修　士　論　文

解集合プログラミングに基づく系統的探索と確率的局所探索の統合的手法に関する研究

252005066　桑原 和也

名古屋大学大学院情報学研究科

情報システム学専攻

2021年度

概要

解集合プログラミング (Answer Set Programming; ASP [2, 6, 9]) 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る．ASP ソルバーは安定モデル意味論 [6] に基づく解集合を計算するプログラムである．近年，SAT ソルバーの実装技術を応用した高速 ASP ソルバーが実現され，人工知能分野の諸問題を中心に実用的応用が急速に拡大している [5].

解集合プログラミングが成功した応用事例の一つに，**カリキュラムベース・コース時間割** (Curriculum-Based Course TimeTabling; CB-CTT [3]) がある．CB-CTT は大学等での一週間の講義スケジュールを編成する求解困難な組合せ最適化問題である．CB-CTT は必ず満たすべきハード制約と，できるだけ満たしたい重み付きソフト制約から構成される．違反するソフト制約の重み (ペナルティ) の総和の最小化が目的となる．解集合プログラミングは，この問題に対し，ASP ソルバーの系統的探索を活かして，未解決問題の最適値決定を含む優れた性能を示している [1].

しかし，その一方で，ソフト制約の種類が多い問題集においては，確率的局所探索に基づくメタ戦略が，多くの問題に対してより高精度な解を求めている．以上から，系統的探索の長所である “最適性の保証” と確率的局所探索の長所である “計算時間相応の解精度” の両方を備えた統合的探索手法を実現することは重要な研究課題といえる．

本論文では，解集合プログラミング技術を用いた**優先度付き巨大近傍探索** (Large Neighborhood Prioritized Search; LNPS) の実装，および，開発したソルバー *asprior* の性能評価について述べる．

先行研究で提案した LNPS [8] は，ASP の系統的探索とメタ戦略の一種である巨大近傍探索 (Large Neighborhood Search; LNS [10]) を統合した探索手法である．LNS は解に含まれる変数の値割当ての一部をランダムに選んで取り消し，その変数のみに対して再割当てを行うことで解を再構築する反復解法である．これに対して，LNPS は，解の再構築の操作を，値割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，取り消されなかった変数への再割当てを許す．これによって，どの値割当てを取り消すかに依存しすぎない探索を行うことができる点が特長である．

開発した *asprior* は，ASP ファクト形式の問題インスタンスと問題を解く ASP 符号化を入力とし，LNPS アルゴリズムを用いて解を求める汎用的なソルバーである (図 5.1 参

照)．*asprior* は，高速 ASP ソルバー *clingo* の Python インターフェースを利用して実装されている．

提案手法を評価するために，CB-CTT 問題集 (全 61 問) を用いて性能評価を行った．その結果，競技会使用問題については表 6.2 に示すように，既知の最良値との比について，通常の ASP 解法が +437% であったのに対し，提案手法は，その比を +16% まで大幅に改善できた．さらに，6 問について，既知の最良値を更新することに成功した．今後の課題としては，アダプティブ LNPS への拡張や他の時間割問題への適用などが挙げられる．

目 次

図 目 次

表 目 次

コード目次

第1章　はじめに

解集合プログラミング (Answer Set Programming; ASP [2, 6, 9]) 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る．ASP ソルバーは安定モデル意味論 [6] に基づく解集合を計算するプログラムである．近年，SAT ソルバーの実装技術を応用した高速 ASP ソルバーが実現され，人工知能分野の諸問題を中心に実用的応用が急速に拡大している [5].

解集合プログラミングが成功した応用事例の一つに，**カリキュラムベース・コース時間割** (Curriculum-Based Course TimeTabling; CB-CTT [3]) がある．CB-CTT は大学等での一週間の講義スケジュールを編成する求解困難な組合せ最適化問題である．CB-CTT は必ず満たすべきハード制約と，できるだけ満たしたい重み付きソフト制約から構成される．違反するソフト制約の重み (ペナルティ) の総和の最小化が目的となる．解集合プログラミングは，この問題に対し，ASP ソルバーの系統的探索を活かして，未解決問題の最適値決定を含む優れた性能を示している [1].

