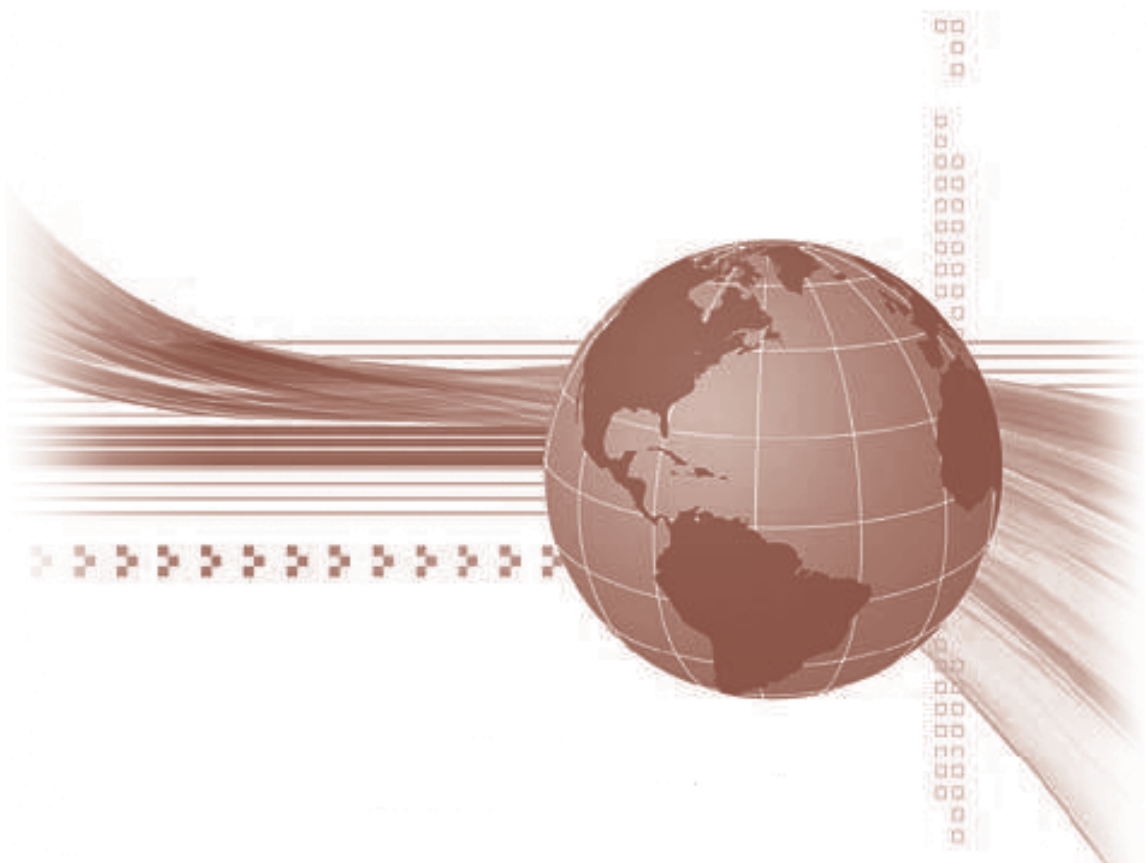
STUDIA UNIVERSITATIS
BABEŞ-BOLYAI



INFORMATICA

1/2011

Anul LVI **2011**

S T U D I A
UNIVERSITATIS BABEŞ-BOLYAI

INFORMATICA

1

Redacţia: M. Kogălniceanu 1 • 400084 Cluj-Napoca • Tel: 0264.405300

SUMAR – CONTENTS – SOMMAIRE

STUDIA UNIV. BABEŞ–BOLYAI, INFORMATICA, Volume **LVI**, Number 1, 2011

MODIFIED STRONG AND COALITION PROOF NASH EQUILIBRIA. AN EVOLUTIONARY APPROACH

NOÉMI GASKÓ, D. DUMITRESCU, RODICA IOANA LUNG

Abstract. In non-cooperative games one of the most important solution concept is the Nash equilibrium based on the idea of stability against unilateral deviations. In games having more Nash equilibria a selection problem can appear. The modified strong Nash equilibrium and the coalition proof Nash equilibrium are important refinements of the Nash equilibrium that can solve the selection problem. A generative relation for the modified strong Nash and for the coalition proof Nash equilibrium based on nondomination is proposed. Some examples illustrate the effectiveness of the proposed method.

1. Introduction

Game Theory represents a basis for neo-classical microeconomic theory and it is an important research field [11].

A finite strategic game is defined a system $G = ((N, S_i, u_i), i = 1, n)$, where:

- N represents a set of players, and n is the number of players;
- for each player $i \in N$, S_i is the set of available actions,

$$S = S_1 \times S_2 \times ... \times S_n$$

  is the set of all possible situations of the game and $s \in S$ is a strategy (or strategy profile) of the game;
- for each player $i \in N$, $u_i : S \to R$ represents the payoff function (utility) of the player i.

The Nash equilibrium [9] is one of the most important solving concepts in non-cooperative game theory. Playing in Nash sense means that no player has a better chance to improve her payoff while others keep theirs unchanged.

Received by the editors: December 12, 2010.

2010 *Mathematics Subject Classification.* 91A10.

1998 *CR Categories and Descriptors.* I.2.8 [**Heuristic methods**]: *Artificial intelligence*; *Key words and phrases.* evolutionary detection, games, equilibrium.

Definition 1. *A strategy profile $s^* \in S$ is a Nash equilibrium if the inequality holds:*

$$u_i(s_{ij}, s^*_{-i}) \leq u_i(s^*), \forall i = 1, n, \forall s_{ij} \in S_i,$$

where (s_{ij}, s^*_{-i}) denotes the strategy profile obtained from s^* by replacing the strategy of player i with s_{ij}.

The problem of detecting the Nash equilibrium is an important computational task. In [8] Nash equilibrium is characterized by a generative relation.

A selection problem can appear in games having more Nash equilibria. Several refinements have been introduced to solve this selection problem. One of this is the Aumann equilibrium [1].

This paper is concerned with on two refinements of the Nash equilibrium: the modified strong Nash equilibrium and the coalition proof Nash equilibrium.

2. Nash equilibrium refinements

Two important Nash equilibrium refinements are presented in this section: the modified strong Nash equilibrium and the coalition proof Nash equilibrium.

2.1. Modified strong Nash equilibrium.

The modified strong Nash equilibrium is introduced by Ray [12] and Greenberg [6].

Let us consider a finite strategic game $G = ((N, S_i, u_i), i = 1, n)$, and the following notations: $S_I = \prod_{i \in I} S_i$ and $x_I = (x_i)_{i \in I}$.

The following definitions are necessary to introduce the modified strong Nash equilibrium:

Definition 2. *For $I \in 2^N - \{\emptyset\}$, $x \in S_N, y_I \in S_I$ we say that y_I is blocked by $T \subset I$ given x if there exists a vector $z_T \in S_T$ such that:*

$$u_T(z_T, y_{I-T}, x_{N-T}) \geq u_T(y_T, x_{N-T}).$$

Definition 3. *I is credible given x if there is a $y_I \in S_I, y_I \neq x_I$, that is not blocked by any credible $T \subset I$ given x.*

Definition 4. *A strategy profile $x \in S_N$ is a modified strong Nash equilibrium if it is not blocked by any credible coalition (given x).*

Example 1. *Game G_1. Let us consider the two person game, for that the payoffs are represented in Table 1. The game has a pure Nash equilibria (B, B), but this not an Aumann equilibrium. However this is a modified strong equilibrium, because it can be blocked by any credible coalition.*

TABLE 1. The payoff functions of the two players for game G_1

		Player2	
		A	*B*
Player1	*A*	*(5,5)*	*(3,6)*
	B	*(6,3)*	*(4,4)*

2.2. **Coalition proof Nash equilibrium.** Bernheim [2] introduced the coalition proof Nash equilibrium. A coalition-proof equilibrium is a correlated strategy from which no coalition has an improving and self-enforcing deviation.

Definition 5. *Let $s^* \in S$ and let P the set of the subsets of N. An internally consistent improvement of P upon s^* is defined by induction on $card(P)$ [7]:*

- *if $card(P) = 1$, then $P = \{i\}$, then s_i is an ICI upon s^*, if*
$$u_i(s_i, s^*_{N-i}) > u_i(s^*);$$
- *if $card(P) > 1$, then $s^P \in S^P$ is an ICI of P upon s^**
(i) s^P is an improvement of P upon s^*;
and
(ii) if $T \subset P$ and $card(T) < card(S)$ then T has no ICI upon (s^P, s^*_{N-S}).

Definition 6. *A strategy profile $s \in S$ is a coalition proof Nash equilibrium, if no P subcoalition has an ICI upon s^*.*

Let us denote by CNE the coalition proof Nash equilibrium.

Remark 1. *The coalition proof Nash equilibrium is a subset of the Nash equilibrium:*
$$CNE \subseteq NE.$$

3. GENERATIVE RELATIONS

Generative relations for modified strong Nash and coalition proof Nash equilibria are introduced.

3.1. **Generative relation for modified strong Nash equilibrium.** Consider two strategy profiles x and y from S. Denote by $ms(x, y)$ the number of players in coalition $T, T \subset I$, $I \subseteq N$ benefiting from switching between strategies:

$$ms(x, y) = card[t \in T, T \neq \phi, T \subset I, \phi \neq I \subseteq N,$$
$$u_t(z_t, y_{I-T}, x_{N-I}) \geq u_t(y_I, x_{N-I}), y_I \neq x_I, z_t \in S_T],$$

where $card[M]$ denotes the cardinality of the multiset M (an element i can appear several times in M and each occurrence is counted in $card[M]$).

Definition 7. *Let $x, y \in S$. We say the strategy x is better than strategy y with respect to modified strong Nash equilibrium, and we write $x \prec_{MS} y$, if and only if the inequality*

$$ms(x, y) < ms(y, x),$$

holds.

Definition 8. *The strategy profile $y \in S$ is a modified strong Nash non-dominated (NMSN) strategy, if and only if there is no strategy $x \in S, x \neq y$ such that x dominates y, i.e.*

$$x \prec_{MS} y.$$

We consider relation $\prec_{MS}$ as the generative relation of the modified strong Nash equilibrium. The nondominant strategies with respect to the relation $\prec_{MS}$ can be a suitable representation of the modified strong Nash equilibrium.

3.2. **Generative relation for coalition proof Nash equilibrium.** Consider two strategy profiles x and y from S.

We may define the quality $cn(x, y)$ as:

$$cn(s, s^*) = card[i \in I, \phi \neq I \subseteq N, u_i(y^I, x^{*-I}) \geq u_i(x^*), y^i \neq x^{*-i}]$$

$$+card[t \in T, T \neq \phi, T \subset I, \phi \neq I \subseteq N, u_t(z_t, y_{I-T}, x_{N-I}) \geq u_t(y_I, x_{N-I}),$$

$$y_I \neq x_I, z_t \in S_T],$$

where $card[M]$ denotes the cardinality of the multiset M.

Definition 9. *Let $x, y \in S$. We say the strategy x is better than strategy y with respect to coalition proof Nash equilibrium, and we write $x \prec_{CN} y$, if and only if the inequality*

$$cn(x, y) < cn(y, x),$$

holds.

Definition 10. *The strategy profile $y \in S$ is a coalition proof Nash non-dominated strategy, if and only if there is no strategy $x \in S, x \neq y$ such that x dominates y with respect to $\prec_{CN}$ i.e.*

$$x \prec_{CN} y.$$

We may consider relation $\prec_{CN}$ as a candidate for generative relation of the coalition proof Nash equilibrium.

4. Evolutionary equilibria detection

Generative relations may be used by evolutionary techniques for equilibria detection.

A population of game strategies is evolved. Every individual is encoded as a n-dimensional vector representing a strategy $s \in S$.

An initial strategy population is randomly generated. Population at iteration t may be viewed as the set of current equilibrium approximation.

Simulated binary crossover (SBX) [5] and real polynomial mutation [4] operators are used. The generative relation is used for rank-based fitness assignment.

The evolutionary technique is called the Relational Evolutionary Equilibria Detection (REED), which can be described as follows:

REED method

Step1. Set $t = 0$;
Step2. Randomly initialize a population $P(0)$ of strategies;
Step3. Binary tournament selection and recombination using the simulated binary crossover (SBX) operator for $P(t) \to Q$;
Step4. Mutation on Q using real polynomial mutation $\to P$;
Step5. Compute the rank of each population member in $P(t) \cup P$ with respect to the generative relation. Order by rank $(P(t) \cup P)$;
Step6. Rank based selection for survival $\to$ P (t + 1);
Step7. Repeat steps Step3 - Step6 until the maximum generation number is reached.

5. Numerical experiments

The generative relations are used for the rank based fitness assignment. The population size is 300 and the number of generation is 150. The used parameter setting is described in [4].

The experiments have been conducted for ten runs with different random seed generators.

In order to illustrate the proposed technique some discrete and continuous games are presented.

5.1. **Experiment 1.** Let us consider the following three person game [3], denoted by G_2, where the payoffs are represented in Table 5.1. The first player has three strategies, and her payoff is the first value from the triplet. The second player has three strategies, as well, her payoff is the second value from the triplet. The third player has two strategies her payoff is the third value.

TABLE 2. The payoff values of the three players in the game G_2

Player3 A

		Player2 A	Player2 B	Player2 C
Player1	A	(-2,-2,-10)	(-10,-10,-10)	(-10,-10,-10)
	B	(-10,-10,-10)	(1,1,-5)	(-10,2,-10)
	C	(-10,-10,-10)	(2,-10,-10)	(0,0,10)

Player3 B

		Player2 A	Player2 B	Player2 C
Player1	A	(-1,-1,5)	(-5,-5,0)	(-10,-10,-10)
	B	(-5,-5, 0)	(-2,-2,-10)	(-10,-10,-10)
	C	(-10,-10,-10)	(-10,-10,-10)	(-15,-15,-15)

The game has two pure Nash equilibria (C, C, A), the first player plays C, the second C, and the third plays A. The other Nash equilibrium is (A, A, B). The game has only one modified strong Nash equilibrium (A, A, B), and only one coalition proof Nash equilibrium: (C, C, A).

The algorithm detected correctly all these different types of equilibria.

5.2. **Experiment 2.** Let us consider the game G_3 [10], having the following payoff functions:

$$u_1(x_1, x_2) = -x_1^2 - x_1 + x_2,$$
$$u_2(x_1, x_2) = 2x_1^2 + 3x_1 - x_2^2 - 3x_2, x_1, x_2 \in [0, 1].$$

The corresponding payoffs for the Nash equilibrium, the modified strong Nash equilibrium and the Pareto front are depicted in Figure 1. The Nash equilibrium and the modified strong Nash equilibrium are the same, (0,0) and the corresponding payoff is (0,0).

5.3. **Experiment 3.** Let us consider the three players game G_4, having the following payoff functions:

$$u_1(x, y, z) = x(10 - sin(x^2 + y^2 + z^2)),$$
$$u_2(x, y, z) = y(10 - sin(x^2 + y^2 + z^2)),$$
$$u_3(x, y, z) = z(10 - sin(x^2 + y^2 + z^2)),$$
$$x, y, z, \in [0, 10].$$

This game has more Nash equilibria, and only one modified strong and coalition proof Nash equilibrium, which is the strategy pair $(10, 10, 10)$, and the corresponding payoff $(110, 110, 110)$. The three equilibria types are depicted on Figure 2.

