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

A Study on ASP-based Integration of Systematic and Stochastic Local Search

桑原 和也 *1　番原 睦則 *1　田村 直之 *2
Kazuya Kuwahara　Mutsunori Banbara　Naoyuki Tamura

*1名古屋大学　*2神戸大学
Nagoya University　Kobe University

In this publication, I suggest technique to apply systematic and stochastic local search for combinatorial optimization problem by using Answer Set Programming, one of the developed form of SAT. The suggestion is based on Large Neighborhood Search, a kind of approximate solution method. LNS is iterative solution method, that cancels a part of assignment of variables included in the solution at random and rebuild solution by reassign only the variables. In the suggestion, I permit reassignment to the variable that was not canceled by replacing rebuilding solution with re-search that maintained assignment if possible. In this way, I can perform the search that does not depend too much which assignment you cancel. I implemented suggestion technique on ASP solver clingo and evaluate performance using benchmarks of the International Timetabling Competition (all 21 problems). As a result, the suggestion was able to get a better solution for many problems in comparison with normal ASP. In addition, I succeeded in updating the known best value at one problem.

1. はじめに

本発表では，SAT の発展形の一つである解集合プログラミング (Answer Set Programming; ASP) 技術を用い，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手法を提案する．

提案手法は，近似解法の一種である巨大近傍探索 (Large Neighborhood Search; LNS) のアイデアをベースにしている．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．提案手法では，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．

提案手法を ASP ソルバー clingo 上に実装し，国際時間割競技会の問題集 (全 21 問) を用いて性能評価を行った．その結果，提案手法は，通常の ASP 解法と比較して，多くの問題に対してより良い解を得ることができた．また，1 問について，既知の最良値を更新することに成功した．

2. 巨大近傍探索 (Large Neighborhood Search; LNS)

2.1 LNS の概要

LNS は確率的局所探索による近似解法の一種である．LNS では問題に対する暫定解を求め，暫定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解の再構築を繰り返す．LNS の特徴として，解の再構築では，暫定解のうち取り消された変数に対してのみ再割り当てが行われ，その他の変数に対する値割り当ては変化しないという点が挙げられる．

連絡先: 氏名，所属，住所，電話番号，Fax 番号，電子メールアドレスなど

2.2 LNS のアルゴリズム

LNS のアルゴリズムを以下の Algorythm 1 に示す．

Algorythm 1 Large neighborhood search

```
1: input: a feasible solution x
2: x^b = x;
3: repeat
4:       x^t = repair(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 行目では，初期解を求め x と置く．2 行目で最良解 $x^b = x$ として 3 行目以下のループに入る．4 行目では，以下の destroy と repair で x から得られた解を x^t とする．

- destroy : x から値割当ての一部を取り消し x' とする．
- repair : x' の値割当てを変化させずに解を再構築する．

5〜7 行目では，x^t を受理する条件を満たしている場合に $x = x^t$ とする．受理条件には,「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 より改善された解である場合に $x^b = x^t$ とする．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して 12 行目に進み，そうでなければ 4 行目に戻る．終了条件には，繰り返し回数や制限時間などを用いる．12 行目では，最良解 x^b を返して終了する．

3. 優先度付き巨大近傍探索 (Large Neighborhood Prioritized Search; LNPS)

3.1 LNPS の概要

LNS では，取り消された変数に対してのみ再割当てが行われ，他の変数に対する値割り当ては変化しない．そこで，LNS における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，取り消されなかった変数への再割当ても許す，優先度付き巨大近傍探索 (Large Neighborhood Prioritized Search; LNPS) を提案する．

詳細は後述するが，今回実装に用いた解集合プログラミング (Answer Set Programming; ASP) 技術によって，値割当てをなるべく維持したままでの再探索が自然に実現できる．また，ASP では解の再探索を系統的探索で行うことができる．

3.2 LNPS のアルゴリズム

LNPS のアルゴリズムを以下の Algorythm 2 に示す．

Algorythm 2 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
```