しかし，その一方で，ソフト制約の種類が多い問題集においては，確率的局所探索に基づくメタ戦略が，多くの問題に対してより高精度な解を求めている．以上から，系統的探索の長所である “最適性の保証” と確率的局所探索の長所である “計算時間相応の解精度” の両方を備えた統合的探索手法を実現することは重要な研究課題といえる．

本論文では，解集合プログラミング技術を用いた**優先度付き巨大近傍探索** (Large Neighborhood Prioritized Search; LNPS) の実装，および，開発したソルバー *asprior* の性能評価について述べる．

先行研究で提案した LNPS [8] は，ASP の系統的探索とメタ戦略の一種である巨大近傍探索 (Large Neighborhood Search; LNS [10]) を統合した探索手法である．LNS は解に含まれる変数の値割当ての一部をランダムに選んで取り消し，その変数のみに対して再割当てを行うことで解を再構築する反復解法である．これに対して，LNPS は，解の再構築の操作を，値割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，取り消されなかった変数への再割当てを許す．これによって，どの値割当てを取り消すかに依存しすぎない探索を行うことができる点が特長である．

開発した *asprior* は，ASP ファクト形式の問題インスタンスと問題を解く ASP 符号化を入力とし，LNPS アルゴリズムを用いて解を求める汎用的なソルバーである (図 5.1 参

照)．*asprior* は，高速 ASP ソルバー *clingo* の Python インターフェースを利用して実装されている．

提案手法を評価するために，国際時間割競技会の CB-CTT 問題集 (全 21 問) を用いて性能評価を行った．その結果，表 6.2 に示すように，既知の最良値との比について，通常の ASP 解法が +437% であったのに対し，提案手法は，その比を +16% まで大幅に改善できた．さらに，comp07, comp09, comp13, comp18 について，既知の最良値を更新することに成功した．今後の課題としては，アダプティブ LNPS への拡張や他の時間割問題への適用などが挙げられる．

第2章　解集合プログラミング

2.1　ASP 言語

解集合プログラミング (ASP; [2, 6, 9]) 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 [6] 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.

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

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 [2, 6, 9] 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.

2.2　プログラム例

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 2.1 に示す.

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

表 2.1: 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 2.1 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 2.1 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 2.1 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. 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

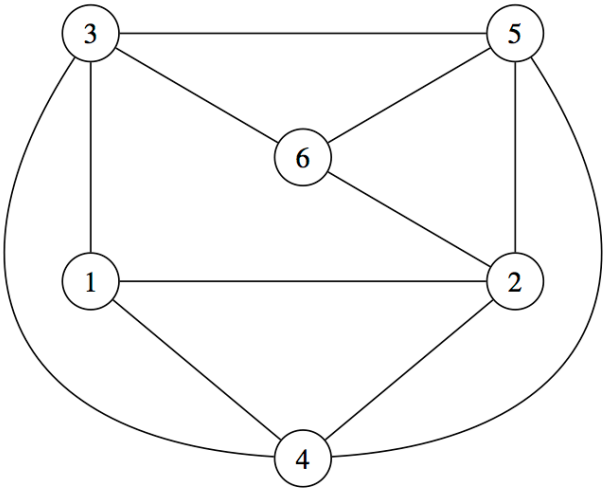

図 2.1: グラフ彩色問題のグラフ

```
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 2.1: グラフ彩色問題の論理プログラム (color.lp)

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 2.2 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, 論理プログラム上で以下のような表記を用いる

```
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 2.2: color.lpに対する *clingo* の実行例

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 5.1 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 5.2 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 5.1 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.

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

コード 2.3: 変数選択ヒューリスティクスの例 (heu.lp)

```
$ clingo heu.lp --heu=domain
clingo version 5.4.0
Reading from heu.lp
Solving...
Answer: 1
b
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 2.4: heu.lp に対する *clingo* の実行例

第3章　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. 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 3.1 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

```
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 3.1: 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 3.2: 時間割問題の出力例

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 3.1 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 3.1 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 3.1 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 3.1 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 3.1 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 3.2 に, コード 3.1 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.

表 3.1: 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	-

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 3.1 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4章　優先度付き巨大近傍探索

4.1　LNS

巨大近傍探索 (LNS) は，解に含まれる変数の値割当ての一部をランダムに選んで取り消し，その変数のみに対して再割当てを行うことで解を再構築 (*repair*) する反復解法である．

図 4.1 に LNS のアルゴリズムを示す．

Algorythm Large Neighborhood Search

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

図 4.1: LNS アルゴリズム

1. 初期解を x と置き，暫定解 $x^* := x$ とする (1–2 行目)．
2. 以下の *destroy* と *repair* で x から得られた解を x^t と置く (4 行目)．
   - *destroy* は x から値割当ての一部を取り消し x' とする．
   - *re-search* は x' の値割当てを変化させずに解を再構築する．
3. 受理条件 *accept* を満たしていたら $x := x^t$ とする (5–7 行目)．

- 例えば,「x^t が x より改善された解なら」という条件を用いる.

