 (International Timetabling Competition; ITC) が開催され，時間割ソルバーの性能向上に貢献している．

解集合プログラミング (Answer Set Programming; ASP) [?, ?, ?, ?] は，論理プログラミングから派生した宣言的プログラミングパラダイムの一つである．ASP 言語は一階論理に基づく知識表現言語の一種であり，論理プログラムは ASP のルールの有限集合である．ASP システムは論理プログラムから安定モデル意味論に基づく解集合を計算するシステムである．近年，SAT 技術を応用した高速 ASP システムが実現され，ロボット工学，システム生物学，システム検証，プランニングなど様々な分野への実用的応用が急速に拡大している．

近年，**カリキュラムベース・コース時間割** (Curriculum-based Course Timetabling; CB-CTT) 問題に対する ASP を用いた解法が提案され，成功を収めている [?]．CB-CTT 問題は，大学等の 1 週間の講義スケジュールを求める問題であり，最も研究が盛んな教育時間割問題の一つである．ASP は系統的探索であることを活かして，未解決問題の最適値を決定するなど優れた性能を示している．しかし，その一方で，ソフト制約が多く含まれるような問題集において，局所的探索を用いた解法より性能が劣っている場合が見られる．

この問題を解決するために，系統的探索と局所的探索を組み合わせた **Large Neighborhood Prioritized Search** (LNPS) [?] が提案されている．LNPS は，暫定解に含まれる変数の値割り当ての一部をランダム

に選んで取り消し，他の値割り当てをなるべく維持したままで解を再構築する反復法の一種である．ASP を用いた LNPS の利点は，解の再構築を系統的探索で行え，値割り当てをなるべく維持したままでの再構築が自然に実現できることである．LNPS の性能は，暫定解の一部をランダムに選んで取り消す destroy 演算子に依存するが，十分な研究がなされてない．

本論文では，LNPS を用いたカリキュラムベース・コース時間割 (CB-CTT) 問題の解法について述べる．CB-CTT に対する既存研究 [?] を応用して，3 種類の destroy 演算子 (random, day-period, day-room) を実装した．random が問題の性質をまったく利用しないのに対し，day-period と day-room は CB-CTT のソフト制約を考慮して暫定解の一部をランダムに選んで取り消す点が特長である．提案手法の有効性を評価するために，国際時間割競技会 ITC-2007 のベンチマーク問題 (21 問) を用いて実行実験を行った．その結果，多くの問題に対して，day-period と day-room が既存 ASP 解法より良い解を生成し，提案手法の有効性が確認できた．

以降の構成は以下の通りである．第二章で時間割問題，特に本論文が対象とするカリキュラムベース・コース時間割問題について述べる．第三章で解集合プログラミングについて述べ，その中で，LNPS の実装において重要な変数選択ヒューリスティクスの変更機能について述べる．第四章で LNPS 及び 実装した destroy 演算について述べる．第五章でカリキュラムベース・コース時間割問題に実装解法を適用した実験の結果を述べ，既存 ASP との比較や，destroy 演算同士での比較を交え考察を行う．最後に第六章で結論を述べる．

第2章　時間割問題

時間割問題は求解困難な組合せ最適化問題の一種である．社会の様々な場面に応じた種々の時間割が存在し, 現状では, 質の高い時間割を編成するために多くの人間の労力が費やされている．このような背景から, 時間割に関する国際会議 PATAT が 1995 年から開催されている．主な研究対象として, 教育時間割 (educational timetabling), 輸送時間割 (transport timetabling), 従業員時間割 (employee timetabling), スポーツ時間割 (sports timetabling) などがある．

その中で教育時間割は, 与えられた制約を満たしながら, 講義や試験などをそれぞれの日時と教室に割り当てることによって編成され, 教育機関にとって重要な問題である．さらに, 教育時間割はコース時間割 (course timetabling), 試験時間割 (examination timetabling), 高校時間割 (school timetabling) に大きく分けられる．また近年では, 国際時間割競技会 ITC も開催され, 時間割ソルバーの性能向上に貢献している．

2.1　カリキュラムベース・コース時間割問題

本研究が対象とする時間割問題は, カリキュラムベース・コース時間割である．この問題は最も研究が盛んな時間割問題の一つであり, ITC2007 競技会のトラック３で使用された問題である．ITC2007 競技会終了後, 時間割問題のポータルサイトが整備され, 問題インスタンス, 最適値・最良値の一覧などが提供されている．最適値・最良値は, メタヒューリスティクスに基づく各種アルゴリズム, 整数計画法, SAT・MaxSAT などの様々な手法で求められている．以下では, カリキュラムベース・コース時間割問題を, 単に時間割問題と呼ぶ．

まず, この時間割問題に関する用語を説明する．課程 (curriculum) は共通の受講者をもつ複数の科目から構成される．科目 (course) は担当教員, 講義回数, 受講者数などが決められており, 通常の授業科目に対応する．

```
Name: Toy                          CURRICULA:
Courses: 4                         Cur1 3 SceCosC ArcTec TecCos
Rooms: 3                           Cur2 2 TecCos Geotec
Days: 5
Periods_per_day: 4                 UNAVAILABILITY_CONSTRAINTS:
Curricula: 2                       TecCos 2 0
Min_Max_Daily_Lectures: 2 3        TecCos 2 1
UnavailabilityConstraints: 8       TecCos 3 2
RoomConstraints: 3                 TecCos 3 3
                                   ArcTec 4 0
COURSES:                           ArcTec 4 1
SceCosC Ocra 3 3 30 1              ArcTec 4 2
ArcTec Indaco 3 2 42 0             ArcTec 4 3
TecCos Rosa 5 4 40 1
Geotec Scarlatti 5 4 18 1          ROOM_CONSTRAINTS:
                                   SceCosC rA
ROOMS:                             Geotec rB
rA 32 1                            TecCos rC
rB 50 0
rC 40 0                            END.
```

