Vol.2011-CE-108 No.14
2011/2/6

教育用制約プログラミング環境 G-Cope の実験的評価

櫻井友香[†1]　西田誠幸[†2]

制約条件を宣言的に記述して，制約を満たす答えを見つけ出すことを目的とした制約プログラミングがある．我々はこれまでに制約プログラミングの学習環境として G-Cope を提案し，評価実験を行った．前回の評価実験では，G-Cope の評価の指標として実験経過時間を計測した．しかし，計測した時間には G-Cope の使い方を習得する時間も含まれていたため，G-Cope を正当に評価できなかった．そこで，再度評価実験を行った．本稿では新しい評価実験とその結果について述べる．新しい評価実験の結果，有意水準 0.05 としたとき，G-Cope は制約プログラムの作成に要する時間を短縮できることわかった．

An Experimental Evaluation for An Educational Constraint Programming Environment: G-Cope

Yuka Sakurai[†1] and Seikoh Nishita[†2]

Constraint programming is a programming paradigm whose purpose is to declaratively describe in the form of constraints and find values for given variables satisfying given constraints. We have proposed an educational environment for constraint programming named G-Cope and made an experiment. In the last experiment, we measured time that takes to write a constraint program as metrics. However, it was not enough to evaluate G-Cope, because the time includes time to learn how to use G-Cope. Therefore, we made a new experiment. This paper proposes the new experiment and its result. The experimental result of this experiment shows G-Cope foreshorten time for writing the constraint program with significance level 0.05.

1. はじめに

プログラミングのための言語や手法は数多く存在する．その 1 つに制約条件を満たす答えを探して見つけ出す問題を解くことを目的とした制約プログラミングがある．制約プログラミングは，要因配置計画をはじめ，時刻表，時間割などのタイムテーブル作成，生産計画，工程計画，資源割当て計画など様々な分野で問題解決を図る技術として近年脚光を浴びている．

我々はこれまでに制約プログラミングの学習環境として G-Cope[4] を提案し，評価実験を行った．前回の評価実験では，制約プログラミングの未経験者に G-Cope とテキストエディタの 2 種類のツールでそれぞれ制約プログラムを作成させ，作成時間を計測した．この作成時間を学習効果の指標とし，G-Cope の制約プログラミングへの効果を測った．しかし，計測結果の時間には，制約プログラミングの時間とツールの使い方を習得する時間の 2 種類が含まれていた．さらに，ツールを使用する順番が計測時間に影響を及ぼしている可能性があった．また，評価基準を設けていなかったため，G-Cope を正当に評価できなかった．そこで，前回の評価実験での問題点を踏まえて，再度評価実験を行った．

本稿では，新しい評価実験とその結果について述べる．新しい評価実験では，前回と同様に実験経過時間を計測する．しかし，前回の評価実験とは違って，ツールの使い方を習得してからどちらか一方のツールによる実験経過時間を計測する．また，評価基準として統計を用いて実験を行い，G-Cope を評価する．

以下 2 節では制約プログラミングについて述べる．また，3 節では我々が提案した教育用制約プログラミング環境について述べる．4 節，5 節では前回実施した評価実験と，新たな評価実験について述べる．そして 7 節では関連研究について述べる．

2. 制約プログラミングとは

制約プログラミングとは，制約条件を宣言的に記述し，制約条件を満たす答えを見つけ出すこと目的としたプログラミングパラダイムである．制約プログラムは制約による問題の記

†1 拓殖大学大学院工学研究科電子情報工学専攻
Electronics and Information Science Course, Graduate School of Engineering, Takushoku University
†2 拓殖大学工学部情報工学科
Department of Computer Science, Faculty of Engineering, Takushoku University

Vol.2011-CE-108 No.14
2011/2/6

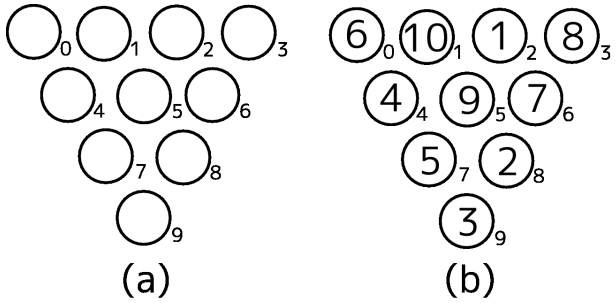

図 1 ぶどうの房パズルと解の一例
Fig. 1 The instance of the bunch of grape puzzle and one of its solution.