4. x^t が暫定解 x^* より改善された解なら，$x^* := x^t$ とする (8–10 行目).
5. 終了条件が満たされるまで，ステップ 2〜4 を繰り返す (11 行目).
   - 例えば，反復回数や制限時間などを終了条件に用いる.
6. 暫定解 x^* を返す (12 行目).

解の再構築には貪欲法等が用いられ，一般に解の最適性を保証できない．LNS の応用としては，巡回セールスマン問題を一般化した運搬経路問題 (Vehicle Routing Problem) に対する有効性が示されている [10].

4.2 LNPS

優先度付き巨大近傍探索 (LNPS) は，メタ戦略の一種である巨大近傍探索 (LNS [10]) を基に，組合せ最適化問題に対して，系統的探索と確率的局所探索を統合的に適用するよう拡張した手法である．図 4.2 に LNPS のアルゴリズムを示す.

Algorythm Large Neighborhood Prioritized Search

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

図 4.2: LNPS アルゴリズム

1. 初期解を x と置き，暫定解 $x^* := x$ とする (1–2 行目).
2. 以下の *destroy* と *re-search* で x から得られた解を x^t と置く (4 行目).
   - *destroy* は x から値割当ての一部を取り消し x' とする.

- *re-search* は x' の値割当てをなるべく維持したまま再探索する．

3. 受理条件 *accept* を満たしていたら $x := x^t$ とする (5–7 行目)．
   - 例えば，「x^t が x より改善された解なら」という条件を用いる．
4. x^t が暫定解 x^* より改善された解なら，$x^* := x^t$ とする (8–10 行目)．
5. 終了条件が満たされるまで，ステップ 2〜4 を繰り返す (11 行目)．
   - 例えば，反復回数や制限時間などを終了条件に用いる．
6. 暫定解 x^* を返す (12 行目)．

提案手法 LNPS では，LNS の再構築 (*repair*) を，値割当てをなるべく維持したままでの再探索 (*re-search*) に置き換えることで，取り消されなかった変数への再割当てを許す．これによって，どの値割当てを取り消すかに依存しすぎない探索を行うことができる点が特長である．また，再探索 (*re-search*) の終了条件を適切に設定することで解の最適性も保証できる．

第5章　ASP ソルバー 上での LNPS の実装

提案手法 LNPS を 高速 ASP ソルバー *clingo* [1] 上に実装した．提案ソルバー *asprior* は，ASP ファクト形式の問題インスタンスと問題を解く ASP 符号化を入力とし，LNPS アルゴリズムを用いて解を求める (図 5.1 参照)．

提案ソルバーは，*clingo* の Python インターフェースを利用して実装されている．以下に，LNPS を実装する上で重要な役割を果たす ASP 技術について簡単にまとめる．

探索ヒューリスティックス

ASP ソルバー *clingo* では，`#heuristic` 文を用いて，探索ヒューリスティックスをカスタマイズすることができる[2]．以下に例を示す．

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

コード 5.1: `#heuristic`文の例 (`heu.lp`)

1 行目のルールは，選択子を使ってアトム `a` と `b` を導入している．2 行目のルールは，`a` と `b` が同時に成り立たないことを表している．このプログラム例の解集合は`{}`,`{a}`,`{b}`の 3 つである．3 行目の `#heuristic` 文は，優先度 1 で `a` に真を割当てることを表している．同様に，4 行目は優先度 2 で `b` に真を割当てることを表している．アトムに対するデフォルトの優先度は 0 である．コード 5.2 に `#heuristic` 文を無効にした実行例，コード 5.3 に有効にした実行例を示す．これらの例より，`#heuristic` 文を使うことによって，解の探索において優先度の高いアトムから順に値が割当てられていることがわかる．

---

[1] `https://potassco.org/clingo/`

[2] *clingo* のデフォルト探索ヒューリスティックスは，SAT ソルバーと同じ VSIDS (Variable State Independent Decaying Sum)