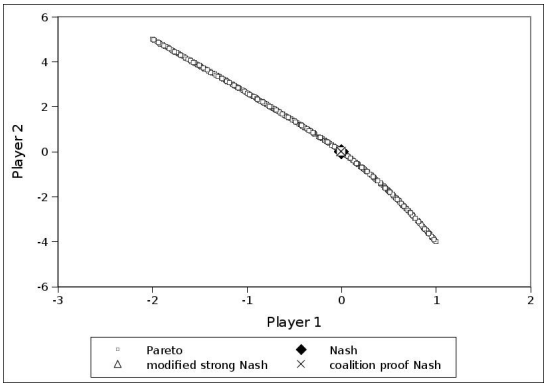


FIGURE 1. Detected payoffs for Pareto front, Nash equilibrium and modified strong Nash equilibrium for the game G_3

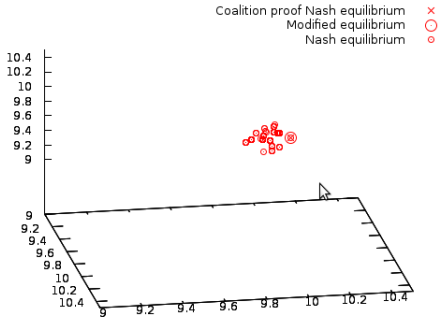


FIGURE 2. Detected strategies for Nash equilibrium, modified strong Nash equilibrium and coalition proof Nash equilibrium for Game G_4

6. CONCLUSIONS

The modified strong Nash and the coalition proof Nash equilibrium are refinements of the well-studied Nash equilibrium. Generative relations for modified strong Nash equilibria and of coalition proof Nash equilibrium are proposed.

An evolutionary approach is presented for detecting the modified strong Nash and the coalition proof Nash equilibria. Some discrete and continuous games are considered for numerical experiments. The experiments illustrate

the effectiveness of the proposed method. A further step can be simulation of games with more players.

7. Acknowledgment

This work was partially supported by CNCSIS-UEFISCSU, project number PNII-IDEI 508/2007. The first author wishes to thank for the financial support provided from programs co-financed by The Sectoral Operational Programme Human Resources Development, Contract POSDRU 6/1.5/S/3 - "Doctoral studies: through science towards society".

References

[1] Aumann, R.: *Acceptable Points in General Cooperative n Person Games* , Contributions to the Theory of Games, Vol IV, Annals of Mathematics Studies, 40. 287-324, 1959.

[2] Bernheim B.D,. Peleg B., Whinston M.D.: *Coalition-proof equilibria. I. Concepts.*, J Econ Theory 42: 112, 1987.

[3] Borm, P., Otten, G-J., Peters, H., Larbani, M.: *Core Implementation in Modified Strong and Coalition Proof Equilibria*, Cahiers du Centre d'Etudes de Recherche Operationnelle, 1992, v.34,187-197.

[4] Deb, K., Agrawal, S., Pratab, A., Meyarivan, T.: *A Fast Elitist Non-Dominated Sorting Genetic Algorithm for Multi-Objective Optimization: NSGA-II*, Marc Schoenauer, Kalyanmoy Deb, Günter Rudolph, Xin Yao, Evelyne Lutton, Juan Julian Merelo, and Hans-Paul Schwefel, editors, Proceedings of the Parallel Problem Solving from Nature VI Conference, Paris, France, 2000.

[5] Deb, K., Beyer, H.: *Self-adaptive genetic algorithms with simulated binary crossover*, Complex Systems, 1995, vol.9, 431-454.

[6] Greenberg, J.: *The core and the solution as abstract stable sets*, mimeo, University of Haiffa.

[7] Keiding, H., Peleg, B.: *Representation of effectivity functions in coalition proof Nash equilibrium: A complete characterization*, Soc Choise Welfare (2002) 19, 241-263.

[8] Lung, R. I., Dumitrescu, D.: *Computing Nash Equilibria by Means of Evolutionary Computation*, Int. J. of Computers, Communications & Control, 2008, 364-368.

[9] Nash, J. F.: *Non-cooperative games*, Annals of Mathematics, 54:286-295, 1951.

[10] Nessiah, R., Tazdait, T.: *Strong Berge and Pareto Equilibrium Existence for a Noncooperative Game*, Working paper.

[11] Osborne, M.: *An Introduction to Game Theory*, Oxford University Press, New York, 2004.

[12] Ray, D.: *Credible coalitions and the core* International Journal of Game Theory 18, 185-187.

Babes-Bolyai University, Cluj-Napoca
E-mail address: gaskonomi@cs.ubbcluj.ro
E-mail address: ddumitr@cs.ubbcluj.ro
E-mail address: rodica.lung@econ.ubbcluj.ro

STUDIA UNIV. BABEŞ–BOLYAI, INFORMATICA, Volume **LVI**, Number 1, 2011

A P SYSTEM DESIGN USING CLONAL SELECTION ALGORITHM

EMAD NABIL[1], AMR BADR[2], AND IBRAHIM FARAG[2]

Abstract. Membrane Computing is an emergent and promising branch of Natural Computing. Designing P systems is heavy constitutes a difficult problem. The candidate has often had an idea about the problem solution form. On the other hand, finding the exact and precise configurations and rules is a hard task, especially if there is no tool used to help in the designing process. The clonal selection algorithm, which is inspired from the vertebrate immune system, is introduced here to help in designing a P system that performs a specific task. This paper illustrates the use of the clonal selection algorithm with adaptive mutation in P systems design and compares it with genetic algorithms previously used to achieve the same purpose. Experimental results show that clonal selection algorithm surpasses genetic algorithms with a great difference.

1. Introduction

Artificial intelligence can be seen as a combination of several research disciplines such as computer science, physiology, philosophy, sociology, biology, physics and chemistry. Enormous successes have been achieved through modeling of biological and natural intelligence [4] resulting in what is called natural computing.

The natural computing can be classified into the three following branches.

- Bio-inspired approaches
- Artificial Life
- Computing With Natural Means

The above branches - depicted in figure 1 with their fields - together with logic, deductive reasoning, expert systems, case-based reasoning and symbolic

Received by the editors: March 20, 2011.

2010 *Mathematics Subject Classification.* 68T20, 92D25, 92F05.

1998 *CR Categories and Descriptors.* I.2.8 [**Computing Methodologies**]: artificial intelligence – *Problem Solving, Control Methods, and Search*; I.1.2 [**Computing Methodologies**]: Symbolic and algebraic manipulation – *Algorithms*.

Key words and phrases. membrane computing, P systems, artificial immune system, clonal selection algorithm.

machine learning systems form a big part of Artificial Intelligence (AI). A Brief preview of each category is described below.

Bio-inspired approaches are inspired from nature for the development of novel problem-solving techniques. Bio-inspired approaches include the following fields: Artificial Neural Networks inspired by the functioning of mammalian brain [22, 36], Evolutionary Algorithms motivated by evolutionary biology [21, 34], Simulated Annealing which borrows ideas from the annealing of metals and glasses [7, 30], Swarm Intelligence which is based on the collective behavior of social organisms[8, 20], Artificial Immune Systems inspired by the vertebrate immune system [6, 9, 23, 24, 25] and Growth and Developmental Models which are based upon the growth and development processes of living organisms[31].

The synthesis of natural phenomena using computers is the second branch of natural computing that provides new tools for synthesizing and studying of natural phenomena which can be used to test biological theories that cannot be tested via traditional experimental and analytic techniques. There are basically two main approaches to the simulation and emulation of nature in computers: using tools for studying the fractal geometry of nature and using artificial life techniques[23].

There are a number of techniques for modeling fractal patterns and structures; these techniques include cellular automata [1, 33], L-systems [2], iterated function systems [19, 27, 28], particle systems [37] and Brownian motion [11, 29].

Computing with Natural Means (molecular computing) is the third branch of natural computing that employs natural materials (e.g., molecules) for computing. Computing with natural means is the approach that brings the most radical change in paradigm. The question that led to the thinking and creation of this approach was: "What are the other means or media which can be used to perform computation in place of silicon?" Motivated by the need to identify alternative media for computing, researchers are now trying to design new computers based on molecules, such as: Membrane Computing [13, 14, 15, 17, 18], DNA Computing [16] and Quantum Computing [3].

All forms of molecular computing are currently in their infancy. But in the long run they are likely to replace traditional silicon computers which face barriers in reaching higher levels of performance. This paper will present how clonal selection algorithm, which is inspired from the human body immunity, can be used to help in designing P systems. This paper is organized as follows: section 2 represents a background about P systems. Section 3 deals with the clonal selection algorithm which will be used as a helping tool for designing P systems. Section 4 tackles the problem of designing P systems and the

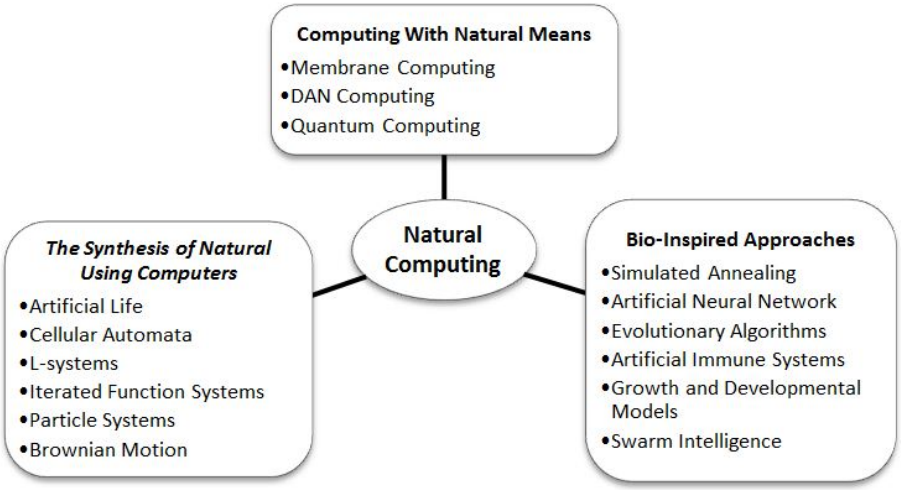


FIGURE 1. illustration of natural computing branches

experimental results of applying the clonal selection algorithm. Finally section 5 gives some conclusions and future work remarks.

2. MEMBRANE COMPUTING

Membrane computing (MC) or P systems, is an area of computer science aiming to abstract computing ideas and models from the structure and the functioning of living cells, as well as from the way the cells are organized in tissues or higher order structures [23]. Membrane computing (P systems) is the branch of Molecular Computing initiated by Gheorghe Paun discussed in a paper entitled Computing With Membranes [15] and this is the reason for calling it P systems.

A P system is a computing model which abstracts the way alive cells process chemical compounds in their compartmental structure. In short, in the regions defined by a membrane structure, one finds objects which evolve according to given rules. The objects can be described by symbols or strings of symbols. By using the rules in a nondeterministic and maximally parallel manner, one gets transitions between system configurations. A sequence of transitions is a computation. With a halting computation, one can associate a result, in the form of objects present in a given membrane in the halting configuration or expelled from the system during computation.

Various ways of controlling the transfer of objects from a region to another and applying the rules, as well as possibilities to dissolve, divide, create, or move membranes, were considered. Moreover, tissue P systems, neural P systems, and population P systems were investigated. Many of these variants lead to computationally universal systems, while several variants with an enhanced parallelism are able to "solve" NP-complete problems in polynomial (often, linear) time, by making use of an exponential space. A series of applications, in biology, linguistics, computer science, management, and many other areas were reported [13].

Formally, a P system with active membranes is a construct of the form below:

$$\prod = (O, H, \mu, \omega_1, ..., \omega_m, R)$$

where:

(1) $m \leftarrow 1$(the initial degree of the system);
(2) O is the alphabet of objects;
(3) H is a finite set of labels for membranes;
(4) μ is a membrane structure, consisting of m membranes initially having neutral polarizations, labeled (not necessarily in a one-to-one manner) with elements of H;
(5) $\omega_1, ..., \omega_m$ are strings over O, describing the multisets of objects placed in the m regions of μ;
(6) R is a finite set of developmental rules, of the following forms:

a: $[a \rightarrow v]_h^e$, for $h \in H, e \in \{+, -, 0\}, a \in O, v \in O^*$
Object evolution rules, associated with membranes and depending on the label and the charge of the membranes. Hint: only for simplicity, the label is written only one time outside the brackets and the internal label is omitted.

b: $a[v]_h^{e_1} \rightarrow [b]_h^{e_2}$, for $h \in H, e \in \{+, -, 0\}, a, b \in O$
Communication rules: an object is introduced in the membrane, and possibly modified during this process; the polarization of membrane can also be modified, but its label may not.

c: $[a]_h^{e_1} \rightarrow []_h^{e_2}$, for $h \in H, e_1, e_2 \in \{+, -, 0\}, a, b \in O$
Out-communication rules; an object is sent out of the membrane, and possibly modified during this process; the polarization of the membrane can also be modified.

d: $[a]_h^e \to b$, for $h \in H, e \in \{+, -, 0\}, a, b \in O$
Dissolving rules; in reaction with an object, a membrane can be dissolved, while the object specified in the rule can be modified.

e: $[a]_h^{e_1} \to [b]_h^{e_2}[c]_h^{e_3}$, for $h \in H, e_1, e_2, e_3 \in \{+, -, 0\}, a, b, c \in O$
Division rules for elementary membranes: in reaction to an object, the membrane is divided into two membranes with the same label, and possibly of different polarizations. The object specified in the rule is replaced in the two new membranes possibly by new objects; the remaining objects are duplicated and may evolve in the same step by rules of type (a). It is possible to allow the change of membrane labels. For instance, a division rule can take the more general form below.

$$[a]_{h_1}^{e_1} \to [b]_{h_2}^{e_2}[c]_{h_3}^{e_3}, \text{ for } h_1, h_2, h_3 \in H, e_1, e_2, e_3 \in \{+, -, 0\}, a, b, c \in O$$

The change of labels can also be considered for rules of types (b) and (c). The possibility of dividing membranes into more than two copies or even of dividing non-elementary membranes can be considered. In such case, all inner membranes are duplicated in the new copies of the membrane.

It is important to note that in case of P systems with active membranes, the membrane structure evolves during computation by decreasing the number of membranes, due to dissolution operations and increasing the number of membranes by division. The increase can be exponential in a linear number of steps: using a division rule successively, due to the maximal parallelism, 2^n copies of the same membrane can be obtained. This is one of the most frequently investigated ways of obtaining an exponential working space in order to trade time for space and solve computationally hard problems, i.e. NP-complete problems, in polynomial or even linear time [13].

3. The Clonal Selection Algorithm

The clonal selection principle is an algorithm used by the immune system to describe the basic features of an *immune response* to an antigenic stimulus. The clonal selection principle is depicted in figure 2. The principle establishes the idea that only those cells that recognize the antigens proliferate, thus being selected against those which do not. Clonal selection operates on both T cells and B cells. The *immune response* occurs inside the lymph nodes .When an animal is exposed to an antigen, some subpopulation of its bone marrow's derived cells (B lymphocytes) respond by producing antibodies. Each cell secretes only one kind of antibody, which is relatively specific for the antigen. By binding to these immune receptors, with a second signal from accessory

cells, such as the T-helper cell, an antigen stimulates the B cell to proliferate (divide) and mature into terminal (non-dividing) antibody secreting cells, called *plasma cells*. While plasma cells are the most active antibody secretors, large B lymphocytes, which divide rapidly, also secrete Ab, albeit at a lower rate. While B cells secrete Ab, T cells do not secrete antibodies, but play a central role in the regulation of the B cell response and are core in cell mediated immune responses. Lymphocytes, in addition to proliferating or differentiating into plasma cells, can differentiate into long-lived B *memory cells*. Memory cells circulate through the blood, lymph and tissues, probably not manufacturing antibodies [32], but when exposed to a second antigenic stimulus commence differentiating into large lymphocytes capable of producing high affinity antibody, preselected for the specific antigen that had stimulated the primary response. Figure 2 depicts the clonal selection principle [10]. The main features of the clonal selection theory are described below:

- the new cells are copies of their parents (clone) subjected to a mutation mechanism with high rates (somatic hypermutation);
- elimination of newly differentiated lymphocytes carrying self-reactive receptors;
- proliferation and differentiation on contact of mature cells with antigens
- The persistence of forbidden clones, resistant to early elimination by self-antigens, as the basis of autoimmune diseases.