コード 2.1: 時間割問題の入力例 (ectt 形式)

```
SceCosC rB 1 2          TecCos rB 0 2          Geotec rA 0 1
SceCosC rB 3 3          TecCos rB 0 3          Geotec rA 1 0
SceCosC rB 4 2          TecCos rB 1 1          Geotec rA 2 2
ArcTec rB 2 1           TecCos rB 2 3          Geotec rA 4 0
ArcTec rB 2 2           TecCos rB 4 1
ArcTec rB 3 2           Geotec rA 0 0
```

コード 2.2: 時間割問題の出力例

各科目は複数回の講義から成り, 各講義には曜日 (day) , 時限 (period) および教室 (room) が割り当てられる. 日時は曜日と時限の組で表される.

時間割問題の入力は, 科目と教室と課程の集合, 曜日と時限の数, 1 日あたりの講義数の上下限, 開講不可能な科目と日時 (および教室) の組合せの集合である. 出力は, 各科目の全ての講義に対する日時と教室の割り当てである. また, この問題にはハード制約とソフト制約が存在し, ソフト制約に違反するとペナルティが与えられる. 必ず満たすべきハード制約を満たしながら, ペナルティの総和を最小にするような解を求めることが目的である.

時間割問題の入力は ectt と呼ばれるテキスト形式で表される. コード **??**に入力例を示す. 入力は問題名等を表すヘッダ部分と科目等を表す 5 つのブロック部分からなる. `Name` ヘッダは問題名を表す. `Courses` ヘッダは科目数を表す. `Rooms` ヘッダは教室数を表す. `Days` ヘッダは曜日数を示す. `Periods_per_day` ヘッダは 1 日あたりの時限数を表す. `Curricula`

ヘッダは課程数を表す. Min_Max_Daily_Lectures ヘッダは各課程における, 1 日あたりの講義数の上下限を表す. UnavailabilityConstraints ヘッダは開講不可能な科目と日時の組合せの数を表す. RoomConstraints ヘッダは開講不可能な科目と教室の組合せの数を表す.

COURSES ブロックは科目の集合からなる. 各行が 1 つの科目を表し, 科目名, 教員名, 講義回数, その科目が開講される曜日数の最小値, 受講者数, 連続講義フラグが順に示されている. 連続講義とは, 同一曜日, 同一教室において連続した時限で開講される講義のことである. コード ??の例では, 科目 SceCosC は教員 Ocra が担当し, 講義回数は週 3 回で, 3 日以上開講し, 受講者数が 30 名, 同一曜日に 2 回以上開講される場合は連続講義の形態をとる.

ROOMS ブロックは教室の集合からなる. 各行が 1 つの教室を表し, 教室名, 収容可能人数, 建物名が順に示されている. コード ??の例では, 教室 rA は建物 1 にあり, 収容可能人数は 32 名である.

CURRICULA ブロックは課程の集合からなる. 各行が 1 つの課程を表し, 課程名, その課程に属する科目数, その課程に属する全ての科目名が順に示されている. コード ??の例では, 課程 Cur1 は SceCosC, ArcTec, TecCos の 3 つの科目から構成される.

UNAVAILABILITY_CONSTRAINTS ブロックは, 開講不可能な科目と日時の組合せの集合からなる. 各行が 1 つの組合わせを表し, 科目名, 曜日, 時限が順に示されている. コード ??の例では, 科目 TecCos は, 水曜日 (曜日 2) の 1 時限目 (時限 0) と 2 時限目 (時限 1), 木曜日 (曜日 3) の 3 時限目 (時限 2) と 4 時限目 (時限 3) には開講できない.

ROOM_CONSTRAINTS ブロックは, 開講不可能な科目と教室の組合せの集合からなる. 各行が 1 つの組み合わせを表し, 科目名, 教室名が順に示されている. コード ??の例では, 科目 SceCosC は教室 rA では開講できない.

時間割問題の出力は 1 週間の講義スケジュールであり, 科目名とそれが開講される教室, 曜日, 時限が表される. コード ??に, コード ??の入力例に対する出力例を示す. この出力では, 科目 SceCosC は火曜日 (曜日 1) の 3 時限目 (時限 2), 木曜日 (曜日 3) の 4 時限目 (時限 3), 金曜日 (曜日 4) の 3 時限目 (時限 2) にすべて教室 rB で開講されるということがわかる.

次に制約について説明を行う. 時間割問題のハード制約は以下の 4 つである.

(H_1) **Lectures:**
各科目のすべての講義は, 異なる日時に開講される. 各科目の講義回数は COURSES ブロックで指定される.

(H_2) **Conflicts:**
同一教員が担当する科目のすべての講義は, 異なる日時で開講される. また, 同一課程に属する科目のすべての講義は, 異なる日時で開講される. 各科目の担当教員は COURSES ブロックで指定され, 各課程に属する科目は CURRICULA ブロックで指定される.

(H_3) **RoomOccupancy:**
同一日時に同一教室で異なる講義を開講できない.

(H_4) **Availability:**
各科目の講義は, 開講不可能な日時に開講されることはない. 各科目の開講不可能な日時は UNAVAILABILITY_CONSTRAINTS ブロックで指定される.

時間割問題のソフト制約は以下の 9 つである.

(S_1) **RoomCapacity:**
各科目について, 受講者数が使用する教室の収容可能人数を超えてはいけない. 違反した場合, 超過人数に応じたペナルティが課される. 各科目の受講者数は COURSES ブロックで指定され, 各教室の収容可能人数は ROOMS ブロックで指定される.