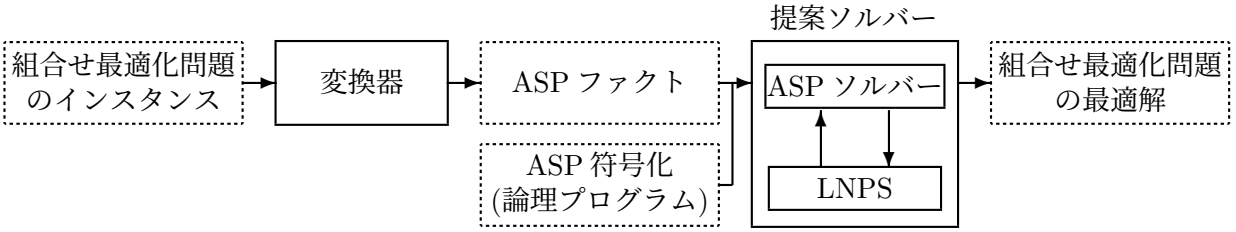

図 5.1: 提案ソルバー *asprior* の構成

```
$ clingo heu.lp --heu=domain
clingo version 5.4.0
Reading from heu.lp
Solving...
Answer: 1
b
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 5.2: `#heuristic`文を無効にした実行例

LNPS の実装では，*destroy* で値割当てを取り消されなかった変数に対し，`#heuristic` 文を利用して優先的に同じ値を割当てる．これにより，値割当てをなるべく維持したままでの解の再探索 (*re-search*) を *clingo* の系統的探索を用いて自然に実装できる．

マルチショット ASP 解法

ASP ソルバー *clingo* は，同様の探索失敗を防ぐために獲得した学習節を保持することで，無駄な探索を行うことなく，ルールを追加・削除した論理プログラムを連続的に解くことができる．このマルチショット ASP 解法は，別名インクリメンタル ASP 解法とも呼ばれ，モデル検査やプランニング分野の諸問題に応用されている．

LNPS の実装では，*clingo* の Python インターフェースを介してこの解法を利用することで，*destroy* と *re-search* の反復実行を簡潔に実装できる．

コード 5.4 は LNPS プログラムのメイン関数部分（一部コメント等を除く）である．順にコードの内容を説明する．

```
% clingo heu.lp -n 0 --heu=domain
clingo version 5.4.0
Reading from heu.lp
Solving...
Answer: 1
b
Answer: 2
a
Answer: 3

SATISFIABLE
```

コード 5.3: #heuristic文を有効にした実行例

- reuse プログラムの基礎化を行う (2 行目).
  - 入力された論理プログラムを命題論理レベルのプログラムに変換する手順を基礎化と呼ぶ．また入力された論理プログラムはそれぞれ名前を付けて部分プログラムに分けることが可能であり，reuse と名前が付けられた部分には，LNPS アルゴリズムにおける *destroy* と *re-search* のオプションの情報が記述されている．
- *destroy* と *re-search* のオプションを取得する (3 行目).
- base プログラムの基礎化を行う (5 行目).
  - 入力された論理プログラムのうち，名前を付けていない部分は全て base プログラムとして扱われる．
- インスタンスを生成する (6 行目).
- 初期解を求め，暫定解とする (7–8 行目).
- リストや変数などの初期化を行う (10–13 行目).
- #heuristic 文を文字列として生成する (14–16 行目).
- 生成した #heuristic 文を動的にプログラムに追加する (17 行目).
  - heuristic はプログラムに付ける名前，t は LNPS アルゴリズムにおける反復の回数を表している．

```
def main(prg):
    prg.ground([('reuse', [])])
    get_lnps_config(prg)

    prg.ground([('base', [])])
    solver = Solver(prg)
    sol, cost = solver.run(init_option)
    sol_best, cost_best = sol, cost

    heu_atoms = []
    prev_heu_atoms = []
    t = 0
    h = ''
    for c in lnps_config:
        a = c['atom_name'].name + '(' + ','.join(['X'+str(i) for i in range(c['
    args_num'].number)]) + ')'
        h += '#heuristic {0} : heuristic({0},W,M,t). [W,M]'.format(a)
    prg.add('heuristic', ['t'], h)

    while not solver.finished:
        t += 1
        for i in range(len(lnps_config)):
            targets = destroy(sol, lnps_config[i])
            atoms = prioritize(targets, lnps_config[i], t)
            heu_atoms.extend(atoms)
        for a in prev_heu_atoms:
            prg.release_external(a)