The analogy with natural selection [24] should be obvious, the fittest clones being the ones that best recognize antigen or, more precisely, the ones that are triggered best. For this algorithm to work, the receptor population or repertoire has to be diverse enough to recognize any foreign shape. A mammalian immune system contains a heterogeneous repertoire of approximately 10^{12} lymphocytes in human [32], and a resting (unstimulated) B cell may display around $10^5 - 10^7$ identical antibody-like receptors. The repertoire is believed to be complete, which means that it can recognize any shape.

In our case the repertoire contains P systems; each p system will represent an antibody. The best antibody achieves the smallest difference from our target (solution). The smallest difference is zero in our case. Look at subsection 4.1. for more details about the affinity measure.

4. The Problem and Experimental Results

Designing a P system evaluation rules in order to perform a specific task is a hard job. In many cases, the designer has an idea about membrane structure, initial multi-sets and approximately the set of rules necessary to describe the

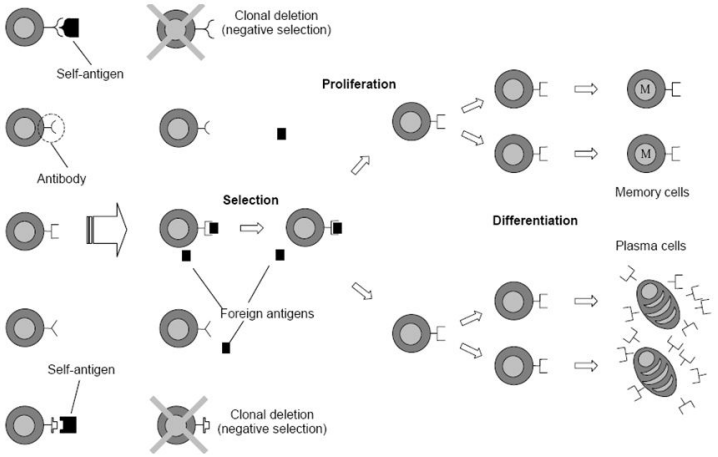


FIGURE 2. The clonal selection principle, small resting B cells created in the bone marrow, each carry a different receptor type. Those cells carrying receptors specific for the antigen, proliferate and differentiate into plasma and memory cells.

P system, but a small mistake in the description of the initial configuration or in the set of rules can lead to undesired consequences [12].

In this paper, an example of designing a P system using the clonal selection algorithms is presented. In order to do this, some information about the problem should be known, namely: membrane structure, initial multi-sets and approximately the set of rules necessary to describe the P system.

The clonal selection is applied on designing a simple p system for computing a simple mathematical operation, namely, square of 4. Clonal selection results will be compared with the previous work that solved the same problem using genetic algorithms; we used the same set of rules for repertoire initialization and the same affinity measure used in [12].

An initial repertoire of rules will be generated. Such repertoire will be evolved according to the clonal selection algorithm using cloning and mutation in order to reach the desired P system, of course by the help of a predefined affinity measure.

Only rules of type (a) and (d), namely evolution and dissolution rules respectively are used in our experiments. The initial repertoire is considered to have the same configuration, the initial configuration goes as follows.

$\prod = (O, H, \mu, \omega_1, ..., \omega_m, R_i)$

- The alphabet $O = x, y, z, m, n, u, v$

TABLE 1. The set of rules R

$r_1 : [x \to xy]_a$	$r_7 : [n \to m]_a$	$r_{13} : [x \to \lambda]_b$
$r_2 : [y \to yc]_a$	$r_8 : [u \to v]_a$	$r_{14} : [y \to \lambda]_b$
$r_3 : [c \to y_2]_a$	$r_9 : [m]_a \to y$	$r_{15} : [y \to c]_a$
$r_4 : [x \to yc]_a$	$r_{10} : [n]_a \to x$	$r_{16} : [c \to \lambda]_a$
$r_5 : [m \to n]_a$	$r_{11} : [u]_a \to c$	$r_{17} : [v \to m]_a$
$r_6 : [n \to u]_a$	$r_{12} : [v]_a \to x$	$r_{18} : [v]_a \to y$

- The set of labels $H = a, b$
- The membrane structure $\mu = [[]_a]_b$
- The initial multisets $w_a = x^2ym, w_b = \Phi$
- R is the set of rules such that $R_i \in R, R$ is explained in table 1.

4.1. **The Clonal Selection Algorithm.** The clonal selection algorithm with the properties described in section 3 is depicted below as Pseudocode.

```
Begin
 t=0;
 Initialize initial repertoire p(t);
 Identify the affinity function;
 Validate repertoire;
 Evaluate p(t);
   While (condition) do
   Begin
      1.t= t+1;
      2.Select C from p(t-1);
      3.Clone C to form C';
      4.mutate C' to form C";
      5.validate C";
      6.Select individuals from C"(t) and P(t-1) to create P(t);
      7.Metadymanics;
   End
End
```

A brief explanation of each function in the clonal selection algorithm is depicted below.

- Initial repertoire Initialization
  The initial repertoire is initialized with a randomly selected subset of rules, which are depicted in table 1. The maximum number of rules in each individual is 14 rules.

- Repertoire Validation
  The validation is made by ensuring that: for each P system and each membrane. No two rules are triggered by the same object, i.e. if a P system $\prod_1$ contains the following rules:
  $r_1 : [n]_a \to x$
  $r_2 : [n]_a \to c$
  $r_3 : [n]_a \to x$
  $r_4 : [x \to xyz]_b$
  $r_5 : [x \to \lambda]_b$

  Two rules from r_1, r_2 and r_3 are selected randomly to be removed from $\prod_1$; also one rule is selected randomly from r_4 and r_5 to be deleted.
- Repertoire evaluation and affinity measure
  The affinity measure is defined to select the highest affinity individuals. According to our example square of 4, the affinity measure is the absolute difference between the count of object z in membrane b in the halting configuration of the P system and the expected number of such objects in the ideal state, i.e., 16 objects of z. In order to prevent non-ending computations, the number of computations is limited to 20 steps.
- The selection mechanism
  The repertoire is evaluated, then the highest affinity members are selected in order to be cloned. The mutation operation is performed on the cloned members. The new population's individuals are selected from the old one and from the mutated cloned members. This ensures that our population is continuously enhanced by time. Results show that selecting the best individuals is always better than replacing the old generation by a new one.
- Mutation
  Mutation is applied as follows: Given a rule $[x \to y]_h$ such that $x \in O$, and $y \in O^*$, the mutation operator changes the object x by one from O other than x, or the object w, such that $w \in y$, by one object from $O - \{w\}$ or by λ.

  For a dissolution rule $[x]_h \to y$, the mutation operator changes the object x by a different one from O, and y by a different one from $O \cup \{\lambda\}$.
- Metadymanics
  To keep repertoire diversity, and enhance the exploration of solutions in space, a number of randomly generated individuals from the set of rules R are added in each iteration to the repertoire.

- Algorithm setting
  The following setting is used in our experiments; this setting is the same setting mentioned in [12] to enable us compare both algorithms: Repertoire size = 30 individuals, maximum number of generations= 30.

4.2. **Experimental Results.** Table 2 explains a comparison between 4 experiments using genetic algorithm depicted from [12] and three experiments using clonal selection with a new mutation mechanism. Each experiment consists of 30 runs. Genetic algorithm uses a fixed mutation rate in each experiment while we use clonal selection with mutation ranges which are applied as follows: The first experiment mutation range was from 0.1 to 0.4. The second experiment mutation range was from 0.5 to 1.0. The third one was from 0.1 to 1.0. It is clear from observing clonal selection results that low mutation rates give good results. On the other hand, it is too difficult for high mutation rates to find a solution. This situation is reversed in genetic algorithm cases where high rates find a solution with great difficulty, and low rates find no solutions.

TABLE 2. A comparison between clonal selection algorithm and genetic algorithm

experiment	Clonal Selection		Genetic Algorithm		
	Mutation	successful runs	Crossover	Mutation	successful runs
1	0.1 to 0.4	17/30	0	0.5	0/30
			0.5	0.5	0/30
2	0.5 to 1.0	0/30	0.8	0.8	1/30
3	0.1 to 1.0	15/30	1.0	0.8	1/30

Instead of the usual mutation method we used an adaptive mutation mechanism, i.e. mutation value is proportional to the individual affinity, high affinity individuals have low mutation value and low affinity individuals have high rate of mutation. This proposed adaptive mutation takes into consideration that good solutions don't distorted too much, on the other hand low affinity solutions needs more changes, so we assign it higher mutation rate.

One can also observe that there is no significant difference in results between the first mutation's range (0.1 to 0.4) and the third one (0.1 to 1.0). This is because the range (0.1 to 0.4) is applied in the two cases, and the best individuals are chosen to be included in the new repertoire.

Cloning makes the repertoire almost full with good solutions, which means that the algorithm can fall in a local optimum solution. However this is prevented by using meta-dynamics mechanism which adds randomly initializes

TABLE 3. The best P systems rules generated by the clonal selection algorithm

$[z \to z, z]_a$	$[y \to m]_a$	$[v]_a \to y$
$[n \to u]_a$	$[u]_a \to n$	$[x \to z, z]_a$
	$[m \to n]_a$	

individuals to keep repertoire diversity. This maintains the balance between exploration and exploitation.

It is clear that clonal selection algorithm surpasses genetic algorithm, at a time when genetic algorithm finds one solution from 30 runs, clonal selection algorithm find 17 solutions from 30 runs. This is because of cloning and adaptive mutation which differentiates between individuals according to an affinity value. Furthermore, clonal selection algorithm finds more than one solution in the same run. This is a very important advantage, where one can choose the most appropriate initial configuration and structure when one uses this algorithm in designing a P system for more complex problems. It also finds solutions before reaching half of the maximum generation's number in most runs. This means that the algorithm converges are very fast.

Plingua simulator [16] is used for calculating the affinity of each P system in the repertoire. One of the best P system's rules that achieved affinity measure is depicted in table 3. Table 3 rules are used for generating a *plingua* code illustrated below. This code is executed and the generated output determines the affinity of these rules. A graphical representation that explains the *plingua* code execution is depicted in figure 3.

```
@model<membrane_creation>
def numOfZs()
{
  @mu = [[]'a]'b;
  @ms(a) = x,x,y,m;
  @ms(b)=#;

  [z-->z,z]'a;  [y-->m]'a;  [v]'a-->y;
  [n-->u]'a;    [u]'a-->n;  [x-->z,z]'a;
  [m-->n]'a;
}
def main()
{
  call numOfZs();
}
```

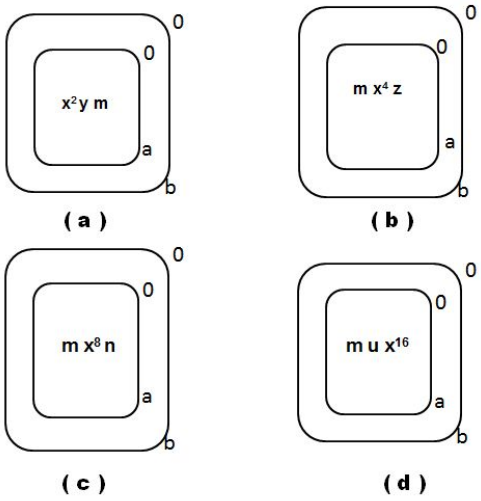


FIGURE 3. . Representation of the membranes generated by the p system rules depicted in table 3, (a) represents the initial membrane and (d) represents the final one.

5. Conclusion and Future Work

Membrane computing is an interesting area of research. Designing a P system to solve complex real world problems is not easy, and the researcher has no alternatives to do this hard task by hand. Moreover, Membrane Computing solutions to real-life problems need to be quite precise in the design in order to find a sharp simulation of the processes [35]. For all these reasons, the candidate explored the use of Clonal Selection Algorithm as an aid for designing P systems and as an alternative of genetic algorithms. Results show that clonal selection can find 17 solutions out of 30 runs while genetic algorithm finds one solution out of 30 runs.

Many questions arise regarding the automation of P systems design using Clonal selection algorithm. One of them, the affinity measure, being the most complex factor, discusses how the researcher can determine that a P system is better than another. In the candidate's example, a simple problem is used, but in more complex problems, it needs more work. The second question is what about the mutation which is responsible for the repertoire maturity. In this paper, mutation is applied to only objects, but it could be applied to membrane structures, polarizations, activation and inactivation of rules.

Another issue is what the appropriate selection method is, which is more adequate for such types of applications. Meta-dynamics also needs a deeper

view, and it could be implemented by applying very high mutation rates to a number of selected members besides adding some new randomly generated ones.

In this paper, the use of clonal selection algorithm is just illustrated as an aid in designing a simple P system, but the target is to use clonal selection in more complex real world problems. Hybridization between bio-inspired approaches [9], depicted on figure 1, may be used for the automation of P systems design, and may result in benefits better than clonal selection only, so this is another open point of research.

References

[1] A. Ilachinski, *Cellular automata: A discrete universe*, World Scientific, Singapore, 2001.

[2] A. Lindenmayer, *Mathematical models for cellular interaction in development, Parts I and II*, Journal of Theoretical Biology, 18(1968), No. 3, pp. 280-315.

[3] A.O. Pittenger, *An introduction to quantum computing algorithms*, Birkhauser, 2000.

[4] A.P. Engelbrecht, *Computational Intelligence: An Introduction*, Second Edition, John Wiley and Sons. 2007.

[5] C.S. Calude, G. Paun, Computing with cells and atoms. Taylor & Francis. 2001.

[6] D. Dasgupta, *Artificial immune systems and their applications*, Springer-Verlag, 1999.

[7] E. Aarts, and J. Korst, *Simulated Annealing and Boltzman machines - A stochastic approach to combinatorial optimization and neural computing*, New York, USA :John Wiley & Sons, Inc., 1989.

[8] E. Bonabeau, M. Dorigo and T. Theraulaz, *Swarm intelligence: From natural to artificial systems*, New York: Oxford University Press, 1999.

[9] E. Nabil, A. Badr and I. Farag, *An immuno-hybrid genetic algorithm*, International Journal of Computers, Communications & Control, 4(2009), No. 4, pp. 374-385.

[10] F. M. Burnet, *Clonal Selection and After*, in Theoretical Immunology, G. I. Bell , A. S. perelson,G. H. Jr. Pimbley, ed., Marcel Dekker Inc., 1978, pp. 63-85 .

[11] Fournier, D. Fussell, L. Carpenter, *Computer rendering of stochastic models*, Communications of the ACM, 25(1982), no. 6, pp. 371-384.

[12] G. Escuela, M.A. Gutirrez-Naranjo, *An application of Genetic Algorithms to Membrane Computing*, In M.A. Martinez-del-Amor, Gh. Paun, I. Prez-Hurtado, A. Riscos-Nunez (Eds.), Eighth Brainstorming Week on Membrane Computing (BWMC 2010). Fnix Editora, 2010.

[13] G. Paun, *Applications of membrane computing*, Springer-Verlag, Berlin, 2002.

[14] G. Paun, *Computing with Membranes*, Journal of Computer and System Sciences, 61(2000), No. 1, pp. 108-143.

[15] G. Paun, *Computing with membranes*,TUCS Report 208, Turku Center for Computer Science, 1998.

[16] G. Paun, G. Rozenberg, A. Saloma, *DNA computing*, Springer-Verlag, 1998.

[17] G. Paun, *Membrane Computing: An Introduction*, Springer-Verlag, Berlin, 2002.

[18] http://psystems.disco.unimib.it/.

[19] J. Hutchinson, *Fractals and self-similarity*, Indiana Journal of Mathematics, 30(1981), No. 5, 713-747.

[20] J. Kennedy, R. Eberhart, Y. Shi, *Swarm intelligence*, Morgan Kaufmann Publishers, 2002.