述と，その問題を解く方法によって構成される．

制約プログラムをぶどうの房パズル[2] を用いて説明する．n マスのぶどうの房パズルは次の 3 つの制約を満たすように各セルに数値を設定するパズルである．

(1) 数値の範囲は 1 から n までである
(2) 全てのセルの数値は異なる
(3) 隣り合う 2 マスの差が隣接する下段のマスと等しい

図 1 に 10 マスのぶどうの房パズルの一例を，**図 2** にぶどうの房パズルを解くプログラムをそれぞれ示す．図 1 の図形の数字は図 2 のプログラムの配列の添数を表す．

ここで，図 2 のプログラムについて説明する．右側のコメントについては，次章で説明する．

まず，変数宣言をする．Network はドメイン変数と制約を管理するオブジェクトのクラスである．ドメイン変数は，値が取り得る範囲を情報として持つ IntVariable 型の変数である．ぶどうの房パズルの 10 マスに対応する要素数 10 のドメイン変数を宣言する (4 行目)．また，制約 (1) より，ドメイン変数の範囲を 1 から 10 と設定する (6–8 行目)．次に，制約宣言をする．制約生成関数 sub を呼び出し，制約 (3) を設定する (9–14 行目)．また，制約 (2) より，配列全ての変数が異なるという制約を設定する (15 行目)．そして，解を管理する Solver によって，解を探索し，見つけた解を図 1 の形式で表示する (18–25 行目)．制約生成関数 sub は引数であるドメイン変数 v0, v1, v2 と Network に対して，v0 と v1 の差が v2 に等しいという制約を生成する関数である (29–34 行目)．

```
1: public class Grape {
2:   public static void main(String[] args) {
3:     Network net = new Network();
4:     IntVariable var[] = new IntVariable[10];
5:
6:     for (int i = 0; i  < 10; i++) {
7:       var[i] = new IntVariable(net, 1, 10);
8:     }
9:     sub(net, var[0], var[1], var[4]);
10:    sub(net, var[1], var[2], var[5]);
11:    sub(net, var[2], var[3], var[6]);
12:    sub(net, var[4], var[5], var[7]);
13:    sub(net, var[5], var[6], var[8]);
14:    sub(net, var[7], var[8], var[9]);
15:    new NotEquals(net, var);
16:
17:    Solver solver = new DefaultSolver(net);
18:    for (solver.start(); solver.waitNext(); solver.resume()) {
19:      Solution solution = solver.getSolution();
20:      System.out.println(
               solution.getIntValue(var[0]) + " " + solution.getIntValue(var[1])
               + " " + solution.getIntValue(var[2]) + " " + solution.getIntValue(var[3]));
21:      System.out.println(
               " " + solution.getIntValue(var[4]) + " " + solution.getIntValue(var[5])
               + " " + solution.getIntValue(var[6]));
22:      System.out.println(
               "  " + solution.getIntValue(var[7]) + " " + solution.getIntValue(var[8]));
23:      System.out.println("   " + solution.getIntValue(var[9]));
24:      System.out.println();
25:    }
26:    solver.stop();
27:  }
28:
29:  public static void sub(Network net, IntVariable v0, IntVariable v1, IntVariable v2) {
30:    IntVariable max = v0.max(v1);
31:    IntVariable min = v0.min(v1);
32:    IntVariable v = max.subtract(min);
33:    new Equals(net, v, v2);
34:  }
35: }
```

変数宣言
制約宣言
解の導出
制約生成関数の作成

図 2 制約プログラムの一例
Fig. 2 A sample of the constraint program for the bunch of grape puzzle.