LNS とは 4 行目だけが異なっており，LNPS 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 : x から値割当ての一部を取り消し x' とする．
- re-search : x' の値割当てをなるべく維持したまま解を再探索する．

値割り当てを固定しない再探索の実現によって，暫定解のどの値割り当てを取り消すかに依存しすぎない探索を行うことが可能である．

4. ASP ソルバー *clingo* 上での実装

4.1 解集合プログラミング (Answer Set Programming; ASP)

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．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．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．

LNPS の実装には ASP システムとしては *clingo* を用いた．以下では，*clingo* に存在する機能の一つで，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機能について説明を行う．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 1 に示す．

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

表 1: 論理プログラムとソースコードにおける記号の対応

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

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

コード 1: 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 2: heu.lp に対する *clingo* の実行例

変数選択ヒューリスティクスの変更には，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．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，$false$ 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．

コード 1 は，*#heuristic* ルールを用いた論理プログラムの例である．1 行目には拡張構文である選択子が使用されており，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* が同時に真になることを禁止している．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．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 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．*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，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．また，コード 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．

4.2 LNPS の実装

LNPS の実装にはプログラミング言語 Python を用い，実行には *clingo* の Python インターフェースを利用する．最適化を行う任意の ASP プログラムに適用して，LNPS による探索を可能にするシステムを実現した．ASP システムに対し，任意の問題を解く為の ASP プログラムと Python で記述された LNPS プログラムを与えることで，問題の解を得られるという仕組みになっている．

実際にどのような動作が行われているかを順を追って説明する．まず既存 ASP 解法によって問題を解き，ある探索中断基準に基づいて探索を中断し，暫定解を得る．得た暫定解に対して，destroy 演算によって値割り当ての一部を取り消す．取り消されなかった値割り当てに対して，次回探索開始時に優先的に真となるように *#heuristic* ルールを生成する．ルールを有効にした状態で re-search による再探索を行う．これにより，値割り当てをなるべく維持したままの再探索が実現され，再度ある基準で再探索を中断，値割り当てを取り消すという動作を繰り返す．

4.3 destroy 演算の実装

LNPS では，暫定解のどの値割り当てを取り消すかに依存しすぎない探索を行うことが可能であると述べたが，それでも LNPS の性能には，適切な destroy 演算を設計する必要がある．今回実装した複数の destroy 演算の説明をするにあたって，解く問題の性質を利用したものもあるため，まず簡単に今回使用した問題である時間割問題について述べる．

本研究で対象とした時間割問題は，カリキュラムベース・コース時間割問題で，時間割問題の中でも最も研究が盛んな時間割問題の一つである．カリキュラムベース・コース時間割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，違反する制約の重み（ペナルティ）の最小化が目的である．ハード制約を満たしながら与えられた講義をそれぞれの日時と教室に割り当て，「受講者数が，教室の収容人数を超えた分だけペナルティが与えられる」などといったソフト制約によって目的関数の値が決まる．

次に実装した 4 種類の destroy 演算について説明を行う．

1. random N
   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．N として 0 も選択可能であり，その場合，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が狙いとなる．値割り当てをある程度の割合で取り消す場合は，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
   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 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
   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 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．

4. swap-room N
   暫定解からランダムに N%の値割り当てを選び，曜日と時限はそのままで，割り当てられている教室の情報を取り消す．この destroy 演算は，day-room と比較して割