(S_2) **MinWorkingDays:**
各科目について開講される日数が, 指定された開講される曜日数の最小値を下回ってはいけない. 違反した場合, 下回った日数に応じたペナルティが課される. 各科目が開講される曜日数の最小値は COURSES ブロックで指定される.

(S_3) **IsolatedLectures:**
同一課程に属する講義は, 連続した時限に開講される. 同一曜日に同一課程に属する他のどの講義とも隣接していない (孤立した) 講義がある場合に違反となり, 孤立した講義毎にペナルティが課される.

(S_4) **Windows:**
同一課程に属する講義は, 空き時限なしで開講される. 同一曜日に

同一課程に属する２つの講義の間に空き時限 (同一課程に属する講義のない時限) がある場合に違反となり, 空き時限の長さに応じたペナルティが課される.

(S_5) **RoomStability:**
同一科目のすべての講義は, 同一教室で開講される. 違反した場合, 異なる教室数 (最初の教室は除く) に応じたペナルティが課される.

(S_6) **StudentMinMaxLoad:**
各課程について, １日あたりの講義数は決められた範囲に収まらなければならない. 違反した場合, 範囲の上限を上回った (あるいは, 下限を下回った) 講義数に応じたペナルティが課される. 各課程の１日あたりの講義数の上下限は `Min_Max_Daily_Lectures` ヘッダで指定される.

(S_7) **TravelDistance:**
学生は講義と講義の間に建物を移動する時間が必要である. 各課程について, 同一曜日に異なる建物の教室で開講される連続した２つの講義がある場合, すなわち, 瞬間移動が必要な場合に違反となり, 瞬間移動毎にペナルティが課される.

(S_8) **RoomSuitability:**
各科目の講義は, 開講不可能な教室で開講されることはない. 違反した講義毎にペナルティが課される. 開講不可能な教室は `ROOM_CONSTRAINTS` ブロックで指定される.

(S_9) **DoubleLectures:**
いくつかの科目は連続講義の形態をとる. 連続講義の形態をとる科目は, 同一曜日に複数の講義がある場合, それらは連続した時限に同一教室で開講される. 違反した講義毎にペナルティが課される. 連続講義の形態をとる科目は `COURSES` ブロックで指定される.

時間割問題のシナリオとは, 特定のソフト制約の集合と, 各ソフト制約のペナルティに対する重みを加えたものである. 表 **??**に, これまでに提案されている５つのシナリオを示す. “制約” の行は各シナリオの名称を表す. 各シナリオの列はそれぞれ, 整数値がシナリオに含まれるソフト制約に対する重みを, “H” はシナリオに含まれるハード制約を, “-” はシナリオに含まれないことを表している. UD1 は最も基本的なシナリオであ

表 2.1: 時間割問題の制約とシナリオ

制約	UD1	UD2	UD3	UD4	UD5
H_1. Lectures	H	H	H	H	H
H_2. Conflicts	H	H	H	H	H
H_3. RoomOccupancy	H	H	H	H	H
H_4. Availability	H	H	H	H	H
S_1. RoomCapacity	1	1	1	1	1
S_2. MinWorkingDays	5	5	-	1	5
S_3. IsolatedLectures	1	2	-	-	1
S_4. Windows	-	-	4	1	2
S_5. RoomStability	-	1	-	-	-
S_6. StudentMinMaxLoad	-	-	2	1	2
S_7. TravelDistance	-	-	-	-	2
S_8. RoomSuitability	-	-	3	H	-
S_9. DoubleLectures	-	-	-	1	-

り, すべてのハード制約と 3 つのソフト制約 (S1, S2, S3) からなる. UD2 は ITC2007 競技会で使用されたシナリオである. UD3, UD4, UD5 は比較的新しいシナリオであり, UD3 は各課程の 1 日当たりの講義数, UD4 は連続講義, UD5 は移動時間に着目したシナリオとなっている.

第3章　解集合プログラミング

解集合プログラミング (ASP; [?, ?, ?, ?]) の言語は, 一般拡張選言プログラムをベースとしている．本章では, 簡単のため, そのサブクラスである標準論理プログラムについて説明する. 以下では, 標準論理プログラムを単に論理プログラムと呼ぶ．論理プログラムは, 以下の形式のルールの有限集合である.

$$a_0 \leftarrow a_1, \ldots, a_m, {\sim}a_{m+1}, \ldots, {\sim}a_n$$

ここで，$0 \leq m \leq n$ であり, このルールの直観的な意味は, 「$a_1, \ldots, a_m$ がすべて成り立ち，$a_{m+1}, \ldots, a_n$ のそれぞれが成り立たないならば，a_0 が成り立つ」である．各 a_i はアトム, $\sim$ はデフォルトの否定 (述語論理における否定 ($\neg$) とは意味が異なる), "," は連言 ($\wedge$) を表す．また, $\leftarrow$ の左側をヘッド, 右側をボディと呼ぶ. ボディが空のルールをファクトと呼び, $\leftarrow$ を省略して表すことが出来る. ヘッドが空のルールを一貫性制約と呼ぶ.

ASP 言語には, 組合せ問題を解くために便利な拡張構文が用意されている．その一例として, 選択子や個数制約がある．選択子は $\{a_1; \ldots; a_n\}$ のように表され, アトム集合 $\{a_1, \ldots, a_n\}$ の任意の部分集合を表現することが出来る. 選択子の両端に選択可能な個数の上下限を付けることで, 任意の部分集合を表していた選択子ではなく, アトムの個数が上下限内に収まるような部分集合を表す個数制約となる. また, 組合せ最適化問題を解くための最小化関数 (`#minimize`) や最大化関数 (`#maximize`) も存在する. 最小化, および最大化したい目的関数は複数記述することが可能であり, それぞれに優先度を付けることで優先度の高い目的関数から最適化探索を行うようにすることも可能である.