        s = ''
        for a in heu_atoms:
            s += '#external {0}.'.format(a)
        prg.add('external', ['t'], s)
        prg.ground([('external', [t])])
        prg.ground([('heuristic', [t])])
        for a in heu_atoms:
            prg.assign_external(a, True)
        prev_heu_atoms = copy.copy(heu_atoms)
        heu_atoms.clear()

        sol_tmp, cost_tmp = solver.run(iter_option)
        if cost_tmp < cost:
            sol, cost = sol_tmp, cost_tmp
        if cost_tmp < cost_best:
            sol_best, cost_best = sol_tmp, cost_tmp

    print(sol_best)
```

コード 5.4: LNPS プログラム (メイン関数のみ)

- 終了条件が満たされるまで以下のステップを繰り返す (19 行目).
- 反復回数を 1 増やす (20 行目).
- 暫定解の中で再利用したいアトムの情報を持った新しいアトム（以下，heu アトム）を生成し，リスト heu_atoms に加えていく (21–24 行目).
  - heu アトムに真が割り当てられることによって #heuristic 文が有効になり，再利用したいアトムに優先的な値割り当てが行われるようになっている.
- 前回の反復での heu アトムを削除する (25–26 行目).
- 生成した heu アトムの頭に #external を付けて動的にプログラムに追加する (28–31 行目).
  - #external によって宣言されたアトムは，実行中でのプログラムへの動的な追加・削除が可能になる.
- まだ基礎化されていない追加されたプログラムをそれぞれ基礎化する (32–33 行目).
- heu アトムに真を割り当てる (34–35 行目).
- heu アトムの入ったリストを別のリストへコピーして空にする (36–37 行目).
- *re-search* を行って解（x^t）を得る (39 行目).
- x^t が受理条件を満たしていれば受理する (40–41 行目).
  - 以前受理された解より改善された解であれば受理している.
- x^t が暫定解より改善された解であれば暫定解を更新する (42–43 行目).
- 暫定解を出力する (45 行目).

第6章　評価実験

提案手法の有効性を評価するためにカリキュラムベース・コース時間割問題に対する実行実験を行なった.

6.1　destroy 演算

LNPS の *destroy* については，CB-CTT の既存研究 [7] を参考に，以下の 3 種類を実装した.

- Random N (R-N)
  暫定解から変数の値割当ての $N\%$をランダムに選んで取り消す.
- Day-Period (DP)
  曜日 D と時限 P をランダムに 1 組選び，暫定解から D 曜 P 限に関する変数の値割当てをすべて取り消す.
- Day-Room (DR)
  曜日 D と教室 R をランダムに 1 組選び，暫定解から D 曜日の R 教室に関する変数の値割当てをすべて取り消す.

また，上記の destroy 演算を参考に以下の 2 種類を新たに考案し実装した．いずれも，科目に対する曜日と時限の割当てはできるだけ維持することで，教室の変更を促し，教室に関するソフト制約への違反を減らすことを狙いとしている.

- Swap-Room N (SR-N)
  Random N と同様に，暫定解から変数の値割当ての $N\%$をランダムに選んで取り消す．ただし，科目に対する曜日と時限の割当てはできるだけ維持する.

- Day-Period-Swap-Room N (DPSR-N)
  Day-Period と同様に，曜日 D と時限 P をランダムに N 組選び，暫定解から D 曜 P 限に関する変数の値割当てをすべて取り消す．ただし，科目に対する曜日と時限の割当てはできるだけ維持する.

6.2　実験概要

比較した手法は以下の 2 つである．

- 既存手法: ASP ソルバー *clingo*
  - 分子限定法による探索（以下 bb）と充足不能コアを用いた探索（以下 usc）の 2 通りで実行を行い，オプションは先行研究 [1] によって結果の良かったものを使用した．
- 提案手法: LNPS を *clingo* 上に実装
  - destroy 演算子：R-0/3/5, DP, DR, SR-5/10, DPSR-1/2/3 の 10 種類
  - 初期解探索のオプション：既存手法の usc
  - 初期解探索の打切り：conflict 数が 250 万以上，または restart 数が 5,000 以上
  - *re-search* の打切り：conflict 数が 3 万以上

ベンチマーク問題には，国際時間割競技会 ITC2007 [1] で公開されているカリキュラムベース・コース時間割 (CB-CTT) の問題集 (全 21 問) と，主に競技会以降に追加された問題集 (全 40 問) に対して，ソフト制約が最も多い UD5 を使って評価を行なった [4, 3]．CB-CTT 問題を解くための ASP 符号化には，teaspoon 符号化 [1] を使用した．

既存手法と提案手法ともに，ASP ソルバーには *clingo*-5.4.0 を利用し，1 問あたりの制限時間は 1 時間とした．実験環境は，Mac OS, 3.2GHz Intel Core i7, 64GB メモリである．

6.3　実験結果

表 6.1 に，競技会使用問題に対して各手法で得られた最適値および最良値を示す．提案手法 LNPS については，3 回の実験で得られた最も良い値が記されている．各問題ごとに全手法の中で最も良かった値を太字で表している．‘$*$’ 付きの値は最適値を意味する．提案手法は，1 問を除くすべての問題に対して，既存手法と同じかより良い解を得ている．各手法について，最適値および最良値を求めた問題数を比較すると，提案手法 DPSR-1 と DPSR-1 が 9 問と最も多く，次いで SR-10 が 7 問であった．