り当てられる教室のみの変更を促進させ，より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．

5. 評価実験

提案手法の有効性を評価するために，実行実験を行った．ベンチマーク問題には，国際時間割競技会 ITC2007 の comp01〜21 の全 21 問を使用し，ソフト制約の多い問題集 (UD5) を使用した．実験に使用した手法としては，既存 ASP 解法のみでの実行に加え，LNPS と 各種 destroy 演算を加えたもので行い，destroy 演算は，random N ($N = 0, 3, 5$)，day-period，day-room，swap-room N ($N = 5, 10$) を選択した．表 2 は，それぞれの手法で得られた最適値・最良値を示している．左の列から順に，問題名，既存 ASP 解法によって得られた値，各種提案手法によって得られた値となっている．各問題ごとに実験した手法の中で最も良い値が得られたものは太字で示してある．それぞれの手法で良い値が得られた問題数を比較すると，順に swap-room 5，swap-room 10 が良い性能を示している．表 3 では別のアプローチとの比較を行っている．既知の最良値は先行研究によって得られた解の上界を表しており，提案ベストの値は，各種提案手法の中で最も良かった値としている．それぞれ最良値との比では，既知の最良値を 100% として，得られた解での増減を表している．既存 ASP 解法では +437% と 5 倍以上となっているが，提案手法では +26%にまで抑えられている．また comp09 については -3% と既知の最良値を更新できた．

6. おわりに

参考文献

[Knuth 84] Knuth, D. E.: The TEXbook, Addison-Wesley (1984), (邦訳：TEX ブック, 斎藤 信男 監修, 鷺谷 好輝 訳, アスキー出版局 (1992)).

[Lamport 86] Leslie, L: LATEX: A Document Preparation System (Updated for LATEX2ε), Addison-Wesley, 2nd edition (1998) (邦訳：文書処理システム LATEX2ε, 阿瀬 はる美 訳, ピアソン・エデュケーション, (1999)).

表 2: 得られた最適値・最良値

問題名	既存 ASP	提案手法						
		R-0	R-3	R-5	DP	DR	SR-5	SR-10
comp01	129	13	13	**11**	13	**11**	**11**	**11**
comp02	331	334	239	273	**172**	242	201	245
comp03	302	173	177	154	149	173	**146**	157
comp04	**49**	**49**	**49**	**49**	**49**	**49**	**49**	**49**
comp05	1940	1124	922	**797**	926	1116	891	861
comp06	822	220	166	135	140	204	118	**106**
comp07	924	225	131	137	118	129	**72**	77
comp08	**55**	**55**	**55**	**55**	**55**	**55**	**55**	**55**
comp09	254	155	154	158	149	146	**145**	151
comp10	822	231	220	177	167	133	**97**	109
comp11	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
comp12	1246	834	740	728	705	**694**	756	809
comp13	301	183	180	163	168	165	**151**	155
comp14	**67**	189	186	145	179	165	104	83
comp15	607	244	238	**213**	234	238	215	224
comp16	944	197	**156**	178	180	162	223	158
comp17	412	254	226	259	230	234	**199**	208
comp18	471	191	170	168	152	158	**144**	149
comp19	890	231	192	197	187	219	176	**163**
comp20	1386	373	274	356	280	305	293	**265**
comp21	310	285	219	202	222	235	192	**178**
最適 (良) 値の数	4	3	4	6	4	5	**11**	8

表 3: 既知の最良値との比較

問題名	既知の最良値	既存 ASP		提案ベスト	
		値	最良値との比	値	最良値との比
comp01	11	129	+1072%	11	+0%
comp02	130	331	+154%	172	+32%
comp03	142	302	+112%	146	+2%
comp04	49	49	+0%	49	+0%
comp05	570	1940	+240%	797	+39%
comp06	85	822	+867%	106	+24%
comp07	42	924	+2100%	72	+71%
comp08	55	55	+0%	55	+0%
comp09	150	254	+69%	145	**-3%**
comp10	72	822	+1041%	97	+34%
comp11	0	0	+0%	0	+0%
comp12	483	1246	+157%	694	+43%
comp13	147	301	+104%	151	+2%
comp14	67	67	+0%	83	+23%
comp15	176	607	+244%	213	+21%
comp16	96	944	+883%	156	+62%
comp17	155	412	+165%	199	+28%
comp18	137	471	+243%	144	+5%
comp19	125	890	+612%	163	+30%
comp20	124	1386	+1017%	265	+113%
comp21	151	310	+105%	178	+17%
比の平均			+437%		+26%