[21] J.J. Holland, *Adaptation in natural and artificial systems*, MIT Press, 1975.
[22] L. Fausett, *Fundamentals of neural networks: architectures, algorithms, and applications*, Prentice-Hall, 1994.
[23] L.N. De Castro, F.J. Von Zuben, *Recent Developments in Biologically Inspired Computing*, Idea Group Publishing, 2005.
[24] L.N. De Castro ,F.J. Von Zuben, *Artificial Immune Systems: Part I - Basic Theory and Applications*, Technical Report - RT DCA, 1999.
[25] L.N. De Castro, J.I. Timmis, *Artificial immune systems: A new computational intelligence approach*, Springer-Verlag, 2002.
[26] M. Garca-Quismondo, R. Gutierrez-Escudero, I. Perez-Hurtado,M.J. Perez-Jimenez, A. Riscos-Nunez, *An overview of P-Lingua 2.0.*,in Membrane Computing,Paun, Gheorghe and Prez-Jimnez, Mario and Riscos-Nunez, Agustin and Rozenberg, Grzegorz and Salomaa, Arto,ed. ,Springer Berlin, 5957(2010), pp.264-288.
[27] M.F. Barnsley, S. Demko, *Iterated function systems and the global construction of fractals*, Proceedings of the Royal Society of London, 339 (1985), No. 1817, pp. 243-275.
[28] M.F. Barnsley, *Fractals everywhere*, Academic Press, 1988.
[29] R.F. Voss, *Random fractals forgeries*. In RA. Earnshaw, Ed., Fundamental Algorithms for Computer Graphics, Berlin: Springer-Verlag,1985, pp. 805-835.
[30] S. Kirkpatrick, C.D. Gerlatt, M.P. Vecchi, *Optimization by simulated annealing*, Science, 220(1983), No. 4598, pp. 671-680.
[31] [31] S. Kumar, P.J. Bentley, *On growth, form and computers*, Academic Press, 2003.
[32] S. Perelson, M. Mirmirani, G.F. Oster, *Optimal Strategies in Immunology I. B-Cell Differentiation and Proliferation*, Journal of mathematical biology, 3(1978),No. 3, pp. 325-67.
[33] S. Wolfram, *Cellular automata and complexity*, Perseus Books, 1994.
[34] T. Back, DB. Fogel, Z. Michalewicz, *Evolutionary computation 2: advanced algorithms and operators*, Bristol and Philadelphia: Institute of Physics Publishing (IOP), 2000.
[35] T. Hinze, T. Lenser, G. Escuela, I. Heiland, S. Schuster, *Modeling Signaling Net- works with Incomplete Information about Protein Activation States: A P System Framework of the KaiABC Oscillator*, Lecture Notes in Computer Science, 5957(2010), pp. 316-334.
[36] W. McCulloch, W. Pitts, *A logical calculus of the ideas immanent in nervous activity*, Bulletin of Mathematical Biophysics 52(1990), No. 1, pp. 99-155.
[37] W.T. Reeves, *Particle systems - A technique for modeling a class of fuzzy objects*, ACM Transactions on Graphics, 2(1983), no. 2, pp. 91-108.

[1]Department of Computer Science, Faculty of Information Technology Misr University for Science and Technology, Al-Motamayez District, Postal code: 15525, 6th of October City, Egypt

[2] Department of Computer Science, Faculty of Computers and Information Cairo University, 5 Dr. Ahmed Zewail Street, Postal Code: 12613,Orman, Giza, Egypt

E-mail address: emadnabilcs@gmail.com
E-mail address: a.badr.fci@gmail.com, i.farag@fci-cu.edu.eg

STUDIA UNIV. BABEŞ–BOLYAI, INFORMATICA, Volume **LVI**, Number 1, 2011

AN EXPERIMENT ON PROTEIN STRUCTURE PREDICTION USING REINFORCEMENT LEARNING

GABRIELA CZIBULA, MARIA-IULIANA BOCICOR AND ISTVAN-GERGELY CZIBULA

Abstract. We are focusing in this paper on investigating a reinforcement learning based model for solving the problem of predicting the bidimensional structure of proteins in the hydrophobic-polar model, a well-known *NP*-hard optimization problem, important within many fields including bioinformatics, biochemistry, molecular biology and medicine. Our model is based on a *Q-learning* agent-based approach. The experimental evaluation confirms a good performance of the proposed model and indicates the potential of our proposal.

1. Introduction

Combinatorial optimization is the seeking for one or more optimal solutions in a well defined discrete problem space. In real life approaches, this means that people are interested in finding efficient allocations of limited resources for achieving desired goals, when all the variables have integer values. As workers, planes or boats are indivisible (like many other resources), the Combinatorial Optimization Problems (COPs) receive today an intense attention from the scientific community.

The current real-life COPs are difficult in many ways: the solution space is huge, the parameters are linked, the decomposability is not obvious, the restrictions are hard to test, the local optimal solutions are many and hard to locate, and the uncertainty and the dynamicity of the environment must be taken into account. All these characteristics, and others more, constantly make the algorithm design and implementation challenging tasks. The quest

Received by the editors: March 20, 2011.

2000 *Mathematics Subject Classification.* 65K10, 68T05.

1998 *CR Categories and Descriptors.* I.2.6[**Computing Methodologies**]: Artificial Intelligence – *Learning*; I.2.8[**Computing Methodologies**]: Problem Solving, Control Methods, and Search – *Heuristic methods* .

Key words and phrases. Combinatorial optimization, Bioinformatics, Reinforcement Learning, Protein Folding.

for more and more efficient solving methods is permanently driven by the growing complexity of our world.

Yet, for COPs that are *NP*-hard, no polynomial time algorithm exists. Therefore, complete methods might need exponential computation time in the worst-case. This often leads to computation times too high for practical purposes. Thus, the use of approximate methods to solve COPs has received more and more attention. In approximate methods we sacrifice the guarantee of finding optimal solutions for the sake of getting good solutions in a significantly reduced amount of time.

Reinforcement Learning (RL) [1] is an approach to machine intelligence in which an agent can learn to behave in a certain way by receiving punishments or rewards on its chosen actions.

In this paper we aim at investigating a reinforcement learning based model for solving a well known optimization problem within bioinformatics, the problem that refers to predicting the structure of a protein from its amino acid sequence. Protein structure prediction is an *NP*-complete problem, being one of the most important goals pursued by bioinformatics and theoretical chemistry; it is highly important in medicine (for example, in drug design) and biotechnology (for example, in the design of novel enzymes).

The model proposed in this paper for solving the bidimensional *protein folding* problem can be easily extended to the problem of predicting the three-dimensional structure of proteins. Moreover, the proposed model can be generalized to address other optimization problems. To our knowledge, except for the ant based approaches [2], the bidimensional *protein structure prediction* problem has not been addressed in the literature using reinforcement learning, so far.

The rest of the paper is organized as follows. Section 2 presents the main aspects related to the *protein structure prediction* problem. The reinforcement learning model that we propose for solving the bidimensional protein folding problem is introduced in Section 3. An experiment is given in Section 4 and in Section 5 we provide an analysis of the proposed reinforcement model, emphasizing its advantages and drawbacks. Section 6 contains some conclusions of the paper and future development of our work.

2. Protein Structure Prediction. The Hydrophobic-Polar Model

The determination of the three-dimensional structure of a protein, using the linear sequence of amino acids is one of the greatest challenges of bioinformatics, being an important research direction due to its numerous applications in medicine (drug design, disease prediction) and genetic engineering

(cell modelling, modification and improvement of the functions of certain proteins). Moreover, unlike the structure of other biological macromolecules (e.g., DNA), proteins have complex structures that are difficult to predict. Protein structure prediction is an important problem within the more general *protein folding* problem, and is also reffered in the literature as the *computational protein folding* problem [3]. Different computational intelligence approaches for solving the protein structure prediction problem have been proposed in the literature, so far.

An important class of abstract models for proteins are lattice-based models - composed of a lattice that describes the possible positions of amino acids in space and an energy function of the protein, that depends on these positions. The goal is to find the global minimum of this energy function, as it is assumed that a protein in its native state has a minimum free energy and the process of folding is the minimization of this energy [4].

One of the most popular lattice-models is Dill's Hydrophobic-Polar (HP) model [5].

In the folding process the most important difference between the amino acids is their hydrophobicity, that is how much they are repelled from water. By this criterion the amino acids can be classified in two categories: *hydrophobic* or *non-polar* (H) - the amino acids belonging to this class are repelled by water; *hydrophilic* or *polar* (P) - the amino acids that belong to this class have an affinity for water and tend to absorb it.

The HP model is based on the observation that the hydrophobic forces are very important factors in the protein folding process, guiding the protein to its native three dimensional structure.

The primary structure of a protein is seen as a sequence of n amino acids and each amino acid is classified in one of the two categories: hydrophobic (H) or hydrophilic (P). A *conformation* of the protein $\mathcal{P}$ is a function C, that maps the protein sequence $\mathcal{P}$ to the points of a two-dimensional cartesian lattice such that any two consecutive amino acids in the primary structure of the protein are neighbors (horizontally or vertically) in the bidimensional lattice. It is considered that any position of an amino acid in the lattice may have a maximum number of 4 neighbors: up, down, left, right.

A configuration C is *valid* if it is a *self avoiding path*, i.e the mapped positions of two different amino acids must not be superposed in the lattice.

Figure 1 shows a configuration example for the protein sequence $\mathcal{P} = HHPH$, of length 4, where the hydrophobic amino acids are represented in black and the hydrophilic ones are in white.

The energy function in the HP model reflects the fact that hydrophobic amino acids have a propensity to form a hydrophobic core. Consequently the energy function adds a value of -1 for each two hydrophobic amino acids

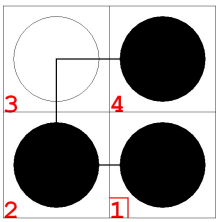


FIGURE 1. A protein configuration for the sequence $\mathcal{P} = HHPH$, of length 4. Black circles represent hydrophobic amino acids, while white circles represent hydrophilic ones. The configuration may be represented by the sequence LUR. The value of the energy function for this configuration is -1.

that are mapped by C on neighboring positions in the lattice, but that are not consecutive in the primary structure $\mathcal{P}$. Such two amino acids are called topological neighbors. Any hydrophobic amino acid in a valid conformation C can have at most 2 such neighbors (except for the first and last aminoacids, that can have at most 3 topological neighbors).

The computational protein folding problem in the HP model is to find the conformation C whose energy is minimum. A solution for the bidimensional HP protein folding problem, corresponding to an n-length sequence P could be represented by a $n-1$ length sequence $\pi = \pi_1\pi_2...\pi_{n-1}$, $\pi_i \in \{L, R, U, D\}$, $\forall 1 \leq i \leq n-1$, where each position encodes the direction of the current amino acid relative to the previous one (L-left, R-right, U-up, D-down). As an example, the solution configuration corresponding to the sequence presented in Figure 1 is LUR.

3. A Reinforcement Learning Model for Solving the Bidimensional Protein Structure Prediction Problem

In the folowing we are addressing the *Bidimensional Protein Structure Prediction* problem ($BPSP$), more exactly the problem of predicting the bidimensional structure of proteins, but our model can be easily extended to the three-dimensional protein folding problem.

Let us consider, in the following, that $\mathcal{P} = p_1p_2...p_n$ $(n \geq 3)$ is a protein HP sequence consisting of n amino acids, where $p_i \in \{H, P\}$, $\forall 1 \leq i \leq n$. As we have indicated in Section 2, the bidimensional structure of $\mathcal{P}$ will be represented as an $n-1$-dimensional sequence $\pi = \pi_1\pi_2...\pi_{n-1}$, where each element π_k $(1 \leq k \leq n)$ encodes the direction (L, U, R or D) of the current amino acid location relative to the previous one.

The RL task associated to the $BPSP$ is defined as follows.

The state space $\mathcal{S}$ (the agent's environment) will consist of $\frac{4^n-1}{3}$ states, i.e $\mathcal{S} = \{s_1, s_2, ..., s_{\frac{4^n-1}{3}}\}$. The *initial state* of the agent in the environment is s_1. A state $s_{i_k} \in \mathcal{S}(i_k \in [1, \frac{4^n-1}{3}])$ reached by the agent at a given moment after it has visited states $s_1, s_{i_1}, s_{i_2}, ...s_{i_{k-1}}$ is a *terminal* (final or goal) state if the number of states visited by the agent in the current sequence is $n-1$, i.e. $k = n-2$. A path from the initial to a final state will represent a possible bidimensional structure of the protein sequence $\mathcal{P}$.

The action space $\mathcal{A}$ consists of 4 actions available to the problem solving agent and corresponding to the 4 possible directions $L(Left)$, $U(Up)$, $R(Right)$, $D(Down)$ used to encode a solution, i.e $\mathcal{A} = \{a_1, a_2, a_3, a_4\}$, where $a_1 = L$, $a_2 = U$, $a_3 = R$ and $a_4 = D$.

The transition function $\delta : \mathcal{S} \to \mathcal{P}(\mathcal{S})$ between the states is defined as in Formula 1.

(1)
$$\delta(s_{\frac{4^k-1}{3}+i}, a_l) = s_{\frac{4^{k+1}-1}{3}+4\cdot(i-1)+l} \; \forall k \in [0, n-1], \; \forall i, 1 \leq i \leq 4^k \; \forall l, 1 \leq l \leq 4.$$

This means that, at a given moment, from a state $s \in \mathcal{S}$ the agent can move in 4 successor states, by executing one of the 4 possible actions. We say that a state $s' \in \mathcal{S}$ that is accessible from state s, i.e $s' \in \bigcup_{a \in \mathcal{A}} \delta(s, a)$, is the *neighbor* (*successor*) state of s.

The transitions between the states are equiprobable, the transition probability $P(s, s')$ between a state s and each neighbor state s' of s is equal to 0.25 .

Let us consider a path π in the above defined evironment from the initial to a final state, $\pi = (\pi_0\pi_1\pi_2 \cdots \pi_{n-1})$, where $\pi_0 = s_1$ and $\forall 0 \leq k \leq n-2$ the state π_{k+1} is a *neighbor* of state π_k. The sequence of actions obtained following the transitions between the successive states from path π will be denoted by $a_\pi = (a_{\pi_0} a_{\pi_1} a_{\pi_2} \cdots a_{\pi_{n-2}})$, where $\pi_{k+1} = \delta(\pi_k, a_{\pi_k})$, $\forall 0 \leq k \leq n-2$. The sequence a_π will be refered as the *configuration* associated to the path π and it can be viewed as a possible bidimensional structure of the protein sequence $\mathcal{P}$. Consequently we can associate to a path π a value denoted by E_π representing the energy of the bidimensional configuration a_π of protein $\mathcal{P}$ (Section 2).

The *BPSP* formulated as a RL problem will consist in training the agent to find a path π from the initial to a final state that will corespond to the bidimensional structure of protein $\mathcal{P}$ given by the coresponding configuration a_π and having the minimum associated energy.

It is known that the estimated utility of a state [6] in a reinforcement learning process is the estimated *reward-to-go* of the state (the sum of rewards

received from the given state to a final state). So, after a reinforcement learning process, the agent learns to execute those transitions that maximize the sum of rewards received on a path from the initial to a final state.

As we aim at obtaining a path π having the minimum associated energy E_π, we define the reinforcement function as follows: if the transition generates a configuration that is not *valid* (i.e self-avoiding) (see Section 2) the received reward is 0.01; the reward received after a transition to a non terminal state is a small positive constant greater than 0.01 (e.q 0.1); the reward received after a transition to a final state π_{n-1} after states $s_1, \pi_1, \pi_2, ...\pi_{n-2}$ were visited is minus the energy of the bidimensional structure of protein $\mathcal{P}$ corresponding to the configuration a_π.

Considering the reward defined as indicated above, as the learning goal is to maximize the total amount of rewards received on a path from the initial to a final state, it can be easily shown that the agent is trained to find a self avoiding path π that minimizes the associated energy E_π.

3.1. **The learning process.** During the training step of the learning process, the agent will determine its *optimal policy* in the environment, i.e the *policy* that maximizes the sum of the received rewards.