表 6.2 に，他のアプローチとの比較結果を示す．左から順に，問題名，既知の最良値 ($\sharp$)，既存手法 ASP の最良値と $\sharp$ との比，提案手法 LNPS の最良値と $\sharp$ との比が示されている．既知の最良値は，これまで，メタ戦略に基づく各種アルゴリズム，整数計画法，SAT, MaxSAT, SMT など様々な手法で求められた結果である [1]．既存手法の最良値は，各問

[1] http://www.cs.qub.ac.uk/itc2007/

題に対して，表 6.1 中の既存手法 (2 種類) で得られた値の中で良かった値を示している．同様に提案手法の最良値も，各問題に対して，表 6.1 中の提案手法 (10 種類) で得られた値の中で最も良い値を示している．また，既知の最良値との比は，以下の計算結果を百分率で表したものである．

$$\text{既知の最良値との比} = \frac{\text{得られた最良値} - \text{既知の最良値}}{\text{既知の最良値}}$$

既存手法 ASP は，既知の最良値との比が $+437\%$ と高く，解の精度が悪いことが確認できる．一方，提案手法 LNPS は，既知の最良値との比が $+16\%$ まで抑えられており，既存手法と比較して約 27 倍改善されている．さらに，提案手法は，comp07, comp09, comp13, comp18 について，既知の最良値を更新することに成功した．

表 6.3 に，競技会で使用されていない問題に対して各手法で得られた最適値および最良値を示す．各問題ごとに全手法の中で最も良かった値を太字で表している．‘$*$’ 付きの値は最適値を意味する．各手法について，最適値および最良値を求めた問題数を比較すると，提案手法 DPSR-2 が 19 問と最も多く，次いで SR-5, DPSR-1, DPSR-3 が 17 問であった．

表 6.1: 実験結果: 競技会使用問題で得られた最適値と最良値

問題名	既存手法 ASP		提案手法 LNPS									
			R			DP	DR	SR		DPSR		
	bb	usc	0	3	5			5	10	1	2	3
comp01	129	283	13	**11**	**11**	**11**	**11**	**11**	**11**	**11**	**11**	**11**
comp02	1049	331	259	239	199	**172**	235	201	213	260	227	236
comp03	791	302	173	173	154	149	149	144	**143**	145	154	144
comp04	231	*__49__	*__49__	*__49__	*__49__	*__49__	*__49__	*__49__	*__49__	*__49__	*__49__	*__49__
comp05	2662	1940	1102	922	797	864	841	891	861	994	**776**	907
comp06	822	1025	216	162	135	135	166	112	106	119	**102**	123
comp07	924	1149	153	131	114	99	105	60	65	56	74	**40**
comp08	348	*__55__	*__55__	*__55__	*__55__	*__55__	*__55__	*__55__	*__55__	*__55__	*__55__	*__55__
comp09	617	254	154	154	151	143	146	145	146	141	**138**	141
comp10	822	1229	209	166	141	124	133	97	**80**	97	98	101
comp11	287	*__0__	*__0__	*__0__	*__0__	*__0__	*__0__	*__0__	*__0__	*__0__	*__0__	*__0__
comp12	2626	1246	787	740	728	705	694	729	**664**	687	718	702
comp13	661	301	171	158	163	168	165	147	152	147	149	**146**
comp14	748	*__67__	189	156	145	143	158	104	83	123	130	128
comp15	852	607	232	214	213	227	206	210	205	**198**	212	213
comp16	944	1090	197	156	168	140	162	167	151	**134**	149	172
comp17	979	412	244	226	211	209	214	196	204	200	**184**	203
comp18	673	471	180	156	148	151	155	140	149	146	**136**	149
comp19	890	919	231	192	190	165	199	176	163	**144**	174	171
comp20	3304	1386	373	274	356	268	273	272	246	**237**	283	291
comp21	893	310	234	210	202	222	214	166	167	**161**	192	170
#最適値・最良値	0	4	3	4	4	5	4	4	7	9	9	6

表 6.2: 他のアプローチとの比較

問題名	既知の最良値 (♯)	既存手法 ASP		提案手法 LNPS	
		最良値	♯との比 (%)	最良値	♯との比 (%)
comp01	11	129	+1,072	11	0
comp02	130	331	+154	172	+32
comp03	142	302	+112	143	+1
comp04	49	49	0	49	0
comp05	570	1,940	+240	776	+36
comp06	85	822	+867	102	+20
comp07	42	924	+2,100	40	**-5**
comp08	55	55	0	55	0
comp09	150	254	+69	138	**-8**
comp10	72	822	+1,041	80	+11
