1D5-OS-11b-6in

SAT ソルバーと密に結合された制約プログラミングシステム Scarab とハミルトン閉路問題への応用

SAT-based CP System Scarab and its Application to Hamiltonian Cycle Problem

宋 剛秀[*1] Takehide Soh　Daniel Le Berre[*2]　Stéphanie Roussel[*2]　番原 睦則[*1] Mutsunori Banbara　田村 直之[*1] Naoyuki Tamura

[*1]神戸大学 情報基盤センター
Information Science and Technology Center, Kobe University

[*2]CRIL-CNRS UMR 8188, Université d'Artois

In this paper, we present Scarab, a constraint programming DSL in Scala tightly integrated with the SAT solver Sat4j. The tight integration results in quick feedback from the solver. By using Scarab DSL, users can manipulate many low level functionality of the SAT solver such as specifying the representation of the constraints without writing complex programs. In addition, well known advanced SAT techniques such as incremental solving, assumption based satisfiability, and dynamic clause addition are available from Scarab DSL. As an application of Scarab, we demonstrate an incremental SAT-based method for solving the Hamiltonian cycle problem. The increase of clauses often prevents SAT-based methods from being scalable. Our method reduces the number of clauses in the encoding by relaxing permutation constraints and using the native support for cardinality constraint which is a low level function of Sat4j. An experimental evaluation is performed on 5 benchmark sets and we succeeded in improving either the number of instances solved or the computation time compared to the state of the art solvers.

1. はじめに

近年の SAT ソルバーの劇的な進歩を背景に SAT 型システムが活発に研究されている [Tamura 09, Metodi 12]. これらの SAT 型システムは節集合ファイルを入力するとモデルが出力されるという SAT ソルバーの高レベルの機能を主に利用しており, SAT ソルバーをある種のブラックボックスとして扱ってきた. しかし, Sat4j をはじめ SAT ソルバー内部には問題解法に役に立つと考えられる低レベルな機能が備わっており, それらを外部から利用できるようにすることによって SAT 型システムの性能を向上できる可能性がある.

本稿では SAT ソルバー Sat4j と密に結合された SAT 型制約プログラミングシステム Scarab とその応用例として抽象精密化法を用いたハミルトン閉路問題の解法を提案する. Scarab と Sat4j は共に Java 仮想マシン上で動作し, ファイルの読み書きやプロセス間通信を行わずに低負荷で連携することが可能である. また Scarab のドメイン特化言語 (DSL) を用いることで Sat4j における組込み基数制約などの低レベルの関数をユーザは複雑なプログラムを書くことなく利用できる.

ハミルトン閉路問題 (HCP) は与えられたグラフの全頂点を通る閉路を求める問題であり, 計算機科学において長年研究されてきた巡回セールスマン問題とも関連がある重要な問題である. HCP の SAT 型解法はこれまでも提案されてきたが, 500 頂点規模のグラフは解くのが困難であった [Prestwich 03, Velev 09]. 主な原因として順列を表す制約が頂点の数 n に対して n^3 で増加することがある. 提案方法では, ソフトウェア検証などで用いられてきた反例を基にした抽象精密化法 (CEGAR) を応用して, この順列制約を抽象精密化する方法を Scarab と Sat4j の組込み基数制約を用いて実装し, HCP に適用する.

提案方法の性能を評価するために 474 個のグラフを用いて計算機実験を行った. 結果として, 従来の SAT 型解法および最新の TSP ソルバーとの比較の結果, Scarab 上に実装した提案方法がより高速に多くの問題を解くことに成功した.

連絡先: 宋 剛秀, 神戸大学情報基盤センター, 〒 657-8501 兵庫県神戸市灘区六甲台町 1-1, soh@lion.kobe-u.ac.jp

2. Sat4j と Scarab

Sat4j [Le Berre 10] は SAT, Max-SAT, 擬似ブール制約など命題論理およびその拡張における推論技術のライブラリである. 多くの SAT ソルバーの基礎である Minisat [Eén 03] の Java における実装として 2004 年に開発が始まり, 機能拡張が続けられている. Sat4j は Minisat の設計でも特に初期の設計を引き継いでおり SAT 問題における節だけでなく, 他の種類の制約を扱うことができる. 組込み基数制約もその 1 つであり, 充足解探索手続きの中で節と同じように n 個のブール変数 $x_i \in \{0,1\}$ に対する基数制約 $\sum_{i=1}^{n} x_i \geq k$ (k は閾値) を扱うことができる.