3. 提案した教育用制約プログラミング環境

本章では，我々が提案した教育用制約プログラミング環境 G-Cope について述べる．

3.1 開 発 方 針

制約プログラムは大きく 4 つのステップに分けられる．

(1) 変数宣言：ドメイン変数の宣言，変数の設定
(2) 制約生成関数の作成：問題の制約を表す
(3) 制約宣言：制約生成関数を呼び出す
(4) 解の導出：解の探索と表示

4 つのステップについて図 2 を用いて説明する．(1) と (4) は多くの問題で同様の形式となる．(2) と (3) は与えられた問題を解くのに適したプログラムを記述する必要がある．

また，制約プログラミングで最も大切なことは，与えられた問題から制約を見つけ出すことであり，制約プログラムを解くために必要な制約を考えることである．つまり，(2) の制約生成関数を記述することが制約プログラミングにおいて最も重要である．

制約プログラムについての以上の検討をもとに，我々はプログラムエディタとドローツールを組み合わせた教育用制約プログラミング環境 G-Cope を提案した．G-Cope の 2 つのツールは，次の方針に従って，上記の各ステップの入力を支援する．

- プログラムエディタは (2) 制約生成関数の入力を促す．
- ドローツールは上記の (1)，(3) の入力を支援する．
- (4) のプログラムは G-Cope が自動生成する．

G-Cope に期待する効果を以下に示す．

- 制約プログラムの構造の学習を支援する．
  G-Cope の構成は制約プログラムの 4 つのステップに基づいており，G-Cope による制約プログラミングによって，プログラムの構成の学習を助けることが期待される．
- プログラム作成時間を短縮する．
  限られた時間内で制約プログラミングを行う際に，与えられた問題から制約を見つけ出しそれをプログラムとして表現する時間を十分確保することを狙う．

3.2 教育的制約プログラミング環境の概要

G-Cope の概要を**図 3** に示す．

GUI エディタは制約プログラム上の変数宣言と制約宣言に対応している．GUI エディタは自由に図形を描画する機能を持ち，GUI エディタ上で図形を 1 つ作成することは，制約プログラムでドメイン変数を 1 つ宣言することを意味する．また，図形の属性を変えることで，ドメイン変数の範囲を設定する．さらに，図形と制約生成関数を選択することによって，制約宣言をする．

プログラムエディタは制約プログラム上の制約生成関数の作成に対応している．プログラムエディタは，引数にドメイン変数と Network を持つ制約生成関数を書くためのエディタである．

G-Cope は GUI エディタで作成されたプログラムと，プログラムエディタで作成されたプログラムを 1 つにまとめて自動生成する．ここで，生成されたプログラムは読み書きすることが可能である．また，自動生成されたプログラムを Cream library[6] と共にコンパイルすることで，ソルバソフトウェアが作成される．

ソルバソフトウェアは制約プログラム上の解の導出と結果の表示に対応している．ソルバソフトウェア上の図形は，GUI エディタで配置した図形の位置と大きさにしたがって描画される．

4. 前回の評価実験

G-Cope の効果を計るために実験を行った．前回の評価実験では，2 種類の制約プログラムを作成するツールを用いて，被験者にプログラミングをしてもらい，プログラミングに要する時間を計測した．この実験により得た時間の差によって G-Cope を評価した．

4.1 実験内容と結果

実験では，制約プログラムを作成するツールとして，テキストエディタと G-Cope を用いた．被験者各人には 2 つのツールを順番に使用させ，各ツールによる制約プログラミングに要した時間を計測した．また，2 つのツールを使用する順番によって，6 人の被験者を 2 つのグループに分けて実験を実施した．

結果として，ほとんどの被験者が最初に使用したツールによる制約プログラミングに長い時間を要した．また，2 度目は G-Cope を使用した被験者のほとんどが前回に比べて時間が短かった．

4.2 問 題 点

しかし，この実験方法では 3 つの問題点があった．

(1) ツールの使い方を習得する時間:
この実験では，ツールの使い方を習得する時間が計測時間に影響を及ぼしている可能性があった．つまり，計測結果の時間が制約プログラミングの時間とツールの使い方