comp11	0	0	0	0	0
comp12	483	1,246	+157	664	+37
comp13	147	301	+104	146	**-1**
comp14	67	67	0	83	+23
comp15	176	607	+244	198	+13
comp16	96	944	+883	134	+40
comp17	155	412	+165	184	+19
comp18	137	471	+243	136	**-1**
comp19	125	890	+612	144	+15
comp20	124	1,386	+1,017	237	+91
comp21	151	310	+105	161	+7
♯との比の平均			+437		+16

表 6.3: 実験結果: 競技会使用問題以外で得られた最適値と最良値

問題名	既存手法 ASP		提案手法 LNPS									
			R			DP	DR	SR		DPSR		
	bb	usc	0	3	5			5	10	1	2	3
DDS1	6536	7544	4864	2993	3317	2780	3047	2834	2879	3076	2739	**2700**
DDS2	407	433	100	81	90	**68**	76	79	70	78	74	**68**
DDS3	**22**	391	36	28	**22**	**22**	**22**	**22**	**22**	**22**	**22**	**22**
DDS4	10486	3123	1443	3056	2990	2690	1595	1305	1539	1183	**1162**	1245
DDS5	539	**76**	**76**	**76**	**76**	**76**	**76**	**76**	**76**	**76**	**76**	**76**
DDS6	849	964	208	174	187	180	160	**158**	203	162	187	202
DDS7	645	1485	120	75	363	87	67	65	68	74	**58**	66
EA01	4708	807	269	226	226	225	249	222	216	**201**	204	217
EA02	1094	1215	140	146	132	135	**130**	165	132	169	137	136
EA03	8192	2555	586	1411	1486	829	**508**	775	789	901	882	834
EA04	**63**	1873	388	1309	1611	681	359	130	842	169	168	120
EA05	26	513	87	16	21	29	18	**14**	**14**	**14**	**14**	**14**
EA06	543	1027	235	208	179	166	172	140	**125**	169	161	164
EA07	10122	2831	995	1653	2624	1199	851	**671**	722	750	701	705
EA08	480	991	166	60	685	60	48	**40**	42	48	42	44
EA09	**48**	865	**48**	53	53	50	52	**48**	**48**	**48**	**48**	**48**
EA10	1711	444	355	456	1235	**273**	306	420	427	410	396	382
EA11	232	771	123	54	59	82	57	40	40	**36**	41	40
EA12	234	518	79	43	45	51	42	**27**	**27**	**27**	**27**	**27**
erlangen2011_2	17035	17263	**16772**	**16772**	**16772**	**16772**	**16772**	**16772**	**16772**	**16772**	**16772**	**16772**
erlangen2012_1	32541	**25061**	25538	25538	25538	25538	25538	25538	25538	25538	25538	25538
erlangen2012_2	38180	34360	**34218**	**34218**	**34218**	**34218**	**34218**	**34218**	**34218**	**34218**	**34218**	**34218**
erlangen2013_1	29459	28302	**27923**	**27923**	**27923**	**27923**	**27923**	**27923**	**27923**	**27923**	**27923**	**27923**
erlangen2013_2	34568	**29140**	30117	30117	30117	30117	30117	30117	30117	30117	30117	30117
erlangen2014_1	30249	24510	**23461**	**23461**	**23461**	**23461**	**23461**	**23461**	**23461**	**23461**	**23461**	**23461**
test1	619	751	334	**232**	**232**	**232**	**232**	**232**	**232**	**232**	**232**	**232**
test2	24	**20**	**20**	21	**20**	**20**	**20**	**20**	**20**	**20**	**20**	**20**
test3	196	117	79	79	78	79	79	**71**	76	73	73	76
test4	705	260	181	172	144	138	150	159	165	140	**117**	148
toy	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
Udine1	1079	953	256	225	250	212	223	213	228	218	199	**185**
Udine2	505	402	133	106	121	108	94	118	117	**89**	93	122
Udine3	343	385	119	100	73	**62**	93	88	96	87	71	69