Scarab は制約プログラミングのためのドメイン特化言語 (DSL) である Scarab DSL, SAT 符号化モジュール, そしてバックエンドの SAT ソルバーへのインターフェースから構成される. Scarab の設計方針は SAT 型システム開発者に表現性, 効率性, 変更性, 可搬性を備えたワークベンチを提供することである. **(表現性)** Scarab DSL と Scala の両方を用いて与えられた問題を記述することが可能である. **(効率性)** Scarab は 2008 年, 2009 年に CSP ソルバー競技会のグローバル部門で優勝した Sugar に採用されている順序符号化法 [Tamura 09] を用いているという点で効率的である. **(変更性)** Scarab では SAT 型システム開発者が自前の制約を定義し, 変更・改良することが可能である. また Scarab のソースコードは全体で 800 行ほどであり, Scarab 本体の変更も可能である. **(可搬性)** Scarab と Sat4j は両方とも Java 仮想マシン上で動作し, 可搬性のあるシステムを実現可能である.

図 1 に Scarab の構成図を示す. もっとも基本的な Scarab の処理の流れは次のようになる. まずユーザが Scala と Scarab DSL を用いて記述したプログラム (Scarab プログラム) によって CSP オブジェクトが生成される. 次に Scarab プログラムによって CSP ソルバーが API を通して呼ばれた時に, CSP オブジェクトは SAT オブジェクトへと変換される. 続いて CSP ソルバーから SAT ソルバー Sat4j が API を通して実行される. 解が存在すれば逆符号化を通して CSP の解が返される.

Scarab では順序符号化法を用いて Sequential Counter や

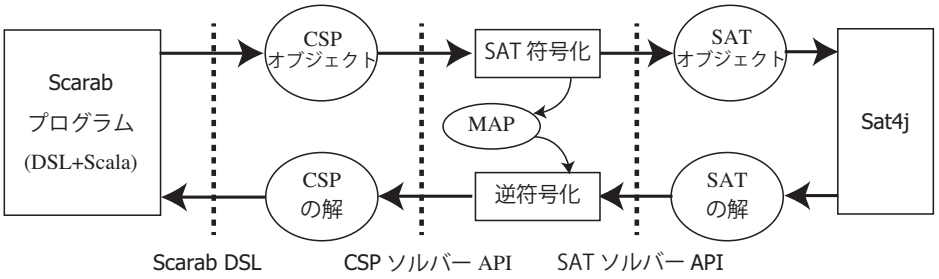

図 1: Scarab の構成

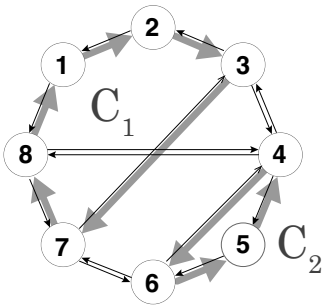

図 3: 反例: 解が複数の閉路を含む場合

Totalizer など様々な基数制約の SAT 符号化法を実装することができる. また Sat4j の低レベルな機能である組込み基数制約を Scarab DSL を用いて利用することが可能である.

3. ハミルトン閉路問題への応用

3.1 ハミルトン閉路問題と従来手法

本研究では無向グラフにおけるハミルトン閉路問題 (HCP) を対象とする. V を n 個の頂点集合, E を辺の集合とし, グラフを $G = (V, E)$ で表す. ここで簡潔な制約モデリングのため辺集合 E に対応する弧の集合 $A = \{(i,j), (j,i) \mid \{i,j\} \in E\}$ を導入する. ブール変数 $x_{ij} (i \neq j)$ はそれぞれの弧 $(i,j) \in A$ に対して定義され, (i,j) が解の閉路に含まれる時に $x_{ij} = 1$ となる. ここで $x_{ij} + x_{ji} \leq 1$ が成り立つものとする. HCP に対する素朴な制約モデルは以下になる.

$$\sum_{(i,j)\in A} x_{ij} = 1 \qquad \text{各頂点 } i = 1, \ldots, n \text{ に対して.} \quad (1)$$

$$\sum_{(i,j)\in A} x_{ij} = 1 \qquad \text{各頂点 } j = 1, \ldots, n \text{ に対して.} \quad (2)$$

$$\sum_{i,j\in S} x_{ij} \leq |S| - 1, \qquad S \subset V,\ 3 \leq |S| \leq n-2 \quad (3)$$