ASP システムは, 与えられた論理プログラムから, 安定モデル意味論 [?] に基づく解集合を計算するシステムである. 近年では, *clingo* [1]，*DLV* [2]，*WASP* [3] など, SAT ソルバー技術を応用した高速な ASP システムが開発

[1] `https://potassco.org/`
[2] `http://www.dlvsystem.com/dlv/`
[3] `https://www.mat.unical.it/ricca/wasp/`

されている．なかでも *clingo* は, 高性能かつ高機能な ASP システムとして世界中で広く使われている.

次に, 解集合の定義について簡単に説明する．詳細については文献 [?, ?, ?, ?] を参考にされたい. 論理プログラム P について考える．そして以下に示すような表記を導入する.

- $head(r)$ はルール r のヘッドを表す
- $body(r)^+$ はルール r のボディにあるデフォルトの否定が付いていないアトムの集合を表す
- $body(r)^-$ はデフォルトの否定が付いているアトムの集合を表す

論理プログラム P のアトム集合 X に関するリダクト P^X を以下のように定義する.

$$P^X = \{head(r) \leftarrow body(r)^+ \mid r \in P, body(r)^- \cap X = \emptyset\}$$

この時, アトム集合 X が P^X の最小モデルであるならば, X が P の解集合となる.

解集合プログラミングを用いた問題解法のプロセスは, 以下の手順からなる．まず最初に, 解きたい問題を論理プログラムとして表現する．次に, ASP システムを用いて, 論理プログラムの解集合を計算する．最後に, 解集合を解釈して元の問題の解を得る．ここでは, グラフ彩色問題を例として, 解法プロセスの説明を行う．ASP システムとしては *clingo* を用いる．以降で示す論理プログラムのソースコードはすべて *gringo* 言語で書かれており, 論理プログラムの説明で用いた記号のソースコード上での表記法を表 ??に示す.

論理プログラム	$\leftarrow$	,	;	$\sim$
ソースコード	`:-`	`,`	`;`	`not`

表 3.1: 論理プログラムとソースコードにおける記号の対応

グラフ彩色問題とは, 辺で結ばれたノードが同じ色にならないように, 各ノードを塗り分ける問題である．図 ??のグラフを赤 (`r`), 青 (`b`), 緑 (`g`) の 3 色で塗り分ける問題を例として用いる．この問題を表す論理プログラムをコード ??に示す.

1〜3行目は, ノード (`node`) と辺 (`edge`) をファクトとして書くことによって, 図 ??のグラフを表している．ピリオド (`.`) はルールの終わりを表す.

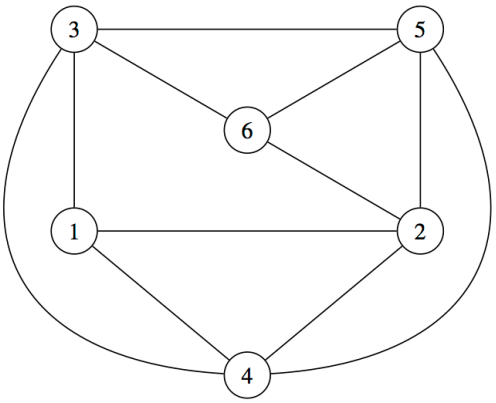

図 3.1: グラフ彩色問題のグラフ

```
1  node(1).    node(2).    node(3).   node(4).    node(5).    node(6).
2  edge(1,2).  edge(1,3).  edge(1,4). edge(2,4).  edge(2,5).  edge(2,6).
3  edge(3,4).  edge(3,5).  edge(3,6). edge(4,5).  edge(5,6).
4
5  col(r).   col(b).   col(g).
6
7  1 { color(X,C) : col(C) } 1 :- node(X).
8  :- edge(X,Y), color(X,C), color(Y,C).
9
10 #show color/2.
```

コード 3.1: グラフ彩色問題の論理プログラム (color.lp)

５行目も同様にして, ファクトによって色 (col) が表されている．７行目のルールは, 個数制約を使って, 各ノードが一つの色で塗られるという制約を表している．アトム color(X,C) は, ノード X が色 C で塗られることを意味する．セミコロン (:) は条件付きリテラルと呼ばれる拡張構文で, このルールのヘッドは, 1 { color(X,r);color(X,b);color(X,g) } 1 のように展開される．つまり, ある node(X) が存在する時, color(X,r),color(X,b), color(X,g) のいずれか一つが真 (いずれか一つの色で X が塗られる) ということになる．８行目のルールは, 一貫性制約を使って, 辺で結ばれたノード (X と Y) は, 同じ色 (C) で塗られないという制約を表している．

ASP システムは解集合を計算して出力する．コード **??**に *clingo* の実行例を示す．この出力から, ノード１と５は緑, ノード４と６は赤, ノード２と３は青に塗り分けられることがわかる．

clingo には, 求解における変数選択ヒューリスティクスの変更を, プログラム上から行うことが出来る機能が存在する．変更には, 論理プログラ

```
clingo version 5.4.0
Reading from color.lp
Solving...
Answer: 1
color(2,b) color(1,g) color(3,b) color(4,r) color(5,g) color(6,r)
SATISFIABLE

Models       : 1+
Calls        : 1
Time         : 0.001s (Solving: 0.00s 1st Model: 0.00s Unsat: 0.00s)
CPU Time     : 0.001s
```

コード 3.2: color.lpに対する *clingo* の実行例

ム上で以下のような表記を用いることで行う.

$$\#heuristic \quad A \ : \ Body. \quad [w, m]$$

これは, *Body* が成り立つ時, アトム A の変数ヒューリスティクスを重み w と指定子 m に従って変更することを表している.