For training the $BPSP$ agent, we propose a Q-learning approach. The idea of the training process is the following:

- The Q values are initialized with 0.
- During some training episodes, the agent will experiment (using the ϵ-Greedy action selection mechanism) some (possible optimal) paths from the initial to a final state, updating the Q-values estimations according to the $Q - learning$ algorithm [7].
- During the training process, the Q-values estimations converge to their exact values, thus, at the end of the training process, the estimations will be in the vicinity of the exact values.

After the training step of the agent has been completed, the solution learned by the agent is constructed by starting from the initial state and following the *Greedy* mechanism until a solution is reached. From a given state i, using the *Greedy* policy, the agent transitions to a neighbor j of i having the maximum Q-value. Consequently, the solution of the $BPSP$ reported by the RL agent is a path $\pi = (s_1\pi_1\pi_2\cdots\pi_{n-2})$ from the initial to a final state, obtained following the policy described above. We mention that there may be more than one optimal policy in the environment determined following the *Greedy* mechanism described above. In this case, the agent may report a single optimal policy of all optimal policies, according to the way it was designed.

It is proven in [8] that the Q-values learned converge to their optimal values as long as all state-action pairs are visited an infinite number of times. Consequently, the configuration a_π corresponding to the path π learned by the $BPSP$ agent converges, in the limit, to the sequence that corresponds to the bidimensional structure of protein $\mathcal{P}$ having the minimum associated energy.

4. Experiment

In this section we aim at experimentally evaluating the proposed *reinforcement learning* approach.

Let us consider a bidimensional HP protein instance $\mathcal{P} = HPHPPHHP$ $HPPHPHHPPHPH$, consisting of twenty amino acids, i.e $n = 20$. The benchmark instance for the 2D HP Protein Folding Problem used in this study can be found in [9] and its known optimal energy value is $E^* = -9$. As we have presented in Section 3, the states space will consist of $\frac{4^{20}-1}{3}$ states. We have trained the $BPSP$ agent as indicated in Subsection 3.1. As proven in [8], the Q-learning algorithm converges to the optimal Q-values as long as all state-action pairs are visited an infinite number of times, the learning rate α is small (e.q 0.01) and the policy converges in the limit to the Greedy policy. We remark the following regarding the parameters setting:

- the learning rate is $\alpha = 0.01$ in order to assure the convergence of the algorithm;
- the discount factor for the future rewards is $\gamma = 0.9$;
- the number of training episodes is $19 \cdot 10^5$;
- the ϵ-Greeedy action selection mechanism was used. Regarding the ϵ parameter used for the *epsilon*-Greedy action selection mechanism during the training step, the following strategy was used: we have started with $\epsilon = 1$ in order to favor exploration, then after the training progresses ϵ is decreased until it reaches a small value, which means that at the end of the training exploitation is favorized.

Using the above defined parameters and under the assumptions that the state action pairs are equally visited during training, the solution reported after the training of the $BPSP$ agent was completed is the *configuration* $a_\pi = (RUULDLULLDRDRDLDRRU)$, determined starting from state s_1, following the *Greedy* policy (as we have indicated in Subsection 3).

The solution learned by the agent is represented in Figure 2 and has an energy of -9.

Consequently, the $BPSP$ agent learns the optimal solution of the computational bidimensional protein folding problem, i.e the bidimensional structure of the protein $\mathcal{P}$ that has a minimum associated energy (-9).

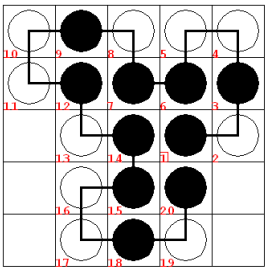


FIGURE 2. The learned solution is $RUULDLULLDRDRDLDRRU$. The value of the energy function for this configuration is -9.

5. Comparison with related approaches

Regarding the Q-learning approach introduced in Section 3 for solving the bidimensional protein folding problem, we remark the following. The training process during an episode has a time complexity of $\theta(n)$, where n is the length of the HP protein sequence. Consequently, assuming that the number of training episodes is k, the overall complexity of the algorithm for training the $BPSP$ agent is $\theta(k \cdot n)$.

In the following we will briefly compare our approach with some of the existing approaches. The comparison is made considering the computational time complexity point of view. Since for the most of the existing approaches the authors do not provide the asymptotic analysis of the time complexity of the proposed approaches, we can not provide a detailed comparison.

Genetic and evolutionary approaches were developed in [10, 11, 12] for predicting the bidimensional structure of proteins. An asymptotic analysis of the computational complexity for evolutionary algorithms (EAs) is difficult [13] and is usually done only for particular problems. Anyway, the number of generations (or equivalently the number of fitness evaluations) is the most important factor in determining the order of EA's computation time. In our view, the time complexity of an evolutionary approach for solving the problem of predicting the structure of an n-dimensional protein is at least $noOfRuns \cdot n \cdot noOfGenerations \cdot populationLength$. For large instances, it is likely (even if we can not rigurously prove) that the computational complexity of our approach is less than the one of an evolutionary approach.

Ant Colony Optimization (ACO) was already used for solving the protein folding problem in the HP model [14, 15]. Neumann et al. show in [16] how simple ACO algorithms can be analyzed with respect to their computational complexity on example functions with different properties, and also claim that

asymptotic analysis for general ACO systems is difficult. In our view, the time complexity of an ACO approach for solving the problem of predicting th structure of an n-dimensional protein is at least $noOfRuns \cdot n \cdot noOfIterations \cdot noOfAnts$. For large instances, it is likely (even if we can not rigurously prove) that our approach has a lower computational complexity.

Compared to the supervised classification approach from [17], the advantage of our RL model is that the learning process needs no external supervision, as in our approach the solution is learned from the rewards obtained by the agent during its training. It is well known that the main drawback of supervised learning models is that a set of inputs with their target outputs is required, and this can be a problem.

The main drawback of our approach is that a very large number of training episodes has to be considered in order to obtain accurate results and this leads to a slow convergence. In order to speed up the convergence process, further improvements, such as local search mechanisms will be considered. Anyway, we think that the direction of using reinforcement learning techniques in solving the protein folding problem is worth being studied and further improvements can lead to valuable results.

6. Conclusions and Further Work

We have proposed in this paper a reinforcement learning based model for solving the bidimensional protein structure prediction problem, a fundamental problem in computational molecular biology and biochemical physics. To our knowledge, except for the ant based approaches, the problem of predicting the bidimensional structure of proteins has not been addressed in the literature using reinforcement learning, so far. The model proposed in this paper can be easily extended to solve the three-dimensional computational protein folding problem, and moreover to solve other optimization problems.

We plan to extend the evaluation of the proposed RL model for other large HP protein sequences, to further test its performance. We will also investigate possible improvements of the RL model by analyzing a temporal difference approach [1], by using different reinforcement functions and by adding different local search mechanisms in order to increase the model's performance. An extension of the $BPSP$ model to a distributed RL approach will be also considered.

ACKNOWLEDGEMENT

This work was supported by CNCSIS - UEFISCDI, project number PNII - IDEI 2286/2008.

References

[1] Sutton, R.S., Barto, A.G., *Reinforcement Learning: An Introduction*, MIT Press, 1998.

[2] Dorigo, M., Stützle, T., *Ant Colony Optimization*, Bradford Company, Scituate, MA, USA, 2004.

[3] Dill, K.A.A., Ozkan, S.B.B., Weikl, T.R.R., Chodera, J.D.D., Voelz, V.A.A., *The protein folding problem: when will it be solved?*, Current Opinion in Structural Biology **17**, 2007, pp. 342-346.

[4] Anfinsen, C.B., *Principles that govern the folding of protein chains*, Science **181**, 1973, pp. 223–230.

[5] Dill, K., Lau, K., *A lattice statistical mechanics model of the conformational sequence spaces of proteins*, Macromolecules **22**, 1989, pp. 3986–3997.

[6] Russell, S., Norvig, P., *Artificial Intelligence - A Modern Approach*, Prentice Hall International Series in Artificial Intelligence, Prentice Hall, 2003.

[7] Dayan, P., Sejnowski, T., *TD(lambda) converges with probability 1*, Mach. Learn. **14**, 1994, pp. 295–301.

[8] Watkins, C.J.C.H., Dayan, P., *Q-learning*, Machine Learning **8**, 1992, pp. 279–292.

[9] Hart, W., Istrail, S., *HP benchmarks* http://www.cs.sandia.gov/tech_reports/compbio/tortilla-hp-benchmarks.html.

[10] Unger, R., Moult, J., *Genetic algorithms for protein folding simulations*, Mol. Biol. **231**, 1993, pp. 75–81.

[11] Zhang, X., Wang, T., Luo, H., Yang, Y., Deng, Y., Tang, J., Yang, M.Q., *3D protein structure prediction with genetic Tabu search algorithm*, BMC Systems Biology **4**, 2009, pp. 1–9.

[12] Chira, C., *Hill-climbing search in evolutionary models for protein folding simulations*, Studia **LV**, 2010, pp. 29–40.

[13] Hart, W.E., Belew, R.K., *Optimizing an arbitrary function is hard for the genetic algorithm*, In: Proceedings of the Fourth International Conference on Genetic Algorithms, Morgan Kaufmann, 1991, pp. 190–195.

[14] Shmygelska, A., Hoos, H., *An ant colony optimisation algorithm for the 2D and 3D hydrophobic polar protein folding problem*, BMC Bioinformatics **6**, 2005, pp. 1–22.

[15] Thalheim, T., Merkle, D., Middendorf, M., *Protein folding in the HP-model solved with a hybrid population based aco algorithm*, IAENG International Jurnal of Computer Science **35**, 2008, pp. 1–10.

[16] Neumann, F., Sudholt, D., Witt, C., *Computational complexity of ant colony optimization and its hybridization with local search.* In: Innovations in Swarm Intelligence, 2009, pp. 91–120.

[17] Ding, C.H.Q., Dubchak, I., *Multi-class protein fold recognition using support vector machines and neural networks*, Bioinformatics **17**, 2001, pp. 349–358.

Babeş-Bolyai University, Department of Computer Science, 1, M. Kogălniceanu street, 400084 Cluj-Napoca, Romania

E-mail address: {gabis,iuliana,istvanc}@cs.ubbcluj.ro

STUDIA UNIV. BABEŞ–BOLYAI, INFORMATICA, Volume **LVI**, Number 1, 2011

"PADSYNTH" SOUND SYNTHESIS ALGORITHM

PAUL NAŞCA

Abstract. The sound synthesizer has changed the way the musical sound is produced by offering nearly unlimited amount of musical timbres. Unfortunately, many of the current synthesizers lack the subjective "warmth" of the acoustic instruments. The current paper describes a new algorithm which produces aesthetically pleasant sounds. There is also described a simple approach for synthesizing sounds, optimized for the generation of the ensemble sounds. This approach can be used in other sound synthesis/sound processing algorithms to acheive a great degree of perceived quality.

1. Introduction

Nowadays there are many sound synthesizers and sound synthesis algorithms. Most of the synthesis algorithms are based on a handfull of approaches, like addition of many simple components or by simulating with the mathematical equations the acoustic instruments.

The music theory considers the musical sound as being composed of many sine waves with a frequency multiple of a fundamental frequency. For example, the note A with a fundamental frequency of 220 Hz has the harmonics with the frequencies of 440 Hz, 660 Hz, 880 Hz, and other frequencies multiples of 220 Hz. Each of these sine-wave components are called "harmonics".

The most used approach to generate harmonics is based on amplitude modulated by a stochastic process (called "micromodulations" [3]). However, using this approach it is very difficult to simulate the ensembles of instruments or choirs.

It seems it is complicated to use modulation because that the speed of modulation needs to be increased on higher harmonics [4] in "Simulating the ensemble effect".

Received by the editors: February 1, 2011.

2010 *Mathematics Subject Classification.* 00A65.

1998 *CR Categories and Descriptors.* H.5.5 [**Information Systems**]: Sound and Music Computing – *Signal analysis, synthesis, and processing.*

Key words and phrases. sound synthesis, harmonics, fast Fourier transform.

This paper describes a simple sound synthesis approach solves the above mentioned problem.

This approach adds another level between "harmonic"/"overtone" and the sine component of the sound. The proposed algorithm models a difficult category of sounds like those generated by the ensembles of instruments and choirs.

The basic idea is to consider the harmonics as being a narrowband signal instead of simple (or frequency/amplitude modulated) sine components.

The frequency-domain property of choirs is described in Jordi Bonada[1] paper, in which he tries to describe a technique to transform a solo sound to "unison choir" sound. His paper acknowledges that using sine components with pure amplitude modulation is not a good idea, because at higher frequency the harmonics overlap. His paper does not explain the cause, the how or the why, for example, the harmonics become wider and wider on higher frequencies according to a linear function. Also, even if he acknowledges that higher harmonics need to be smoothed in frequency, he does not explain how the slight detuning of lower sine components become a continuous frequency band on higher frequencies.

The algorithm described here considers that the harmonics become wider and wider (according to linear function) on higher frequencies, until they merge together into a single continuous band. If the fundamental frequency has a certain width (measured in Hz), the Nth harmonic contains multiples of the each sine frequency component of it. This causes the harmonic to have a bigger bandwidth.

Another paper [5] which describes that voices in a choir do not have exactly the same frequency (called "pitch scatter") and that voice's frequency changes very fast for the same perceived note (called "voice flutter").

In this paper I will present the algorithm "PADsynth" based on Ternstrom [5] definition of "pitch scatter" which - for more clarity - I will call from now as "bandwidth of each harmonic".

The first requirement of this algorithm is a frequency distribution of a single harmonic (usually a Gaussian distribution). Using this distribution, the desired harmonic are added to a large array which represent the signal amplitudes in the frequency domain. After this, it is done a single inverse Fourier transform (IFFT) by considering the phases of the signal as being at random. A long sample will result (usually few seconds long) which can be played at different speeds in order to obtain the desired pitch. This algorithm has been included in several software synthesizers, because of its simplicity and especially because of the high quality of musical instruments generated by it. It is also mentioned in other paper[2] where the algorithm is used in a low cost system-on-chip embedded system.

2. What PADsynth algorithm does

Most people perceive the sounds of ensembles, choirs and the sounds produced by slight detuning of instruments as being "pleasant" or "warm". But it is often believed that the pitched part of the instruments' timbre are composed only by a fundamental frequency and harmonics (another pure sine signals with different frequencies, usually on multiple frequencies).

The approach presented in this paper considers the harmonics (and overtones, in general) as being a "collection" of many sine components with very close frequencies and the phases of these components are random. This randomness of the phases makes the instrument have qualities that resemble the natural/acoustic instruments and ensembles.

Due to this result, it must be defined a parameter of the sound, called "the bandwidth of each harmonic" (BoEH) which represents the frequency spread of the sine components into each harmonic. For example, if we define BoEH as the highest frequency minus lowest frequency from a certain harmonic, for note A-4 (440 Hz) if there are sine components of 435Hz, 438Hz, 442 Hz and 445 Hz the BoEH is 10 Hz. If one harmonic contains many sine components, there are other ways to define BoEH, for example as the standard deviation of frequency values of the components.

In natural ensembles one of the property of the BoEH is that it is proportional to the overtone's frequency. For example the A-4 has harmonics 880 Hz and 1320 Hz, the bandwidth of them will be 20Hz and 30Hz. As a result, BoEH can be described as a single number: the bandwidth of the fundamental frequency, expressed in cents (where 1 cent is one hundredth of a halftone). From this parameter, we can express the BoEH in Hz for each overtone.

From this approach it is not difficult to realise what the implication of the ensembles and detuning on higher harmonics are. Also, BoEH approach predicts that ensembles of pitched sounds with many harmonics(like a huge choirs of singers) can result a hissing sound on the high part of the spectrum. This prediction was confirmed by analyzing the recording of a large number of singers when they pronounced different wowels ("10,000 Voices, The World Choir" - EMI Classics, 1992). The next spectrograms (Fig.2) shows vowel "A" analyzed using PRAAT [6] software.