制約 (1) と (2) は基数制約であり閉路に含まれる弧は各頂点にちょうど 1 回入り，1 回出て行く必要があることを表している. 制約 (3) は部分閉路が構成されることを禁止する. これまでいくつかの SAT 型解法が HCP に対して提案されている. Prestwich はグラフ中の任意の 3 頂点の順列に対する遷移関係を使って制約 (3) を SAT 符号化した [Prestwich 03]. しかし, これは n^3 の節が必要になる. Velev と Gao は Prestwich の方法を踏襲しながらグラフの三角化を事前に行うことで節数を削減することに成功している [Velev 09]. しかし, この方法でも 500 頂点以上のグラフについてはハミルトン閉路を求めることは困難であった.

3.2 反例を基にした抽象精密化法

これまでソフトウェア検証などにおいて，制約の数の増大を防ぐために制約を抽象化したえで得た解 (反例) を用いて新たな制約を生成し徐々に制約を元の制約へと精密化していく方法が, 反例を基にした抽象精密化法 (CEGAR) と呼ばれ研究されている [Clarke 00]. 前節で述べたように, HCP の SAT 型解法では制約 (3) の SAT 符号化で節の数が大きく増加する. 本稿では制約 (3) を抽象化したうえで, 元の解に近づくよう制約を追加していく方法を提案する.

図 2 にグラフ G が与えられた時の提案方法の手続きを示す. まず始めに, 抽象化は制約 (3) を SAT 符号化しないことで行われる. すなわち, 制約 (1) と (2) だけが SAT 符号化され Ψ に代入される (1 行目). 次に SAT ソルバーが起動され, 抽象精密化法の繰返しが開始する (2 行目). Ψ が `UNSAT` であれば直ちに繰返しは終了し, 解が存在しないことが返される (8 行目). もし論理式 Ψ が `SAT` でその充足解が閉路を 1 つしか含まないのであれば, その閉路はハミルトン閉路であるので解として返す (4 行目). その他の場合, 解は複数の部分閉路を含んでおり, それらは反例を表している. 制約を精密化するために, ブロック節を Ψ に追加する. この時ブロック節は見つかった部分閉路を 1 つでも含むような解を以後防ぐように生成される. それぞれの閉路は弧の方向によって時計回りと反時計回りがブロックされる. この繰返しをハミルトン閉路が見つかるか, もしくは Ψ が `UNSAT` になるまで繰り返す.

```
1: Ψ := G における制約 (1) と (2) を SAT 符号化 ;
2: while (Ψ が充足可能)
3:     if (解が閉路を 1 つしか含まない)
4:         return G のハミルトン閉路;
5:     else
6:         Ψ_block := 反例からブロック節を生成 ;
7:     Ψ := Ψ ∧ Ψ_block ;
8: return ハミルトン閉路は存在しない;
```

図 2: HCP に対する反例を基にした抽象精密化法

図 3 は 8 頂点から構成されるグラフから得られた反例を表している. この場合, 以下の 4 つの節が出力される: $\neg x_{12} \vee \neg x_{23} \vee \neg x_{37} \vee \neg x_{78} \vee \neg x_{81}$ は C_1 の時計回り, $\neg x_{87} \vee \neg x_{73} \vee \neg x_{32} \vee \neg x_{21} \vee \neg x_{18}$ は反時計回り. 同様に節 $\neg x_{46} \vee \neg x_{65} \vee \neg x_{54}$ は C_2 の反時計回り, $\neg x_{45} \vee \neg x_{56} \vee \neg x_{64}$ は時計回りであり, C_1 と C_2 を含む解を以後防ぐために追加される. ここで注意されたいのは最悪の場合でもグラフ中の全ての閉路の生成を防がなくても良いということである. 例えば $(1,2,3,4,8)$ はブロックする必要がない. 何故ならば残った頂点 $\{5,6,7\}$ は閉路を構成できないからである.

4. 提案方法の評価

4.1 実験環境, ベンチマーク, 比較手法

提案方法を評価するために計算機実験を行った. 実験は Intel Xeon 2.93GHz を使った計算機上で制限時間 500 秒で行った. また使用メモリは Java 仮想マシンの設定 (`-Xms4g -Xmx4g`) によって 4GB に制限した.