変数選択ヒューリスティクスをどのように変更するかは, 指定子によって決定される. 本論文では, 指定子として $true$ と $false$ を例にとり説明をする. $true$ はアトムに優先的に真を割り当てるようにする指定子であり, $false$ はアトムに優先的に偽を割り当てるようにする指定子である. 例えば, "$\#heuristic\ A.$ [1,$true$]" は A に優先度レベル1で真を優先して割り当てることを表し, "$\#heuristic\ A : B.$ [2,$false$]" は B が真である場合に, A に優先度レベル2で偽を優先して割り当てることを表す. 各アトムのデフォルトのレベルは0であり, 最もレベルの高いアトムから真もしくは偽が割り当てられる.

コード**??**は, *#heuristic* ルールを用いた論理プログラムの例である. 1行目には選択子と呼ばれる拡張構文が使用されており, アトム *a*,*b* は真でも偽でも良いということを表している. 2行目は一貫性制約を用いて, アトム *a*,*b* が同時に真になってはいけないということを表している. 3, 4行目はそれぞれ *a*,*b* に関する *#heuristic* ルールであり, 3行目は, 優先度1で *a* に真を割り当てることを, 4行目は, 優先度2で *b* に真を割り当てることを表している.

コード**??**に *clingo* の実行例を示す. *clingo* では "–heu=domain" をオプションとして指定することで, *#heuristic* ルールが有効となる. この実行では, アトム *b* のみが真となった解が出力されている. これは, *a* に関しても真を優先的に割り当てるようなルールが記述されているが, *a* に対するルールが優先度1であるのに対し, *b* に対するルールは優先度2であるの

```
1 {a; b}.
2 :-a,b.
3 #heuristic a. [1, true]
4 #heuristic b. [2, true]
```

コード 3.3: 変数選択ヒューリスティクスの例 (heu.lp)

```
1  $ clingo heu.lp --heu=domain
2  clingo version 5.4.0
3  Reading from heu.lp
4  Solving...
5  Answer: 1
6  b
7  SATISFIABLE
8
9  Models       : 1+
10 Calls        : 1
11 Time         : 0.001s (Solving: 0.00s 1st Model: 0.00s Unsat: 0.00s)
12 CPU Time     : 0.001s
```

コード 3.4: heu.lp に対する *clingo* の実行例

で, まず b に優先して真が割り当てられる. その後に, 一貫性制約によって a,b が同時に真になることは禁止されているために a に偽が割り当てられ, その結果が出力されている. 仮に a に関するルールの優先度が 3 となっていれば, 実行結果は a のみを含む解が出力される. また, コード**??**で 2 行目の一貫性制約の記述が無ければ, a,b どちらの*#heuristic* ルールも有効となって, 実行結果は a,b の両方を含む解が出力されることになる.

第4章　カリキュラムベース・コース時間割問題に対するLNPSの適用

近年, カリキュラムベース・コース時間割問題に対する ASP を用いた解法が提案され, 成功を収めている [?]. ASP は系統的探索であることを活かして, 未解決問題の最適値を決定するなど優れた性能を示している. しかし, その一方で, ソフト制約が多く含まれるような問題集において, 局所的探索を用いた解法より性能が劣っている場合が見られる. この問題を解決するために, 系統的探索と局所的探索法を組み合わせた LNPS (Large Neighborhood Prioritized Search) [?] が提案されている.

4.1　LNPS (Large Neighborhood Prioritized Serrch)

プランニングやスケジューリングといった問題に対して, 系統的探索と局所的探索を組み合わせた LNS (Large Neighborhood Search) が有効であることが知られている. LNS をベースに系統的探索と局所的探索を組み合わせた手法で ASP に適したものとして LNPS (Large Neighborhood Prioritized Search) が提案され, 応用事例の蓄積が行われている. LNS では解に含まれる変数の値割り当ての一部をランダムに選んで取り消し, その変数に対してのみ再割り当てを行い解を再構築するが, LNPS では解の再構築を, 値割り当てをなるべく維持したままでの再探索に置き換えることで, 取り消されていなかった変数への再割り当てを許している. ASP を用いた LNPS の利点は, 解の再構築を系統的探索で行え, 値割り当てをなるべく維持したままでの再構築が自然に実現できることである.

[?] から引用した LNPS のアルゴリズムを, アルゴリズム 4.1 に示す.

アルゴリズム 4.1 Large neighborhood prioritized search [?]

```
1: input: a feasible solution x
2: x^b = x;
3: repeat
4:       x^t = re-search(d(x));
5:       if accept(x^t,x) then
6:            x = x^t;
7:       end if
8:       if c(x^t) < c(x^b) then
9:            x^b = x^t;
10:      end if
11: until stop criterion is met
12: return x^b
```

1～3 行目では, 初期解を x と置き, 最良解 $x^b = x$ としてループに入る. 4 行目では, 以下の destroy と re-search で x から得られた解を x^t とする. destroy は x から一定の割合でランダムに値割り当てを選択し, x' とする. re-search は x' の値割り当てをなるべく維持したまま再探索する. 5～7 行目では, x^t を受理する条件を満たしていたら $x = x^t$ とする. 受理条件では, 例として「x^t が x より改善された解なら」などの条件を用いる. 8～10 行目では, x^t が最良解 x^b より改善された解なら, $x^b = x^t$ としている. 11 行目では, 終了条件が満たされていればループを抜け出す. 満たされていなければ 4 行目に戻り, ループに入る. 終了条件には, 制限時間や繰り返し回数などを用いる. 12 行目で最良解 x^b を返して終了する.