Udine4	400	**106**	**106**	**106**	**106**	**106**	**106**	**106**	**106**	**106**	**106**	**106**
Udine5	518	695	103	77	72	67	75	80	66	80	**61**	76
Udine6	366	628	156	54	45	43	47	38	40	38	**36**	41
Udine7	233	733	80	77	81	70	77	73	**67**	76	72	76
Udine8	606	941	110	105	112	108	110	111	109	105	100	**96**
Udine9	610	163	87	74	72	70	74	69	63	65	**62**	72
UUMCAS_A131	38929	28688	27397	27911	28081	27099	26843	27364	28037	**24579**	25342	26832
#最適値・最良値	4	6	9	8	10	13	12	17	15	17	19	17

第7章　おわりに

本論文では，系統的探索と確率的局所探索を統合的に適用する手法である優先度付き巨大近傍探索 (LNPS) を ASP 上に実装について述べた．開発ソルバー *asprior* は提案手法 LNPS を高速 ASP ソルバー *clingo* 上に実装し，国際時間割競技会で使用された問題を含む CB-CTT 問題集 (全 61 問) を用いて性能評価を行った．その結果，既知の最良値との比について，通常の ASP 解法が $+437\%$ であったのに対し，提案手法 LNPS は，その比を $+16\%$ まで大幅に改善できた．さらに，6 問について，既知の最良値を更新することに成功した．今後の課題としては，アダプティブ LNPS への拡張や他の時間割問題や組合せ最適化問題への適用などが挙げられる．

参考文献

[1] Mutsunori Banbara, Katsumi Inoue, Benjamin Kaufmann, Tenda Okimoto, Torsten Schaub, Takehide Soh, Naoyuki Tamura, and Philipp Wanko. teaspoon: Solving the curriculum-based course timetabling problems with answer set programming. *Annals of Operations Research*, Vol. 275, No. 1, p. 3–37, 2019.

[2] Chitta Baral. *Knowledge Representation, Reasoning and Declarative Problem Solving*. Cambridge University Press, 2003.

[3] Alex Bonutti, Fabio De Cesco, Luca Di Gaspero, and Andrea Schaerf. Benchmarking curriculum-based course timetabling: formulations, data formats, instances, validation, visualization, and results. *Annals of Operations Research*, Vol. 194, No. 1, pp. 59–70, 2012.

[4] Luca Di Gaspero, Barry McCollum, and Andrea Schaerf. The second international timetabling competition (ITC-2007): Curriculum-based course timetabling (track 3). Technical report, Queen's University, Belfast, United Kingdom, 2007.

[5] E. Erdem, M. Gelfond, and N. Leone. Applications of ASP. *AI Magazine*, Vol. 37, No. 3, pp. 53–68, 2016.

[6] Michael Gelfond and Vladimir Lifschitz. The stable model semantics for logic programming. In *Proceedings of the Fifth International Conference and Symposium on Logic Programming*, pp. 1070–1080. MIT Press, 1988.

[7] Alexander Kiefer, Richard F. Hartl, and Alexander Schnell. Adaptive large neighborhood search for the curriculum-based course timetabling problem. *Annals of Operations Research*, Vol. 252, p. 255–282, 2017.

[8] 桑原和也, 田村直之, 番原睦則. 解集合プログラミングに基づく系統的探索と確率的局所探索の統合的手法に関する一考察. 2021 年度人工知能学会全国大会 (第 35 回) 論文集, pp. 2E1–OS–13a–01, 2021.

[9] Ilkka Niemelä. Logic programs with stable model semantics as a constraint programming paradigm. *Ann. Mathematics and Artificial Intelligence*, Vol. 25, No. 3–4, pp. 241–273, 1999.

[10] David Pisinger and Stefan Ropke. Large neighborhood search. *Handbook of Metaheuristics*, Vol. 146, pp. 399–419, 2010.

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.