実験では 5 つのベンチマークを用いて評価を行った. `color04` は DIMACS のグラフ彩色問題で用いられたグラフ集合であり, 頂点の数が 11 から 10000 までの 119 問で構成されてい

表 1: 符号化された変数, 節, 組込み基数制約の数

問題名	頂点数	辺数	全符号化法 [Velev 09]		提案方法 1 Seq. Counter + Sat4j		提案方法 2 組込み基数制約 + Sat4j	
			変数の数	節数	変数の数	節数	変数の数	制約数
DSJC500.5	500	62,624	248,690	194,186,195	375,744	1,063,608	125,248	64,624
latin_square	900	307350	—	—	1,844,100	5,223,150	614,700	310,950
qg.order100	10,000	990,000	—	—	—	—	1,980,000	1,030,000
rnd_100_01	100	325	2,516	80,481	1,950	5,325	650	725
rnd_1000_01	1,000	4,815	152,161	46,353,569	28,890	79,855	9,630	8,815
rnd_10000_01	10,000	62,199	—	—	373,194	1,037,383	124,398	102,199
complete100	100	4,950	14,850	3,253,901	29,700	83,950	9,900	5,366
complete400	400	79,800	239,400	212,055,601	478,800	1,358,906	159,600	81,400
complete1000	1,000	499,500	—	—	2,997,000	8,489,500	999,000	503,500

る．random はランダムに生成された問題であり頂点の数が 100 から 50000 までの 340 問で構成されている．これらのグラフ集合は相転移領域上のグラフになるようにパラメータ $m = cn(\ln n + \ln\ln n)/2$ を使って生成した．ここで m は辺の数であり，n は頂点の数，c は 1.08 から 1.10 の値をとる．complete は頂点数が 100 から 1500 までの完全グラフである．knight は騎士の巡回問題であり大きさは 8x8, 12x12, 20x20, 30x30, ..., 100x100 である．tsplib は TSPLIB における HCP のベンチマークである．提案方法の性能を評価するための比較に用いたシステムは以下の通りである．

全符号化法 全制約を SAT 符号化する従来手法 [Velev 09]

TSP ソルバー HCP ソルバーにもなる TSP ソルバー LKH

提案方法 1 抽象精密化法 + 疎な SAT 型システム (6 通り)
{ Sequential Counter, Totalizer } +
{ Minisat2.2, Glucose3, Lingeling-ats }

提案方法 2 抽象精密化法 + 密な SAT 型システム (3 通り)
{ Sequential Counter, Totalizer, 組込み基数制約 } +
{ Sat4j }

全符号化法は 3.1 節で説明した Velev と Gao による方法である．**TSP ソルバー**は DIMACS TSP チャレンジの問題全てに対する最良値の記録を持っている TSP ソルバー LKH である．LKH は HCP にも対応しており直接解くことが可能で入力されたグラフに対してハミルトン閉路もしくは解無しを返す．特に今回の実験で用いた最新バージョン LKH2.0.7 では HCP に対する性能改善が行われている．**提案方法 1** は抽象精密化法を疎な SAT 型システム上に実装したものである．ここで用いる 3 つの SAT ソルバーはいずれも競技会における優勝ソルバーであり現在最も高速と考えられる SAT ソルバーである．C++で実装されているが Scarab との連携はファイルの読み書きによって行われ, 組込み基数制約も使うことはできない．**提案方法 2** は抽象精密化法を密な SAT 型システム上に実現したものである．Sat4j は Java で実装されており速度面において他の SAT ソルバーよりも劣ると考えられるが Scarab との連携はメモリ上で行われ, 低レベルな機能である組込み基数制約を使うことができる．

4.2 SAT 符号化後の問題サイズの比較

前述したシステムの中でも SAT 型のものに対して SAT 符号化後の変数の数と節および制約の数の違いを測定した．ここでは全符号化法 [Velev 09] と提案方法 1 の Sequential Counter と提案方法 2 の組込み基数制約を比較した．Sequential Counter と同様の挙動を示したので Totalizer を用いた方法はここでは省略している．