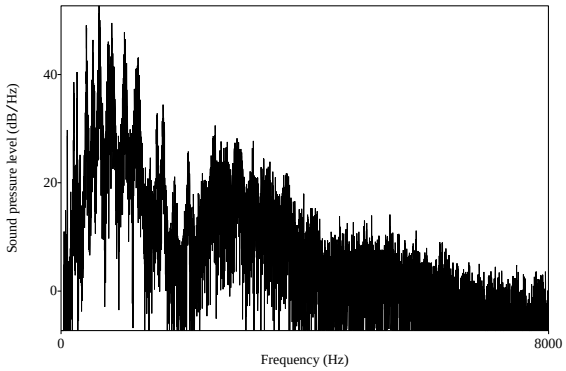


FIGURE 1. Spectrum of "A" vowel

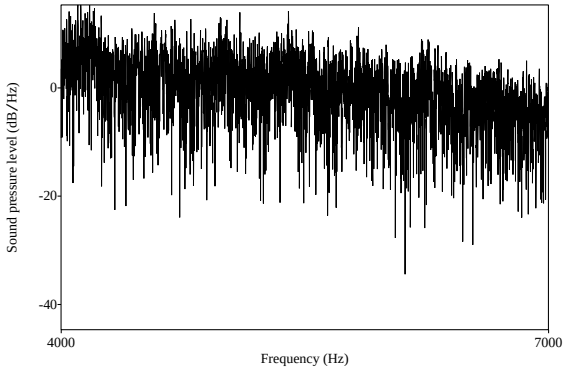


FIGURE 2. Closeup of the spectrum, to show the "hissing" part of the vowel

If the wovel "A" from this recording is passed through a highpass filter, the "hissing" becomes very noticeable for the listener.

3. How PADsynth algorithm works

The PADsynth algorithm generates subjectively pleasant sounds, even if its idea is more simple than of the other sound synthesis algorithms. It generates a perfectly looped wave-table sample which can be played back on different speeds in order to obtain the desired musical pitch. It easily generates sounds of ensembles, choirs, metallic sounds (bells) and many other types of sound. Also, this algorithm is a direct consequence of the BoEH approach. It was created by me at the end of 2003, and it was released on-line with example

implementations [7] under Public Domain. One of the interesting property of PADsynth is that it generates the "hissing" sound for large BoEH, similar to the "hissing" discussed before.

3.1. **PADsynth steps.** The basic steps of this algorithms are:

(1) Make a very large array which represents the amplitude spectrum of the sound (default all values are zero)
(2) Generates the distribution of each harmonic in frequency and add it to the array
(3) Randomize the phases to each frequency of the spectrum
(4) Do a single Inverse Fourier Transform of the whole spectrum. Usage of overlapping windows is not necessary, because there is only one single IFFT for the whole sample.

The resulting sample can be used as a wave-table to generate the desired note.

These four steps are represented graphically in fig.3.

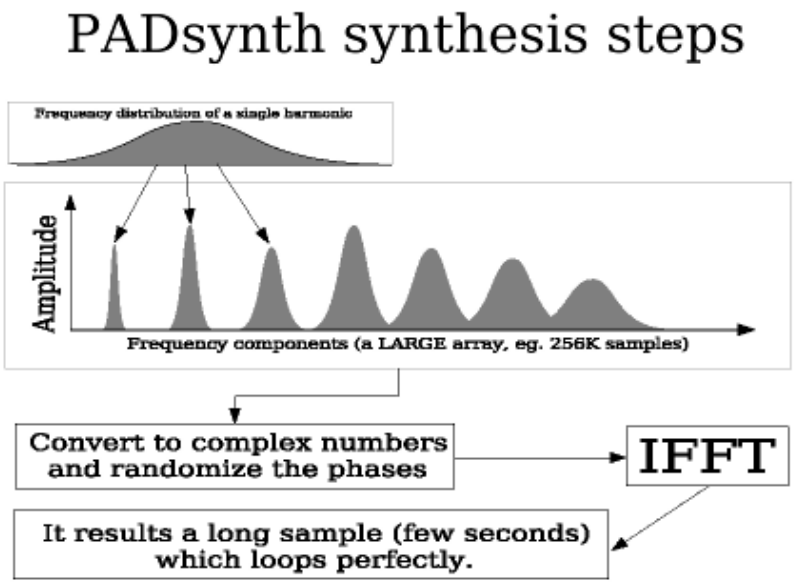


FIGURE 3. PADsynth Steps

There are some important facts of the PADsynth algorithm:

- The bandwidth of each harmonic which was described earlier into this paper is the parameter which gives the subjective quality of "warmth" or "ensemble".

- Another important parameter is the frequency distribution of each harmonic. For example, the sine components of the harmonic can be evenly spread or a Gaussian distribution can be used.

3.2. **PADsynth algorithm described in pseudo-code.** For a better understanding and for helping the implementation of this algorithm, the steps will be described in pseudo-code:

```
Input:
  N - wave-table size. It's recommanded to be a power of 2. This is,
      usually, a big number (like 262144)
  samplerate - the sample-rate (eg. 44100)
  f - frequency of the the fundamental note (eg. 440)
  bw - bandwidth of first harmonic in cents (eg. 50 cents); must be
       greater than zero
  number_harmonics - the number of harmonics; number_harmonics<(sample-rate/f)
  A[1..number_harmonics] - amplitude of the harmonics
Output:
  smp[0..N-1] - the generated wave-table

Internal variables:
  freq_amp[0..N/2-1] = {0,0,0,0,...,0}
  freq_phase[0..N/2-1], etc...

Functions:
  RND() returns a random value between 0 and 1
  IFFT()it is the inverse Fourier transform
  normalize_sample() normalizes samples between -1.0 and 1.0

  profile(fi,bwi){
      x=fi/bwi;
      return exp(-x*x)/bwi;
  };

Steps:
  FOR nh = 1 to number_harmonics
      bw_Hz=(pow(2,bw/1200)-1.0)*f*nh;
      bwi=bw_Hz/(2.0*samplerate);
      fi=f*nh/samplerate;
      FOR i=0 to N/2-1
          hprofile=profile((i/N)-fi,bwi);
          freq_amp[i]=freq_amp[i]+hprofile*A[nh];
```

```
    ENDFOR
ENDFOR

FOR i=0 to N/2-1
    freq_phase[i]=RND()*2*PI;
ENDFOR

smp=IFFT(N,freq_amp,freq_phase);
normalize_sample(N,smp);

OUTPUT smp
```

The frequency domain array ("freq_amp") data is represented below in fig. 4. Notice that on the highest frequencies the overtones merge and this cause the "hissing" discussed above.

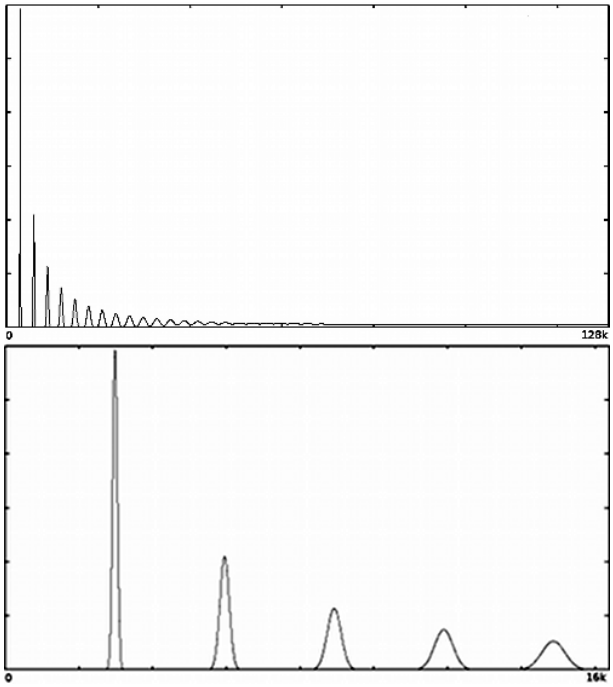


FIGURE 4. Frequency domain data (full and closeup)

3.3. **Suggestions regarding usage of PADsynth algorithm.** For best results, there are some recommendations of using PADsynth algorithm into a synthesizer.

- The algorithm produces a sample which is perfectly looped. It is recommanded to make the samples few seconds long to avoid making noticeable the repetition of the sound.
- For each new musical note the playing should start from a random position of the sample.
- For generating stereo sounds, a single sample is enough, but the sample have to be played from different positions for left/right channels.

4. Implementations of PADsynth

Since the day this algorithm was made public it was implemented into several open-source and commercial synthesizers. The first synthesizer which used this algorithm is ZynAddSubFX[16], an open-source audio synthesizer, mostly written by me. This synthesizer is widely distributed in most Linux software repositories. It has a large number of users, for example, on YouTube[15] there are available hundrieds videos related to ZynAddSubFX. PADsynth algorithm is used into one of its three synthesizer engines (other engines are based on additive and substractive synthesis methods). Other software synthesizers which use this algorithm are WhySynth[14], Kunquat[11], BR404[8], KarmaFX[10] and discoDSP Discovery Pro[9]. There is also, available a module [12] for SynthMaker [13] which implements PADsynth algorithm.

5. Conclusions

PADsynth algorithm is a simple but very versatile synthesis method. Despite of its low complexity, the sounds generated by it are subjectively pleasant and "warm", a characteristic seldom met in many digital synthesizers. Also, the "Bandwidth of each harmonic" approach to the musical instruments is a productive approach for creating new sound synthesis and sound processing algorithms.

References

[1] Jordi Bonada, *Voice solo to unison choir transformation*, Music Technology Group, Institut Universitari de lAudiovisual Universitat Pompeu Fabra, Barcelona, 2005

[2] Akansha Pilley, N.G.Bawane, Jagruti Sanghavi *A Stand-alone and Less Power Consumption Digital Music Synthesizer using a Low Cost SoC*, International Journal of Electronics Engineering, 2010

[3] J. C. Risset, *Stochastic Processes in Quantum Theory and Statistical Physics, chapter "Stochastic processes in music and art"*, Springer Berlin / Heidelberg, 1982

[4] Ternstrom S, *Choir acoustics - an overview of scientific research published to date*, TMH-QPSR vol.43, 2002

[5] Ternstrom S, *Perceptual evaluation of voice scatter in unison and choir sounds*, STL-QPSR vol.32, 1991

[6] PRAAT, `http://www.fon.hum.uva.nl/praat`, retreived Jan 2011

[7] PADsynth algorithm with example implementations, http://zynaddsubfx.sourceforge.net/doc/PADsynth/PADsynth.htm, retreived Jan 2011
[8] BR404, http://www.kvraudio.com/get/3679.html, retreived Jan 2011
[9] discoDSP Discovery Pro, http://www.discodsp.com/discoverypro/, retreived Jan 2011
[10] KarmaFX Synth Modular, http://www.karmafx.net, retreived Jan 2011
[11] Kunquat, https://blueprints.launchpad.net/kunquat/+spec/kunquat-padsynth-generator, retreived Jan 2011
[12] Padpal, http://rekkerd.org/rock-hardbuns-updates-padpal-3/, retreived Jan 2011
[13] SynthMaker, http://synthmaker.co.uk/, retreived Jan 2011
[14] WhySynth, http://www.smbolton.com/whysynth.html, retreived Jan 2011
[15] YouTube, http://www.youtube.com/, retreived Jan 2011
[16] ZynAddSubFX, http://zynaddsubfx.sourceforge.net, retreived Jan 2011

"Petru Maior" University, Faculty of Science and Letters, Targu Mures, Romania
E-mail address: nascapaul@yahoo.com

STUDIA UNIV. BABEŞ–BOLYAI, INFORMATICA, Volume **LVI**, Number 1, 2011

OPTICAL ERATOSTHENES SIEVE

LIVIU ŞTIRB

Abstract. The sieve of Eratosthenes is a simple algorithm for finding all prime numbers up to a specified natural number. Despite its simplicity it is not widely used because of its complexity: $O(n\log n\log\log n)$ bit operations. An optical implementation for the sieve of Eratosthenes is described in this paper. The proposed device is highly parallel, by doing multiple verifications in the same time, thus providing a significant speedup compared to one implemented on a single core electronic computer.

1. Introduction

Solving problems with optical devices has received an increasing attention in the recent past. Several difficult problems have been attacked in this way:The Hamiltonian path [4, 5, 9, 7],The Travelling salesman problem [1, 9, 7],the subset sum [5, 3, 9, 7], 3-SAT [6], cryptography [10], factorization [2, 8] and some other NP-complete problems [5, 9].

Here we describe an optical implementation for the Eratosthenes sieve. The device is based on 2 parallel mirrors between which different beams of light are reflected. One of the mirrors also contains sensors which detect if the light has hit a certain region. If we send a beam at a certain angle, it will be reflected between those 2 mirrors multiple times. This process simulates the marking process of the Eratosthere Sieve. If we send multiple beams at different angles we can parallelize the marking process, thus providing a speedup compared to the sequential version of the algorithm.

The paper is structured as follows: Section 2 contains a short description of the problem and of the Eratosthenes sieve strategy. In the next section 3 we describe the proposed optical implementation. In section 4 we show the straights and weaknesses of the system. The final section 5 concludes our paper.

Received by the editors: February 14, 2011.

2010 *Mathematics Subject Classification.* 82D99, 11A41.

1998 *CR Categories and Descriptors.* F.2.1 [**Theory of Computation**]: Number-theoretic computations .

Key words and phrases. sieve of Eratosthenese, Optical computing.

2. Problem Description

A prime number is a number that can be divided only by 1 and itself. A basic algorithm for determining that a number is prime consists in trying all the numbers from 2 up to the square root of that number and checking if one of them divides it. If none is found we can say that the number is prime.

Sieve of Eratosthenes is a prime number strategy looking to find all prime numbers up to a number. The algorithm is quite simple and it looks like this: Having an array of natural numbers up to a limit, start with the number 2 and remove all its multiples(excluding it); after this step go to next unremoved number and remove all its multiples excluding itself; repeat this step until no next unremoved numbers exist. At the end, all the leftover (unremoved) numbers are prime.

The main difference between the proposed optical algorithm and Eratosthenes sieve is that the optical one will do the check for only one number. This limitation will be explained in section 3.4

3. The optical implementation

In this section we propose a solution for implementing the Eratosthene Sieve in an optical manner. We design a device in which we send beams of light which touch some regions at a regular distance.

The optical implementation is a combination between the sieve of Eratosthenes and the trial division. Trial division tests if an integer n can be divided by any integer greater than one but less than n. Having an array of natural numbers up to a limit, start with number 2 and remove all its multiples; after this go to next number and remove all its multiples including itself and repeat this step until the rounded square of the tested number. Eliminating a number that was not previously hit is a weakness imposed by the way the light is send.

Solving this problem on conventional computers requires an array. Each cell from that array corresponds to a natural number. At the beginning of the algorithm all cells are **true**. A loop iterates from 2 to the square root of the input number. All the cells that have an index which is a multiple of 2 are marked with **false**. After that, the loop selects the next cell in increasing order and marks all its multiples as **false**. At the end if the input number is not marked as **false** it means it is a prime number.

The pseudo code of the sieve of Eratosthenes used for the optical implementation looks like this:

v := an array of n elements(all **false**);
for i **from** 2 **to** $\lfloor\sqrt{n}\rfloor$
    if v[i] is **false**
        $k := i * 2$;

```
        while k < n do
            v[k]:= true;
            k := k+i;
        endwhile
    endif
endfor
```

at the end all the possitions that are false are prime numbers;

3.1. **Required physical components.** Basically we need the following components:

- 2 parallel mirrors,
- an array of sensors (photodiodes) placed on one of the mirrors,
- a prism,
- a source of light

In what follows we give details about each component.

3.2. **Mirrors and sensors.** The implementation is based on 2 parallels mirrors. One mirror is used just for reflections. The second mirror also reflects the light, but it also has a cell (sensor) which is capable of detecting if a beam of light hit it.