4.2　destroy 演算の開発

LNPS の性能は, 暫定解の一部をランダムに選んで取り消す destroy 演算子に依存するが, 十分な研究がなされてない. そこで CB-CTT に対する既存研究 [?] を応用して, 3 種類の destroy 演算子 (random, day-period, day-room) を実装した.

1. **random (Random n% destruction)**
   n を任意の数として, 暫定解の値割り当ての中から, ランダムに n% を選んで取り消す. 今回の実験では n として 0 , 10, 20 を採用した. n として 0 を選んだ場合, 解を取り消さずに暫定解の全ての値割り当てに対して優先度を上げる. そのため, 暫定解の値割り当てに近い値割り当ての解への探索を促進させることを狙いとしている. 値割り当てを 10, 20% 取り消す場合は, ある程度暫定解の近辺を探索しながら, 局所的最適解に陥らないようにさせることを狙いとしている.

2. **day-period (Random day-period destruction)**
   ランダムに曜日と時限をそれぞれ一つ選択し, 選択した曜日の選択した時限に割り当てられている値割り当てを全て取り消す. この時, 取り消す値割り当てが存在しなければ, 再び曜日と時限をランダムに選択し, 同様の操作を何らかの値割り当てを取り消すまで繰り返す. この destroy 演算は, 割り当てられる教室の変更を促進させ, 教室に関するソフト制約へのペナルティを改善することを狙いとしている.

3. **day-room (Random day-room destruction)**
   ランダムに曜日と教室をそれぞれ一つ選択し, 選択した曜日の選択した教室に割り当てられている値割り当てを全て取り消す. この時, 取り消す値割り当てが存在しなければ, 再び曜日と教室をランダムに選択し, 同様の操作を何らかの値割り当てを取り消すまで繰り返す. この destroy 演算は, 割り当てられる時限の変更を促進させ, 時限に関する制約へのペナルティを改善することを狙いとしている.

LNPS の実装は, 3 章の最後に説明した, *#heuristic* ルールを用いて行った. 各種 destroy 演算を行い, 値割り当ての一部が取り消された暫定解の内, 残りの全ての割り当てに対して優先的に真を割り当てるようにして実装した.

第5章　実行実験

5.1　実験概要

実装した解法の性能を評価するため, 以下の比較実験を行った.

- 比較対象：
  1. 既存 ASP 解法：系統的探索のみ
  2. 提案解法 R-n ：LNPS + random (n は 0, 10, 20 の 3 種類)
  3. 提案解法 R-dp：LNPS + day-period
  4. 提案解法 R-dr：LNPS + day-room
- ベンチマーク：カリキュラムベース・コース時間割ベンチマーク問題 (全 21 問, ソフト制約の多い UD5 シナリオ)
- ASP 符号化：*teaspoon* 符号化 [**?**]
- 制限時間：１時間
- 既存 ASP 解法の設定
  - 二通りのオプションを使用
  - 使用コマンド：
    ・bb 「clingo –opt-strat=bb,0 –restart-on-model –opt-heu=3」
    ・usc「clingo –opt-strat=usc,11 –config=jumpy」
- 提案解法の設定
  - 初期解：既存 ASP 解法の設定で, 制限時間を 30 分としたもの
  - destroy と re-search：
    各 destroy 演算をかけ, re-search を 60 秒間行う．これを１サイクルとし, 合計 30 サイクル繰り返す.
  - 使用コマンド：
    既存 ASP 解法のものに各々 –heu=domain を追加した二通り
  - re-search で得た解の受理条件：常に受理

既存 ASP 解法の二通りのオプションは, 先行研究によって結果の良かった 2 種類のものを使用した. bb に続くコマンドを使用すると分枝限定法を用いた探索を行い, usc に続くコマンドを使用すると充足不能コアを用いた探索を行う. 以下では, それぞれ分枝限定法を bb, 充足不能コアを usc と表記する. 既存 ASP 解法において, 制限時間が 30 分の解と 60 分の解を比較すると, 解の改善がみられない, あるいはほとんど改善されなかった問題がしばしば見られた. そのため, 提案解法の初期解には 30 分時点での解を使用し, そこに LNPS を適用することでどれだけ解が改善されるかを実験し確認した. また destroy を行った後の re-search に関して, bb を用いた探索での予備実験を行ったところ, どの destroy 演算においても re-search 開始から 30 秒程度で目的関数の値の改善が滞っていることが確認できたため, 1 度の re-search にかける時間を 60 秒とした.

実験環境として, CPU は 6 コア Intel Core i7 3.2GHz, メモリーは 64GB の Mac mini (OS は macOS Catalina バージョン 10.15.2) を使用した. ASP システムには, *clingo-5.4.0* を使用した.

5.2　実験結果

表 **??**および表 **??**に比較結果を示す.

表 **??**では, “Instance” の列は問題名を表している. “既存 ASP 解法”, “R-0”, “R-10”, “R-20”, “R-dp”, “R-dr”, の列はそれぞれ順に, 既存 ASP 解法, 提案解法 R-0, 提案解法 R-10, 提案解法 R-20, 提案解法 R-dp, 提案解法 R-dr で求めたペナルティの合計値を表している. また各提案解法で re-search を行う際, bb と usc の二通りの探索法で実験を行ったが, usc による探索での結果が bb のものと比較した際にほぼ全ての結果でペナルティの合計値が大きくなっていた. よって, 今回は re-search に bb を用いたもののみ扱う. 既存 ASP 解法の列は bb と usc でペナルティの小さかった方を表の結果とした. 値の右肩には, bb を用いた結果の場合 b, usc を用いた結果の場合 u, bb と usc で同じ値だった場合 b/u と付けてある. 提案解法の列では, 使用した初期解が bb と usc による二通りのものがあり, ペナルティの小さかった方を表の結果とした. こちらも同様にして右肩に文字が付けられている. 各問題について 6 種類の手法の中の最良値を太字で表示している. 表の下 2 行に関してはそれぞれ, 各手法において最適値決定が出来た問題の数, 最適値決定は出来なかったが 6 種類の手法の中での最良値が求められた問題の数を表している.