表 1 はベンチマーク color04, random, complete からそれぞれ 3 つ取り出した例題の変数の数と節および制約の数を示している．各列は左から問題の名前, 頂点数, 辺の数を表し, 続いて 4 列目からはそれぞれの方法による変数の数と節および制約の数を表している．記号'—' は符号化が 500 秒で完了しなかったことを表している．

表より全符号化法 [Velev 09] では, 符号化後の節数が多いもので 2 億節に届くが，提案方法では 100 分の 1 以下に節の数を削減していることが分かる．しかしながら, 提案方法 1 の Sequential Counter を用いた場合にはグラフの頂点数が 400 程度になると節の数は 1 億節になり, qg.order100 については時間内に符号化を終えることができなかった．対照的に, 組込み基数制約を用いた提案方法 2 では頂点数が 1 万，辺数が 10 万になっても符号化後の節および制約の数は高々1 億節であった．もちろん提案方法はこれらベースの節に加えて繰返し毎に追加されるブロック節の数を考慮する必要がある．今回のベンチマークの中で提案方法 2 で組込み基数制約を用いた場合に追加された節数の最大値は 15208 節, 中央値は 26 節, 繰返し回数の最大値は 761 回, 中央値は 7 回であった．追加節数の最大値を考慮しても提案方法では符号化後の節数が大幅に削減できることが分かる．この統計値は提案方法の中でも組合せによって多少変化するが桁数は同じである．

4.3 解いた問題数と計算時間

表 2 は各方法が解いた問題数, 図 4 は各方法が解いた問題数と計算時間の関係をカクタスプロットで表したものである．以下の比較を考察する．

4.3.1 全符号化法と提案方法の比較

提案方法は全符号化法 [Velev 09] よりも 2 倍以上多くの問題を解くことに成功した．表 1 に示すように, 全符号化法の符号化後の節数は提案方法より明らかに多いことが原因と考えられる．これは SAT ソルバーにおける単位伝播の効率に影響することに加えて符号化にかかる時間も無視できないことが理由である．実際, random ではグラフの頂点数が 2000 を超えると全符号化法では符号化が制限時間内に完了していない．

4.3.2 疎な SAT 型システムと密な SAT 型システムの比較

これは提案方法 1 と提案方法 2 の同じ基数制約符号化を用いた組合せの比較になる．両方とも概ね良い結果であったが, C++で実装された最新の SAT ソルバーと Java による実装の Sat4j における速度差にも関わらず, 提案方法 2 (密な SAT 型システム) が解いた問題数においてより良い性能を示した (表 2)．1 つの理由は Scarab から外部の SAT ソルバーを起動するための外部プロセス起動時間と SAT ソルバーが問題を読み込むのにかかる時間である．Sat4j を使う限りにおいては, システム全体が Java 仮想マシン上で動作するので Scarab と Sat4j

表 2: 解いた問題数の比較

ベンチマーク		LKH	[Velev 09]	提案方法 1						提案方法 2		
				Glu-S	Glu-T	Lin-S	Lin-T	Min-S	Min-T	S4J-組	S4J-S	S4J-T
color04	(119)	61	59	88	87	43	47	88	86	**92**	90	88
random	(320)	295	145	308	304	180	177	306	303	**313**	308	299
complete	(15)	0	4	2	1	0	0	2	2	**14**	3	4
knight	(11)	**11**	2	7	6	3	3	6	7	5	5	6
tsplib	(9)	9	1	9	9	2	3	9	9	9	9	8
合計	(474)	376	211	414	407	228	230	411	407	**433**	415	405

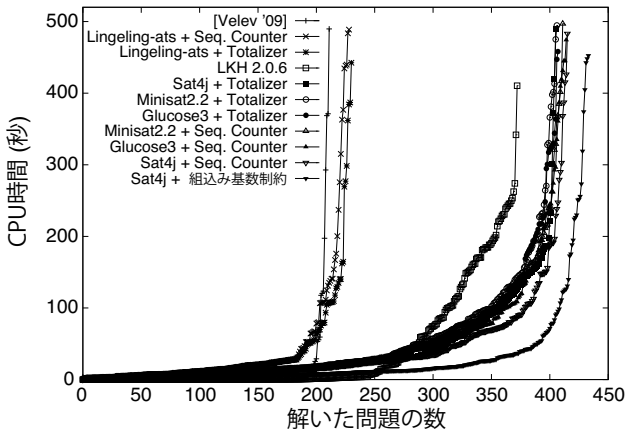

図 4: 解けた問題数と計算時間の関係

の間の通信はメモリ上で完結し外部プロセスを起動する必要がない．他の SAT ソルバーを使う場合は Scarab との通信にファイルの読み書きが発生し, Sat4j にはない負荷が発生する．4.2 節の最後に記述したように繰返し回数の中央値は 10 以下であるので, 多くの問題においてこの負荷は無視できるほど小さい．しかしながら, 500 秒という制限時間を考慮すると繰返し回数が数 100 を超えるような小さい割合の問題においてこれは律速要因となる．