Each cell is placed at a regular interval. The distance between 2 consecutive cells is always the same. A cell has a simple function: it acts as a reflective mirror.The last cell is the number to be tested, and needs to be a photodiode in order to record if has been touched or not by light. The cell index is represented by a number. A number of cells equal to the input number to be tested is needed.

3.3. **Angles of reflection.** When a beam of light is sent under a known angle it will be reflected between the mirrors, touching them at a regular interval. The intervals where the mirror will be hit may be computed based on the distance between the mirrors and the angle at which the beam is sent.

An angle which will generate a step of n cells must be computed using the formula:

$$\sin x = \frac{d}{\sqrt{(\frac{s}{2})^2 + d^2}} \tag{1}$$

where:
x- is the angle at which the beam must be sent
d- is the distance between mirrors

s- is the step(represents distance). Note that s it is computed based on the distance between cells and the frequency for the current beam that touches the mirror. It may be computed with the formula:

s = distance between cells * frequency to hit

Example: if the distance between cells is 2 μ and a beam that hits all divisors of 5 is needed, s will be: 5*2 = 10μ

d and s must use the same unit of measurement

3.4. **Sending the light.** We have, inside the device, multiple rays at different angles. All angles encoding numbers from 2 to $sqrt(n)$ must be generated. Obtaining light at multiple angles can be done by using a prism. Prisms must also deflect a beam of light by a given angle.

Also we have discussed earlier (see the begining of section 3) about a limitation imposed by the way the light is sent, that has as a consequence the usage of the algorithm for a single number. This happens because we send all the light beams from the position 0.

If the avoidance of uncut numbers is desired, this will have unwanted consequences. One of them would be that the light will not be sent in a parallel manner because avoiding a cell should be done based on the previous results. Another limitation would be the need of physical movement of the light emitter.

3.5. **How the device works.** All beams are sent to the device through position 0. After that, the beams will traverse the device and will mark several positions (by hitting the corresponding sensors). At the end of the operation if the number we are interested in is not touched by any light, it is prime.

A basic example is shown in Figure 3.5. The example means to test the input number: 17. In order to test this number we have to send three beams of light. The numbers that must be tested starts from 2 and goes untill the truncated value from the square root of 17 which is 4. So we have to test the division by 2, 3 and 4.

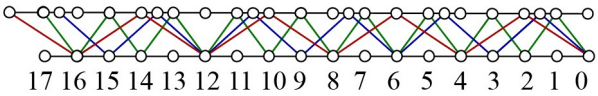


FIGURE 1. Optical implementation for the Eratosthenes Sieve. Bottom mirror contains optical detectors. 3 different beams have been sent (in the same time) from point 0 and we have depicted their path. Numbers which are multiples of 2 (green), 3 (blue) and 4 (red) are marked here.

Interference between two beams will not happen if we send short enough pulses. If we send two short beams from the same point, under different angles, the one that is reflected more rarely will travel faster because it has less distance to cover between mirrors until it gets out.

4. Strengths and weaknesses

The most important benefit of the algorithm is the speed generated by its massive parallelism. It allows us to send all the beams at the same time.

On a conventional computer, for the best parallelized algorithm, the speed up is in the best case equal to number of cores. On this optical system the concurrent speed-up is equal to the numbers of light beams. And there is the possibility to send all the beams at the same time. This means that the gained speed is equal to the square root of the number to be tested.

There are a number of weak points. The most important is that only small problems can be solved. That is because of some technical chalenges: reflection, absorbtion, etc.

5. Conclusions

We have exemplified how the very popular Eratosthene sieve's algorithm can be easily implemented with optics.

Future work will be focused on making a harware implemention and on increasing the parallelism of the system.

References

[1] Haist, T., Osten, W.: An Optical Solution For The Traveling Salesman Problem. Opt. Express, Vol. 15, 10473-10482 (2007)

[2] Kouichi Nitta, Nobuto Katsuta, Osamu Matoba: Improvement of a System for Prime Factorization Based on Optical Interferometer. LNCS 5882 Springer, pp. 124-129, 2009

[3] Md. Raqibul Hasan, M. Sohel Rahman, Computing a Solution for the Subset Sum Problem with a Light Based Device. LNCS 5882 Springer, pp. 70-76, 2009

[4] Oltean, M.: A light-based device for solving the Hamiltonian path problem. Unconventional Computing, Calude C. (et al.) (Eds), LNCS 4135, Springer-Verlag, 217-227 (2006)

[5] Oltean, M., Muntean, O.: Solving NP-Complete Problems with Delayed Signals: An Overview of Current Research Directions, in proceedings of 1st international workshop on Optical SuperComputing, LNCS 5172, pp. 115-128, Springer-Verlag, 2008

[6] Sama Goliaei, Saeed Jalili: An Optical Wavelength-Based Solution to the 3-SAT Problem. LNCS 5882 Springer, pp. 77-85. 2009

[7] Shaked, NT., Messika, S., Dolev, S., and Rosen, J.: Optical solution for bounded NP-complete problems, Applied Optics, Vol. 46, 711-724 (2007)

[8] Shamir, A., Tromer, E.: Factoring Large Numbers with the TWIRL Device, LNCS 2729 Springer-Verlag, pp1-26. 2003

[9] Shlomi Dolev, Hen Fitoussi: The Traveling Beams Optical Solutions for Bounded NP-Complete Problems. FUN 2007: 120-134
[10] Tobias Haist, Wolfgang Osten: Proposal for Secure Key Distribution Using Classical Optics. LNCS 5882 Springer, pp. 99-101, 2009

Department of Computer Science, Faculty of Mathematics and Computer Science, Babeş-Bolyai University, Kogălniceanu 1 Cluj-Napoca, 400084, Romania
E-mail address: liviu@cs.ubbcluj.ro

STUDIA UNIV. BABEŞ–BOLYAI, INFORMATICA, Volume **LVI**, Number 1, 2011

TOWARDS A MIDDLE LAYER FOR LARGE DISTRIBUTED DATABASES

COSTA CIPRIAN

Abstract. This paper addresses the lack of a cheap reusable software component for building large, database backed, distributed systems. Even after these systems have become mainstream through their use in companies like Google and Amazon, there are still difficult to approach for a small group of programmers with limited resources. We intend to have the JStabilizer framework fill this empty spot and we present in this paper some of the problems we would like to simplify solved and a short description of the solution.

1. Introduction

In many cases large distributed databases and services in general prefer to use cheap hardware and deal with the failures in the software, as failures will occur regardless of how expensive the hardware is. The main problem is that, even if the cost of hardware has decreased, the cost of the software is much higher since it is non trivial to write software that operates within some correctness definition even in the presence of failures in the system. We see decreasing the software cost as the next step in making large distributed system a commodity and we started implementing JStabilizer([3]) with this purpose in mind.

In the first part of this paper we present 4 use cases that we consider representative for the problem at hand and analyse the impact of the derived requirements and the differences between the approaches programmers could have when implementing such a distributed system.

In the second part we present an high level overview of JStabilizer as it is implemented now, explain some of the basic flows within the system and give some examples of how we envision the usage of this middle layer framework.

Received by the editors: March 19, 2011.

2010 *Mathematics Subject Classification.* 68P15, 68M14.

1998 *CR Categories and Descriptors.* C.2.4 [**Distributed Systems**]: Subtopic – *Distributed databases*;

Key words and phrases. distributed systems, middle layer.

2. Use cases

The main problem we are trying to solve is the introduction of a framework that would allow regular programmers to work with distributed databases without any strong synchronization requirement. Giving up on synchronization and commiting all transactions locally has the advantage of making the system available even in the case of partial failure (for example, a web server taking order from clients will still be available even if it becomes isolated from the rest of the network). The big problem with this approach is that the distributed database might (and will) become inconsistent.

In this first part of the presentation we will discuss what we consider to be 4 representative use cases where it is preferable to accept the inconsistencies and design the software to be resilient to those inconsistencies as opposed to creating a system with a smaller availability guarantee.

The first use case discusses the use of a distributed database on multiple independent agents. The problem bares some resemblance to DCSP - Distributed Constraint Satisfaction Problem ([11], [10]) but differentiate itself as it studies this in the context of a database and instead of finding the right assignment to some variables in a negotiation process the goal is to achieve consistency of the database.

The second use cases discusses the implementation of a distributed message board. The difference from the first use case is that instead of having conflicts between the updated values of some entity, the conflict is between the order in which replies appear in the discussion threads.

The third use case presents the well known problem of airline ticket reservation. In these commercial systems there is a strong requirement for availability because of the large income that is lost if, for example, the customers would not be able to purchase a certain item for a certain period of time. The difference here is in the conflict resolution process which requires human intervention.

The final use case discusses mobile environments which are known for their high volatility. A lot of research was done in this area ([1], [4], [6], [7]) but they usually address only specific use cases without trying to unify concepts with those found in server side distributed systems.

2.1. **Agent database synchronization.**

We considered and implemented the example of a restaurant where waiters are robots that are connected in arbitrary ways and without any single point of control. The example can be generalized to any type of independent agents that operate based on information from a database that is supposed to be synchronized.

In this case the database would contain the assignments of waiters to tables, so each waiter will know who is assign to which table. When the client

enters the restaurant and sits down at one table, some of the waiters will want to assign the table to themselves, thus introducing an inconsistency in the database (the consistency constraint is that only one waiter should be assigned to one table).

This is very similar to the self stabilization problem that was introduced by Disjkstra ([5]). The difference is that in this case there is no token that is passed from node to node in order to make sure that only one of them "moves".

It is important for each waiter to commit the transaction locally because it has to start working towards the goal of serving the table without requesting confirmation from any other peer (as this confirmation may never arrive because of some system failure and the table would remain without a waiter). It is obvious that the database may not become consistent as long as there are failures in the system, for example, two waiters that are completely cut off from the rest of the system might independently decide to attend a table and the system will not be able to correct this situation until the waiters come back online). In this case the programmer will just have to accept the fact those conflicts can appear and deal with them - the waiter will have to understand the fact that the table was already taken by another peer (ignoring this state would probably end up with a waiter constantly asking for the order from the client and constantly being turned down). As seen from this example a characteristic of these systems is that the cost of fixing an inconsistency is much smaller that the cost of making a system that is always consistent (in this case lots of redundant hardware and failover policies) or accepting temporary unavailability in the system (in this case nobody attending a table due, for example, to the inability of the system to form a majority because of partitioning or a failure in the control process).

In such a distributed system we would like the local database to always be consistent (the consistency predicate always holds on the local database) - in our case, each waiter will always have a database that always assigns a single waiter to a table. Not being able to rely on this constraint being true would make programming the system extremely complicated and costly. In other words we can deal with waiters that are wrong, but we can not deal with waiters that are inconsistent. The way to have this property and still be able to work towards the goal of having the database globally consistent is to create inverse operations for each operation that we would normally have. In this example the inverse operation would be deciding to no longer attend a table due to some sort of information that is received from peers. The key problem here is making sure that the system is doing progress - we do not want waiters to all decide that they are going to attend a table and then all

apply the reverse operation and decide not to attend. We will see later that ensuring progress is being made is not a trivial task.

A informal presentation of the algorithm we came up with is:

- Once a table is occupied several robots will evaluate a probabilistic function and decide if they should attend the table. Once the decision to attend is made, the robot will start working towards its implementation - actually attending the table.
- Each waiter that decided to serve a table will inform his peers of the fact that it is attending the table and also include the number of tables that are assign to him (according to its local database.)
- If a waiter receives the above message one of two things can happen:
  - No conflicts are detected - the information is saved to the local database and the message is forwarded to other peers.
  - A conflict is detected - in which case the conflict has to be resolved prior to saving the information to the local database and forwarding the message. The conflict resolution algorithm is the following:
    - The waiter with the smaller number of total tables wins, if the numbers are different.
    - The youngest waiter wins if the total number of tables are the same (we assume that no two waiters were born at exactly the same time)

Some observations on the proposed algorithm:

- A key assumption is that the system is asymmetric. Without this property a waiter would not be able to choose a winner of a conflict and assume that others will choose the same winner if presented with the same conflict
- The number of tables served by the waiters that are in conflict is not computed based on the information available in the local database, it is taken from the messages received from each of the waiters. As asymmetry, this is a requirement for the convergence of all the independent conflict resolutions performed by each peer.
- The scope of the algorithm is not to come up with a fair distribution of tables to waiters. The "fairness" is only used as a criteria for conflict resolution - could be replaced with any other criteria as long as each waiter resolves the same conflict in the same way.
- The algorithm is easily provable as correct if, for example, there are only two waiters that decide to serve at a table and the waiters are arranged in a circle. However, things get more complicated when the structure or the number of waiters is varied - each intermediary waiter

could resolve a different subset of the total set of conflicts in the system and, as such, introduce new conflicts.
- Conflict detection is an important part of the algorithm. Based on what is defined as a conflict, the algorithm might become stable (always converging to a distributed consistent state), or unstable (capable of remaining in an inconsistent state even without the presence of failures in the system).

2.2. **Distributed message board.** Another example that we considered when analysing the possible approaches to this problem is a distribute message board. The concept is similar to the one used in the USENET service - the message boards are stored in multiple places, comments are made on each server individually and then replicated from one server to another.

We can consider this a problem of ensuring the consistency of a virtual database that contains all the messages in all the servers in the network. The difference from the waiters problem is that in this case the conflicts are given by the order in which the messages appear, not by the content of the messages themselves. From a user perspective the requirement is that once a user has seen a specific order of messages, that order will not change, regardless of what other servers are doing. An example of a possible conflict is the following:

- The database starts with message A.
- On server S_1 a user posts a reply B.
- Before receiving the reply B from S_1, a user from S_2 will post another reply C to A.
- Servers S_1 and S_2 start propagating the actions and server S_k and S_j detect the conflicts at the same time.

Any of the sequences $A \to B \to C$ and $A \to C \to B$ are correct, but both S_k and S_j need to choose the same order, otherwise the servers will enter in an inconsistency that can not be solved while satisfying the user constraint of never seeing messages with different ordering.

From an implementation point of view we see the following differences when comparing this with the waiters example:

- Not all conflicts can be resolved right away, some might have to wait for other data to arrive at the node. For example S_1 creates message A, server S_2 creates B as a reply to A and server S_3 creates C as a reply to B. It is possible that server S_4 receives the message containing the creation of C before it receive the message for B and, as a consequence has to wait for B to arrive.
- There is a granularity difference between the objects on which we evaluate consistency (the entire thread) and the information that is exchanged between servers (the messages composing the actual thread).

While it is possible to always exchange the entire thread, there is no valid reason to do it from a conflict detection and resolution point of view.

2.3. **Airline ticket reservation.** Another use case that presents some different aspects of the problem is an airline ticket reservation system. Consider a database with all the flights in a specific area and distribute that database on several servers. Each time a client reserves a seat, the transaction commits locally, so there is a possibility that a seat is reserved twice - which is clearly an inconsistency. Solutions for reducing the number of conflicts can be found, like, for example, each server could provision some number of seats and, when interested on reserving more, it could ask from other servers to reduce their provision but, if the database is big enough and the infrastructure has a high enough probability for failure, we will end up with empty seats that can not be reserved.

The only solution is to allow distributed conflicts to exist (while maintaining each database locally consistent). Once a conflict is detected it is sent to a human operator which then negotiates with the involved parties in order to come up with a mutually accepted solution. An actual implementation was discussed by S.Bourne and Bruce Lindsay ([2]) in the context of the Sabre system where a tolerance of approximately one duplicate for every 1000 reservation was accepted and it proved to be cheaper to resolve those conflicts (by promoting some of the conflicts to first class or paying for an extra day of vacation for the customer) than to make sure that the system is always in a consistent state (which can only happen if the system has a lot of redundancy and state of the art equipment or if complete failures of the system are acceptable).