最適値の数に関しては, 提案解法では全て３問であり, 既存 ASP 解法では４問と既存 ASP 解法の方が１問多かった. 最良値の数では, R-dp, R-dr がそれぞれ６問, ７問と他の解法より良い結果で, 次いで R-10 が３問であった. それらの合計の数でも, R-dp, R-dr がそれぞれ９問, 10 問と他の解法より良い結果を示した.

表 **??**では, “Instance” の列は問題名を表している. “既存の最良値” の列は先行研究によるペナルティの合計値の既知の上界を表す. “既存 ASP 解法”, “R-0”, “R-10”, “R-20”, “R-dp”, “R-dr”, の列はそれぞれ, 「各手法で求められた値」を「既存の最良値の値」で割った数の小数点第２位までを表す. 最下行は, それぞれの手法で, 前述した計算で算出した比率の 21 問分の平均を表す.

比率に関しては, 既存 ASP 解法の平均が 5.39 なのに対して, R-0, R-10, R-dp, R-dr はより良い結果を示しており, R-20 は 7.49 と既存 ASP 解法よりも悪い結果を示した. 既存 ASP 解法より良かった提案解法で比較すると, R-dp が 2.56 と最も良く, 次いで R-dr が 2.61 であった. 次に R-0 が 2.72 と良く, 少し離れて R-10 が 4.63 であった.

5.3 分析と考察

表 **??**の比較では, 各手法について最適値決定が出来た問題の数が３, ４問となっていた. ただし, comp04, comp08, comp11 に関しては, usc を用いた既存 ASP 解法で 30 分実行した段階で最適値決定が既に出来ていた. そのため, 該当のベンチマーク問題３問を除くと最適値決定が行えたのは, 既存 ASP 解法による comp14 の１問のみであった. しかし, R-20 と R-dr による comp11 の結果では, 30 分時点で最適値決定が出来ていなかった, bb による初期解を利用した実行でも最適値決定が行えた.

今回, re-search での探索は usc を用いた方で良い結果が得られなかったため, bb を用いた方のみ表の結果として採用した. これは usc の探索の性質上一度の目的関数の改善に時間がかかる場合があり, １回の re-search の制限時間に設定した 60 秒が短かった可能性がある. 実行結果を解析すると, re-search を usc を用いた探索で行ったものでは一度の re-search で目的関数の値に変化がないまま次の re-search に移っているパターンが bb を用いたものよりも多くみられた. １回の re-search の時間を長くすることで usc による探索の結果が改善できる可能性がある.

また re-search は bb を用いた探索のみに限定したが, 初期解の探索法

は bb と usc の二通りから結果が良かったものを表に採用した．ベンチマーク問題の種類や destroy の手法によってどちらが良かったかが様々に変わっており, 現状では初期解を二通り用意してパラレルに実行することで良い解を得る想定をしている．ベンチマーク問題をさらに増やすなどして規則性を見つけることができれば, パラレルな実行を行う場合よりリソースを削減できる可能性がある．

既存の最良値に対する比率での比較の際, R-0, R-10, R-20 で比較すると, 解を取り消す割合が大きくなるにつれ結果が悪くなっている．R-10 や R-20 は局所的最適解に陥らないようにすることを目的としていたが, この結果から, 10% を超えるような割合で解の取り消しを行うと取り消しを全く行わないものより結果が悪くなると考えられる．

R-dp, R-dr は最良値の数の比較と, 既存の最良値に対する比率での比較の両方で他の手法より良い結果が得られた．そこで, 表に結果として採用した実行において, 両手法における destroy 演算で解をどれだけの割合取り消していたかを解析した．各問題について 30 回 destroy 演算が施されており, 各 destroy 演算でそれぞれ解の取り消しの割合は異なる．そこで, 各問題ごとに 30 回分の取り消し割合の平均を計算し, その値からさらに 21 問分の平均を計算した．その結果, R-dp は 3.77%, R-dr は 1.55% (小数第二位まで計算) であった．そこで, 実行実験において提案解法 R-n の n として 0 ,10, 20, を設定し実験を行ったが, 新たに n として 3.77, 1.55 を設定し追加実験を行った．既存の最良値に対する比率の平均を算出した結果, R-3.77 では 2.59, R-1.55 では 2.57 という結果が得られた．この結果から, R-dp, R-dr と近い良い結果が得られており, R-dp, R-dr とは効き方にどのような差があるのか調べることが今後の課題の一つとなる．

また以上の結果から, 解の取り消し割合を 10%, 20% と大きくしていくと結果が悪くなったが, 0% よりも良い結果が 1.55% や 3.77% で得られた．0% から 10% の間にさらに良い結果が得られるような取り消しの割合が存在する可能性があり, さらなる実験によって調べる必要がある．