4.3.3 基数制約の SAT 符号化と組込み基数制約の比較

提案方法 2・組込み基数制約が最も高速に最も多い数の問題を解いた．理由として基数制約を符号化する時間が挙げられる．実際, 表 1 において組込み基数制約を用いない方法はいくつかの問題で制限時間内での符号化に失敗している．この比較からも組込み基数制約を利用可能である密な SAT 型システムは有効であるといえる．

4.3.4 TSP ソルバーと提案方法の比較

提案方法 2 はどの組合せにおいても現在最も高性能と考えられる TSP ソルバー LKH よりも多くの問題を解くことに成功した．しかしながら, LKH は knight においては提案方法よりも良い性能を示しており, 問題によって得手不得手があると考えられる．さらに文献を用いた比較も行った．[Eshragh 11] は HCP に対するハイブリッド型アルゴリズムと MIP モデルを提案している．彼らは 8x8, 12x12, 20x20 の騎士の巡回問題と TSPLIB の問題 alb1000 と alb2000 の計算結果を論文で示している．彼らの方法がそれらの問題を解くのに 2 秒から 165600 秒かかっているのに対して, 組込み基数制約と Sat4j を用いた提案方法 2 はそれぞれ 10 秒以内に解いており, これらの問題についてはより良い方法だといえる．

5. まとめ

本稿では, SAT ソルバー Sat4j と密に結合された SAT 型制約プログラミングシステム Scarab およびそれを用いた HCP に対する抽象精密化法を提案した．提案方法は他の SAT 型手法と比較して, 基数制約の符号化時間, SAT ソルバーの起動時間, ファイルの読み書き時間, 符号化後の節数を削減しており, TSP ソルバーと HCP の文献に記載された結果と比較しても求解数と計算時間において良い結果を得た．今後の課題は, どのようなブロック節を加えた時に繰返し回数が少なくなるかを解析し, 提案方法の改良を行うことである．

参考文献

[Clarke 00] Clarke, E. M., Grumberg, O., Jha, S., Lu, Y., and Veith, H.: Counterexample-Guided Abstraction Refinement, in *CAV*, pp. 154–169 (2000)

[Eén 03] Eén, N. and Sörensson, N.: An Extensible SAT-solver, in *Proceedings of the 6th International Conference on Theory and Applications of Satisfiability Testing (SAT 2003), LNCS 2919*, pp. 502–518 (2003)

[Eshragh 11] Eshragh, A., Filar, J. A., and Haythorpe, M.: A hybrid simulation-optimization algorithm for the Hamiltonian cycle problem, *Annals OR*, Vol. 189, No. 1, pp. 103–125 (2011)

[Le Berre 10] Le Berre, D. and Parrain, A.: The Sat4j library, release 2.2, *Journal on Satisfiability, Boolean Modeling and Computation*, Vol. 7, pp. 59–64 (2010)

[Metodi 12] Metodi, A. and Codish, M.: Compiling finite domain constraints to SAT with BEE, *TPLP*, Vol. 12, No. 4-5, pp. 465–483 (2012)

[Prestwich 03] Prestwich, S. D.: SAT problems with chains of dependent variables, *Discrete Applied Mathematics*, Vol. 130, No. 2, pp. 329–350 (2003)

[Tamura 09] Tamura, N., Taga, A., Kitagawa, S., and Banbara, M.: Compiling Finite Linear CSP into SAT, *Constraints*, Vol. 14, No. 2, pp. 254–272 (2009)

[Velev 09] Velev, M. N. and Gao, P.: Efficient SAT Techniques for Relative Encoding of Permutations with Constraints, in *Australasian Conference on Artificial Intelligence*, pp. 517–527 (2009)