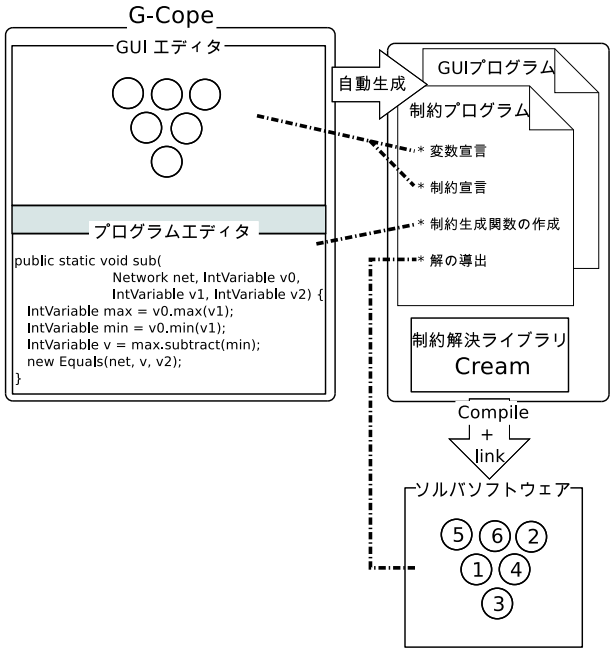

図 3 G-Cope の概要
Fig. 3 The structure of G-Cope

を習得する時間の 2 種類あった．そのため，ツールの使い方を習得する時間と，プログラムを作成する時間を分割する必要がある．

(2) ツールを使用する順番:
テキストエディタではプログラム全体の記述に対し，G-Cope ではプログラムの一部しか記述をしない．そのため，G-Cope，テキストエディタの順番でプログラムを作成した被験者たちは，プログラムの全体を想像しながらプログラムの一部 (制約生成関数) を記述することを強いられたために，プログラミングに長い時間を要したと考えられる．つまり，プログラムの全体を把握してからプログラムの一部を記述する被験者たちと，プログラムの全体を想像しながらプログラムの一部を記述する被験者たちとではプログラムに対する情報量が公平ではなく，差がでてしまった．そのため，使用するツールの順番が計測時間に影響を及ぼしている可能性があった．

(3) 評価基準:
この実験では，評価基準を設けていなかったため，それぞれのグループの作成時間の差を比較しただけで，G-Cope を正当に評価したとは言えなかった．

5. 新たな評価実験

前回の評価実験でも問題点を踏まえて，再度制約プログラミングに要する時間を計測した．また，本評価実験では前回の評価実験での問題点を解決するため，以下の方法をとる．

(1) ツールの使用方法の習得と制約プログラミングの分割
ツールの使用方法を習得する時間の計測時間への影響を避けるため，プログラミングに要する時間を計測する前に，ツールの使用方法を習得する時間を設ける．

(2) 使用ツール
プログラムに対する情報量を公平にするため，制約プログラミングの未経験者を 2 グループに分け，G-Cope かテキストエディタのどちらか一方のツールを用いて制約プログラムを作成させることとする．

(3) 評価基準
評価基準として，統計学を用いて検定をする．検定方法として，計測した結果から 2 グループの間に差があるかを方法とした，平均の差の検定を用いる．

5.1 実験内容

本評価実験では，被験者を 2 グループに分け，G-Cope で制約プログラムを作成するグループを α グループ，テキストエディタで制約プログラムを作成するグループを β グループとした．両グループはツールの体験，制約プログラムの作成の順番で実験を実施した．そして，被験者に 3×3 の魔方陣パズル[3] とグラフの 3 彩色問題を出題し，プログラムを作成させた．3×3 の魔方陣パズルとは縦と横，斜めの 3 つの数の和がそれぞれ 15 になるように 1 から 9 の数字を配置するパズルである．**図 4** に魔方陣パズルを示す．グラフの 3 彩色問題は隣接する頂点同士が同じ色にならないように全頂点に彩色する問題である．**図 5** にグラフの 3 彩色問題を示す．

ここで，α グループと β グループが行う実験の手順を示す．

- α グループ
まず，G-Cope の使用方法が記された資料を元に，魔方陣パズルを解くためのパズルを作成する．この資料には，魔方陣パズルを解くプログラムを作成する手順として，パズルのマスの配置方法や制約の意味，制約設定の方法，プログラムの生成方法などが細か