表 5.1: 実験結果: 求められた最良値

Instance	既存 ASP 解法	R-0	R-10	R-20	R-dp	R-dr
comp01	132^{b}	13^{b}	$\mathbf{11^{b}}$	$\mathbf{11^{b}}$	13^{b}	$\mathbf{11^{b}}$
comp02	$\mathbf{331^{u}}$	455^{b}	413^{b}	883^{u}	413^{b}	387^{b}
comp03	302^{u}	200^{u}	211^{u}	1009^{u}	$\mathbf{187^{u}}$	196^{u}
comp04	$\mathbf{49^{u}}$	$\mathbf{49^{u}}$	$\mathbf{49^{u}}$	$\mathbf{49^{u}}$	$\mathbf{49^{u}}$	$\mathbf{49^{u}}$
comp05	1940^{u}	1135^{u}	1312^{u}	2071^{u}	$\mathbf{1012^{u}}$	1109^{u}
comp06	822^{b}	365^{b}	923^{u}	1039^{u}	394^{b}	$\mathbf{310^{b}}$
comp07	924^{b}	513^{u}	1075^{b}	1263^{b}	$\mathbf{428^{b}}$	530^{u}
comp08	$\mathbf{55^{u}}$	$\mathbf{55^{u}}$	$\mathbf{55^{u}}$	$\mathbf{55^{u}}$	$\mathbf{55^{u}}$	$\mathbf{55^{u}}$
comp09	254^{u}	192^{u}	$\mathbf{176^{u}}$	703^{u}	181^{u}	232^{b}
comp10	822^{b}	377^{b}	837^{b}	1111^{b}	376^{b}	$\mathbf{367^{b}}$
comp11	$\mathbf{0^{u}}$	$\mathbf{0^{u}}$	$\mathbf{0^{u}}$	$\mathbf{0^{b/u}}$	$\mathbf{0^{u}}$	$\mathbf{0^{b/u}}$
comp12	1246^{u}	788^{u}	1895^{b}	2353^{b}	794^{u}	$\mathbf{778^{u}}$
comp13	301^{u}	187^{u}	241^{u}	800^{b}	$\mathbf{181^{u}}$	182^{u}
comp14	$\mathbf{67^{u}}$	72^{u}	71^{u}	826^{u}	71^{u}	72^{u}
comp15	607^{u}	316^{u}	469^{b}	848^{b}	$\mathbf{238^{u}}$	259^{u}
comp16	944^{b}	416^{b}	912^{b}	996^{b}	$\mathbf{356^{b}}$	402^{b}
comp17	412^{u}	232^{u}	328^{u}	1018^{b}	230^{u}	$\mathbf{227^{u}}$
comp18	471^{u}	201^{b}	194^{u}	599^{b}	222^{u}	$\mathbf{181^{u}}$
comp19	890^{b}	331^{b}	330^{u}	768^{b}	304^{b}	$\mathbf{248^{b}}$
comp20	1386^{u}	$\mathbf{755^{u}}$	1374^{u}	1546^{b}	787^{u}	768^{u}
comp21	310^{u}	187^{u}	$\mathbf{168^{u}}$	911^{b}	178^{u}	170^{u}
最適値の数	4	3	3	3	3	3
最良値の数	1	1	3	1	6	7

表 5.2: 比較結果: 既存の最良値を 1 とした場合の比率

Instance	既存 ASP 解法	R-0	R-10	R-20	R-dp	R-dr	既存の最良値
comp01	12.00	1.18	1.00	1.00	1.18	1.00	11
comp02	2.54	3.50	3.17	6.79	3.17	2.97	130
comp03	2.12	1.40	1.48	7.10	1.31	1.38	142
comp04	1.00	1.00	1.00	1.00	1.00	1.00	49
comp05	3.40	1.99	2.30	3.63	1.77	1.94	570
comp06	9.67	4.29	10.85	12.22	4.63	3.64	85
comp07	22.00	12.21	25.59	30.07	10.19	12.61	42
comp08	1.00	1.00	1.00	1.00	1.00	1.00	55
comp09	1.69	1.28	1.17	4.68	1.20	1.54	150
comp10	11.41	5.23	11.62	15.43	5.22	5.09	72
comp11	1.00	1.00	1.00	1.00	1.00	1.00	0
comp12	2.57	1.63	3.92	4.87	1.64	1.61	483
comp13	2.04	1.27	1.63	5.44	1.23	1.23	147
comp14	1.00	1.07	1.05	12.32	1.05	1.07	67
comp15	3.44	1.79	2.66	4.81	1.35	1.47	176
comp16	9.83	4.33	9.50	10.37	3.70	4.18	96
comp17	2.65	1.49	2.11	6.56	1.48	1.46	155
comp18	3.43	1.46	1.41	4.37	1.62	1.32	137
comp19	7.12	2.64	2.64	6.14	2.43	1.98	125
comp20	11.17	6.08	11.08	12.46	6.34	6.19	124
comp21	2.05	1.23	1.11	6.03	1.17	1.12	151
平均	5.39	2.72	4.63	7.49	2.56	2.61	

第6章　結論

本論文では, LNPS の性能に重要な役割を果たす destory 演算子について, 性能評価を行うことを目的として, 3 種類の destory 演算子 (random, day-period, day-room) を実装した.

提案手法の性能を評価するために, 国際時間割協議会 ITC-2007 のベンチマーク問題 (21 問) を用いて実行実験を行った. その結果, UD5 シナリオの多くの問題に対して, day-period と day-room が既存 ASP 解法より良い解を生成し, 提案手法の有効性が確認できた.

今後の課題として, さらなる destroy 演算の考案や評価, 狙いを持って実装した destroy 演算子が意図通りに働いているかの検証, re-search の手法等の destroy 演算以外での問題へのアプローチ, また, 他の組み合わせ最適化問題への適用などが挙げられる.

謝辞

本研究の機会を賜り, 熱心にご指導いただきました名古屋大学大学院情報学研究科番原 睦則教授に心から感謝の意を表します. また, 本研究全般にわたって常日頃より様々なご意見をいただき, 実りある研究生活にしてくださった番原研究室の皆様に感謝いたします. そして, 大学生活で関わった全ての方に感謝いたします.