Vol.2011-CE-108 No.14
2011/2/6

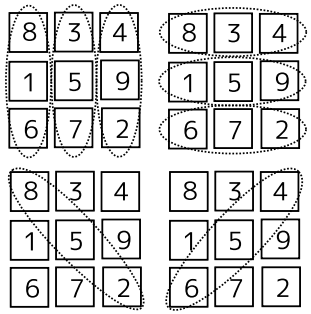

図 4 魔方陣パズルの性質
Fig. 4 Property of a magic square puzzle
丸で囲まれた部分の和は全て “15” となる

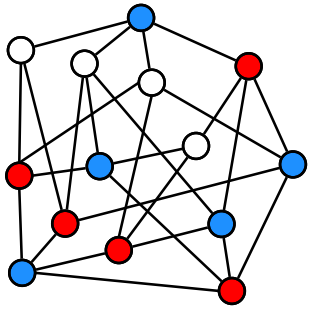

図 5 実験で用いたグラフの 3 彩色問題
Fig. 5 The graph 3-coloring problem on the experiment

く記されている．この時，被験者は魔方陣パズルの制約生成関数は読み書きしない．魔方陣パズルを解くための制約生成関数をあらかじめ作成しておき，被験者はこの制約生成関数を用いて制約設定をする．ソルバソフトウェアの生成後，制約プログラムをコンパイル，実行し，結果を確認する．

次に，グラフの 3 彩色問題を解くためのプログラムを作成する．この時，制約生成関数を含め，G-Cope における制約プログラムを作成する全てのステップを行う．結果を確認し，解があっていたら終了とする．

- β グループ

表 1 制約作成時間
Table 1 Time that takes to write a constraint program

グループ	α	グループ	β
ツール / 被験者	テキストエディタ (H:M:S)	ツール / 被験者	G-Cope (H:M:S)
A	2:23:20	I	1:10:35
B	3:29:45	J	0:39:00
C	4:48:13	K	0:32:48
D	1:11:27	L	1:15:36
E	1:26:21	M	0:59:25
F	7:15:25	N	1:16:54
G	1:14:24	O	0:40:25
H	5:20:13	P	2:30:47
		Q	1:32:56
被験者人数 (人)	8	被験者人数 (人)	9
平均時間 (秒)	11493.57	平均時間 (秒)	4796.22
標本標準偏差 (秒)	8410.61	標本標準偏差 (秒)	2169.08

テキストエディタで制約プログラムを作成する際に，Java 用の制約プログラミングライブラリ Cream[6] を用いる．そこで，まず，魔方陣プログラムの作成方法が記された資料を元に，テキストエディタで魔方陣プログラムを作成する．この資料には，魔方陣プログラムを解くプログラムリストとプログラムの意味が記されている．プログラム作成後，プログラムをコンパイル，実行し，結果を確認する．

次に，グラフの 3 彩色問題を解くためのプログラムを作成する．結果を確認し，解があっていたら終了とする．

両グループに上記の手順で実験を実施し，グラフの 3 彩色問題のプログラミング時間を計測した．

表 1 に実験結果を示す．表 1 より，β グループの方が α グループよりも平均時間が短いことがわかる．α グループと β グループのプログラミングの時間に有意な差があるかどうか，平均の差の検定を用いて検定した．本評価実験での検定仮説と対立仮説を以下のように立てる．

Vol.2011-CE-108 No.14
2011/2/6

検定仮説 H_0: $X_\alpha = X_\beta (X_\alpha - X_\beta = 0)$
対立仮説 H_1: $X_\alpha > X_\beta$
X_α: テキストエディタでのプログラム作成時間の平均
X_β: G-Cope でのプログラム作成時間の平均

ここで，有意水準 α を 0.05 とすると，検定仮説が棄却され対立仮説が真となった．つまり，α グループと β グループのプログラム作成時間に有意な差があり，G-Cope はテキストエディタでプログラムを作成するよりも，短い時間でプログラムを作成できることが証明された．

また，以下に被験者たちの感想を述べる．

α **グループ** (G-Cope)
- 図形の選択ミスをした際に，プログラムを生成してコンパイルをしないとエラーがわからないため，手間がかかる．
- 図形が多いと，どの図形に対して制約設定をしたかわからなくなる．
- 絵を見ながら，プログラムを作成できるので結果が想像できる．

β **グループ** (テキストエディタ)
- 結果の想像ができなくて困惑した．
- プログラムを書くことに抵抗がある．
- 制約を考えるのが大変だった．

5.2 考　　察

実験の結果から，G-Cope は有意水準 0.05 としたとき，テキストエディタで作成するよりも短い時間でプログラムを作成できることがわかった．この原因として，ドローツールを用いてプログラミングをする影響が大きかったと考えられる．G-Cope では自分で配置した図を見ながら制約を作成することに対し，テキストエディタでは変数と図を照らし合わせながら制約を作成する必要があるために，プログラミング時間に差がでてしまったと考えられる．

また，G-Cope はプログラムの正誤判定をするために一度プログラムを生成し，修正する度に端末でコンパイルする必要があるため手間がかかる．そこで，制約プログラミングの学習をさらに支援するためには，プログラムの正誤判定をする必要があると考えられる．

本評価実験では，G-Cope に期待する効果のもう一方である，制約プログラムの構造の学習に関しては今回の実験では効果を測れなかった．そのため，学習効果を測るためには，G-Cope で制約プログラムを作成後，簡単な問題を出題し，制約プログラムに対する理解度を測る必要があると考えられる．

6. 関 連 研 究

関連研究として，既存の制約プログラミング環境を 4 つ述べる．

Gianna[4] は CSP(Constraint satisfaction problem) をモデル化するための化した環境である．Gianna は制約グラフと呼ばれるグラフで G-Cope と同様に視覚的に変数間の関係を表現する．ノードが変数を表し，枝が制約となる．オブジェクト同士の組み合わせを図形で定義し，枝に論理式で制約を記述するツールである．

Oz Explorer[5] は探索木を表示することによって，G-Cope と同様に視覚的に制約プログラミングをサポートするツールである．しかし，G-Cope とは異なりプログラムを作成するプロセスを視覚化するのではなく，制約プログラムを記述し，全ての探索ルートを探索木として表示するツールである．

ThingLab[7] は，制約によるユーザインタフェースの構築についての体験的学習の支援を目的としたツールである．G-Cope とは異なり，線や図形に対して制約プログラムを記述することにより，図形が制約通りに描画される．

Skeleton[8] は G-Cope と同様に，視覚的に制約プログラミングを学習するソフトである．しかし，入力インタフェースは異っており，制約の入力にはスプレッドシートを用いる．スプレッドシートは Excel のワークシートに似たようとセルがあり，セルに制約を数式や関数として記述する．セルに制約を記述することで，セルと図形オブジェクトの座標の関連付けや，図形を数式にしたがって動かすことが可能である．

7. お わ り に

本稿では，教育用制約プログラミング環境 G-Cope とその評価実験について述べた．G-Cope はパズルを題材とした問題を解くためのソフトウェアの作成を通じて，制約プログラミングについての学習支援を目的としたツールである．前回の評価実験では，G-Cope を正当に評価できなかったため，再度評価実験を行った．その結果，有意水準 0.05 としたとき，G-Cope はテキストエディタでプログラムを作成するよりも，短い時間でプログラムを作成できることが証明された．

Vol.2011-CE-108 No.14
2011/2/6

参 考 文 献

1) 櫻井 友香，西田 誠幸.: “GUI ビルダツールを用いた教育用制約プログラミング環境の開発”，コンピュータと教育研究会，研究報告，Vol.2010-CE-103 No.9,2010.
2) ぶどうの房パズル：http://www2.oninet.ne.jp/mazra/math104.htm
3) 魔方陣パズル：http://www.itmn.biz/
4) M.Paltrinieri.: *“A Visual Environment for Constraint Programming”*, 11th International IEEE Symposium on Visual Language, 1995.
5) Christian Schulte.: *“Oz Explorer: Visual Constraint Programming Tool”*, In LeeNaish, editor, Proceedings of the Fourteenth International Conference on Logic Programming, pp286–300, 1997.
6) Naoyuki Tamura.: *Cream:Class Library for Constraint Programming in Java* (2003). http://bach.istc.kobe-u.ac.jp/cream/
7) ALAN Borning.: *“The Programming Language Aspects of ThingLab, A Constraint-Oriented Simulation Laboratory”*(1981).
8) Takashi Yamamiya.: *“Skeleton, easy simulation system”*(2004).