When comparing with the other solutions the following aspects stand out:

- The messages that are exchanged between servers do not contain enough information for the conflict to be resolved locally by each server, so some notion of special authority that is capable of resolving the conflicts is required.
- Even if a special authority exists and is a single point of failure (if it fails the database will never become consistent), it does not affect the reservation service - the clients are still able to reserve seats.
- There is a time constraint on the conflict resolution process. While in the previous cases it was accepted for the conflict to be detected only when the waiter reaches the client or there was no real penalty for posts to not propagate to all servers, in this case, the system is not going to be used if the probability for the conflicts to only be detected when the clients reach their seats is not statistically negligible. But

the probability of conflicts and upper limits on the expected levels of failures can be imposed and can be lowered through traditional techniques - the important thing is that the reservation system is always available to the client and no one goes to the competition because the web server throws an error message when a reservation is made.

From an middle layer implementation point of view the following aspects are important:

- Unlike the distributed message board, where some of the conflicts were not solvable when detected, in this case the resolution can not be made locally at all. Once a conflict is detected is has to be forwarded to a special authority which will resend a message with the resolution before that takes precedence in front of any other message from any other system.
- Depending on how the information is transmitted, it is possible for:
  - A conflict resolution to be received by a server before it detected the conflict
  - A conflict to be reported after it is resolved. This can happen if the conflict was previously detected by another server, resolved by the authority, but the current server found the conflict before receiving the resolution.
- If an incoming message resulted in a conflict then it makes no sense to forward the message as a message with the reservation that is in the local database was sent earlier and the current message would just be detected as a conflict by all servers we could send it to. This does not mean that a conflict can not be detected by several servers at once.

2.4. **Contact synchronization.** A final use case that adds something new to our system is an application for contact synchronization on mobile devices. We could consider a database with all the contacts on all the phones and have an application that creates relations between contacts from different phones. In such a setup an inconsistency could be defined as a difference between the values of two related contacts on different phones.

Following the algorithms described above, in this case, when someone updates a contact the change will propagate to connected phones. Given the nature of this application, the programmer could choose for example to allow the user to accept or deny an update, or the choice could be to always accept the change if there is no conflict. Another important aspect that the programmer might want to control is how the updates are forwarded. Should an update be forwarded even if it was not accepted? A reason why it could be forwarded is to allow all the network to receive the update regardless of the distance between nodes and then propagate the accept/deny option chosen by

the user in order to present other users the number of people in the network that accepted or denied the update.

Another aspect that is different is the conflict detection and resolution mechanism. The conflict detection could be more complicated than a simple value comparison. For example two people might update the phone number of the contact to the same number, but one will use the country code while the other would not, or there could be other syntactic differences that would not qualify as conflicts. Following this argument, the programmer might want to use a simple value comparison and then ask the user if any differences are actually conflicting. Conflict resolution could be done on each phone by each user or the application could forward the messages and let the users who generated the conflict agree on a value.

When comparing with the other examples some differences stand out:

- The conflict resolution strategy does not guarantee that the system stabilizes if failures are fixed an no other conflicts are generated. A simple example would be the classical live lock scenario: two users generate a conflict and, when each user detects the conflict, it can resolve the conflict by choosing the other version. This way the number of conflicts is not decreased and the database remains inconsistent.
- Failures could be permanent. In previous scenarios, the system were more or less under the control of the programmer or of parties that are directly interested in the correct functioning of the system as a whole. In this case the control is entirely in the users hand and some one could just loose their phone or uninstall the application
- Conflict resolution might require a special authority but, unlike the airline ticket reservation system, in this case it is not an external service but instead is a system formed from the users who generated the conflict in the first place. This behaviour introduces the following particularities in the conflict resolution mechanism:
  - It is possible that the special authority in a conflict resolution to have permanent failures. This is different from the airline ticket reservation scenario where we assumed that the conflict resolution authority only suffers transient failures.
  - If more than two users enter a conflicting state, the system should not require all of them to agree on a conflict resolution because this would allow a permanent failure in one of the parties to prevent other users from doing progress. Furthermore, the system should not require the users to actually agree on one version, it should just present each of the user with a list of conflicts they generated and ask them to choose a resolution.

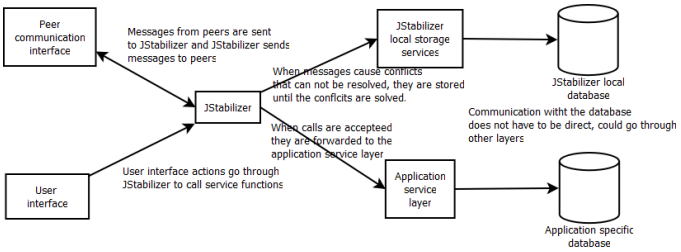


FIGURE 1. Overall system presentation.

While the value of such an application could be argued and is subject to personal preferences, the scenario and the requirements could come up in this or other applications of this sort and the needs could be addressed by the same middle layer used in the other scenarios.

2.5. **Wrap up.** From these examples we can extract some of the requirements for a middle layer that would allow them to be implemented at a smaller cost:

- Communication - while the actual means of communication are irrelevant and could be anything from message queues to shared memory systems, the programmer should be able to decide when the updates are propagated to peers.
- Conflict detection - some applications do not require a high degree of sophistication in this area but others might choose to implement more subtle strategies where a simple last modification timestamp or a versioning scheme might not be enough.
- Conflict resolution - we identified some of the conflict resolution scenarios (local, global special authority, per conflict special authority) but there could also be others, so this is a likely extension point. The middle layer must also be prepared from conflict resolution strategies that are not guaranteed to make progress towards the global consistency goal - there are systems that are still meaningful if such a goal is never reached.
- Special storage - some systems will not be able to apply the incoming messages on the local database immediately, so the middleware needs to have some sort of storage mechanism to ensure that these messages will be retrieved when required. The reason why these updates will have to wait is that the local database is always consistent, so, as long as a conflict is not resolved, the changes can not be applied.

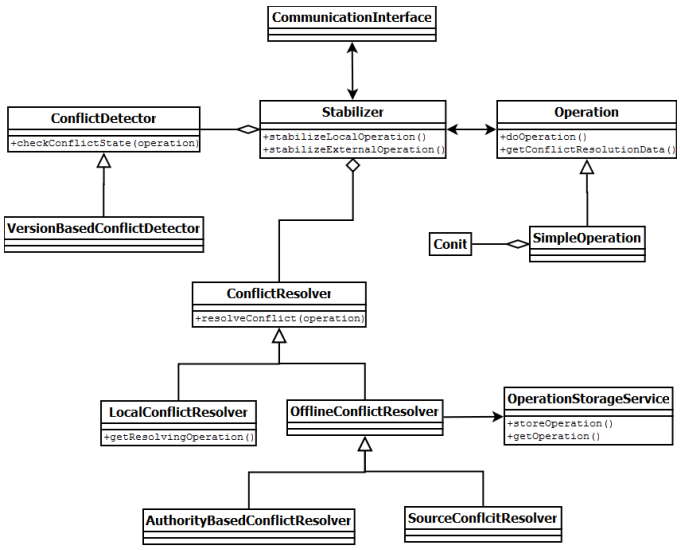


FIGURE 2. JStabilizer internal structure.

3. Implementation details

We started implementing JStabilizer as a middle layer framework that would lower the complexity of implementing the above described applications. The basic idea is to place JStabilizer in front of the service layer of the application, so all calls to the service layer will be registered with JStabilizer and, furthermore, JStabilizer will also receive information from the neighbour peers and send it to the service layer. We do not intend to make distribution transparent to the programmer for various reasons - a good presentation on how being transparent is forcing programmers to pay a high price and basically give up on the advantages of concurrency can be found in [9] (as a testimony to the controversy surrounding this issue, one of the authors went on to write such a transparent framework and explain the special circumstances when this is a good solution in [8]).

An overall presentation of how JStabilizer fits in the application is provided in figure 1. The programmer will have to make sure that all operations that work on items that are distributed will go through JStabilizer and program the services layer so that it supports update calls from both the direct users of the server and other servers in the network. Also the programmer will have to provide an implementation for the offline storage services required by JStabilizer (default implementation will be provided).

In figure 2 we have a brief presentation of the internal structure of JStabilizer. The main class is *Stabilizer* which is the gateway for all the operations that are executed and manages the temporary storage and the conflict resolution process. Each operation knows how to execute itself (call the proper service functions) but is passed to the stabilizer which will decide when and if it is actually executed.

The stabilizer will use on of the *ConflictDetector* and *ConflictResolver* classes in order to asses the state of each operation and proceed based on that.

A key concept in the implementation is the conit (CONsistency unIT) (introduced in [12] in a slightly different context) - defined as the smallest piece of information for which the study of consistency is significant. We use the concept of partial consistency - applications are more interested if a certain conit is consistent than if the entire database is consistent (for example the message board application can function correctly for threads that are consistent even if other threads in the database are not).

4. Conclusions and future work

We managed to present 4 representative use cases for the problem at hand that were used in testing the validity of the implementation and introduce the proper extension points. Most of the use cases were treated to some extend in other works but we present a unified solution for them. Also we inserted a brief presentation of the current JStabilizer implementation to get a sense of how the use cases have influenced it.

References

[1] Pauline M. Berry, Tomás Uribe, Neil Yorke-Smith, Cory Albright, Emma Bowring, Ken Conley, Kenneth Nitz, Jonathan P. Pearce, Bart Peintner, Shahin Saadati, and Milind Tambe. Conflict negotiation among personal calendar agents. *Proceedings of the fifth international joint conference on Autonomous agents and multiagent systems - AAMAS '06*, page 1467, 2006.

[2] S. Bourne. A conversation with Bruce Lindsay. *Queue*, (November), 2004.

[3] Costa Ciprian. JStabilizer code repository (http://code.google.com/p/jstabilizer/), 2009.

[4] S. Citro, J. McGovern, and C. Ryan. Conflict management for real-time collaborative editing in mobile replicated architectures. *Proceedings of the thirtieth Australasian conference on Computer science-Volume 62*, page 124, 2007.

[5] Edsger W. Dijkstra. Self-stabilizing systems in spite of distributed control. *Communications of the ACM*, 17(11):643–644, November 1974.

[6] C Sun, X Jia, Y Zhang, Y Yang, and D Chen. Achieving convergence, causality preservation, and intention preservation in real-time cooperative editing systems. *ACM Transactions on Computer Human Interaction*, 5(1):63–108, 1998.

[7] D. Sun, S. Xia, C. Sun, and D. Chen. Operational transformation for collaborative word processing. *Proceedings of the 2004 ACM conference on Computer supported cooperative work*, 6(3):446, 2004.
[8] J. Waldo. Scaling in games \& virtual worlds. *Queue*, 6(7):10–16, 2008.
[9] J. Waldo, G. Wyant, A. Wollrath, and S. Kendall. A note on distributed computing. *Lecture Notes in Computer Science*, pages 49–64, 1997.
[10] M. Yokoo, E.H. Durfee, T. Ishida, and K. Kuwabara. The distributed constraint satisfaction problem: Formalization and algorithms. *IEEE Transactions on Knowledge and Data Engineering*, 10(5):673–685, 1998.
[11] Makoto Yokoo and Katsutoshi Hirayama. Algorithms for Distributed Constraint Satisfaction:A Review. *Autonomous Agents and Multi-Agent Systems*, 3(2), 2000.
[12] H Yu and A Vahdat. Design and evaluation of a continuous consistency model for replicated services. *of the Fourth Symposium on Operating Systems Design*, 2000.

Babes-Bolyai University, Department of Computer Science, Cluj-Napoca, Romania
E-mail address: costa@cs.ubbcluj.ro

STUDIA UNIV. BABEŞ–BOLYAI, INFORMATICA, Volume **LVI**, Number 1, 2011

CONCEPTUAL MODELING EVOLUTION. A FORMAL APPROACH

MARIA-CAMELIA CHISĂLIŢĂ-CREŢU

Abstract. The aim of this paper is to investigate the possible types of conceptual variability and to propose shifting strategies, like refactoring, forward conceptual abstraction, and conceptual specialization to switch between conceptual models. A biological evolution-based model is proposed to describe the changes within the structure of the studied models. The transformations that highlight the conceptual modeling variability in ontogenic and phylogenic processes are formalized.

1. Introduction

Software systems continually change as they evolve to reflect new requirements, but their internal structure tends to decay. Refactoring is a commonly accepted technique to improve the structure of object oriented software. Its aim is to reverse the decaying process in software quality by applying a series of small and behaviour-preserving transformations, each improving a certain aspect of the system [9]. The *variability* [19] in the context of conceptual modeling means the possibility to build distinct and still correct conceptual models for the same set of requirements from the real world (Universe of Discourse, UoD) problems. Such conceptual model is called *variant*. A non-exhaustive framework of three types of variability was proposed, based on a literature survey and empirical evidence [19].

Within conceptual modeling variability, refactoring has proved to be a feasible technique to switch between variants. In order to emphasize refactoring transformations, the artifact may be represented as graph [14]. As refactoring was initially applied at the implementation level, conceptual models became a potential target for application of refactorings.

Research within conceptual modeling reveals the possibility to integrate the refactoring process in the analysis development phase too. A biological

Received by the editors: March 20, 2011.

2010 *Mathematics Subject Classification.* 68N30, 92B10.

1998 *CR Categories and Descriptors.* D.2.1 [**Reusable Software**]: – *Reusable Software*; I.6.5 [**Model Development**]: – *Modeling Methodologies*.

Key words and phrases. conceptual modeling, refactoring, biological evolution.

evolution model is proposed in order to cope with different types of variability that were identified. Specific refactorings are suggested to shift between *ontogenic* conceptual models, while forward conceptual abstraction and conceptual specialization are advanced to achieve *phylogenic* conceptual models.

The rest of the paper is organized as follows. Section 2 investigates the three types of conceptual modeling variability identified in the literature. Three types of transformations are advanced in order to cope with the variability among variants. A biological evolution-based model is proposed in Section 3, while Section 3.2 formalizes the identified transformations within the three types of conceptual modeling variability studied as ontogenic and phylogenic processes. Section 4 contains some conclusions of the paper and suggestions of future work.

2. Conceptual Modeling Variability Types

A non-exhaustive framework of three types of variability was proposed, based on a literature survey and empirical evidence: construct, vertical abstraction, and horizontal abstraction variability [19].

2.1. **Construct Variability.**

Definition 2.1. *Construct Variability ([19])*
Construct variability represents the possibility of modeling concepts in the UoD using different constructs in the same modeling language.

Within construct variability, the concepts within UoD have the same semantics in all variants. They are represented by a class (entity), an attribute, a relationship.

2.1.1. *Types of Equivalent Construct Variants.* There are many types of construct variability. In [3] the possible refactorings that may be applied to the conceptual modeling variability are studied. In Figure 1 a frequently case of construct variability within object-oriented analysis and design is presented. The `Price` concept may be both modeled as a class (*cmA* variant) with a single attribute `amount` or as an attribute of type integer (*cmB* variant). The semantic definition for the `price` and the `Product` are identical in both variants, though different language constructs to represent it are used, i.e., an attribute (*cmB* variant) instead of a concept (*cmA* variant).

Another type of construct variability is similar to normalization of a database definition, by removing all redundant data elements from the class definitions. Figure 2 emphasizes such a normalization in the *cmA* variant, where the product `value` aspect is modeled as an attribute. In the *cmB* variant, the product `value` is modeled as a method, which multiplies the *quantity* by the *price* to obtain the correct `value` in Figure 2. A motivation to consider the

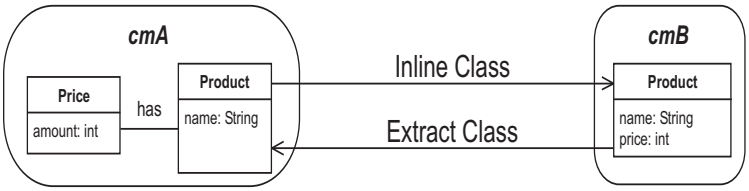


FIGURE 1. Construct variability for the concept `Price`. *cmA* variant is an entity-based model; *cmB* variant is an attribute-based model.

value a method is because it represents a calculation done predominantly at design level. Within the analysis phase, the semantic definition of the *value* is given by the same formula within different conceptual models.

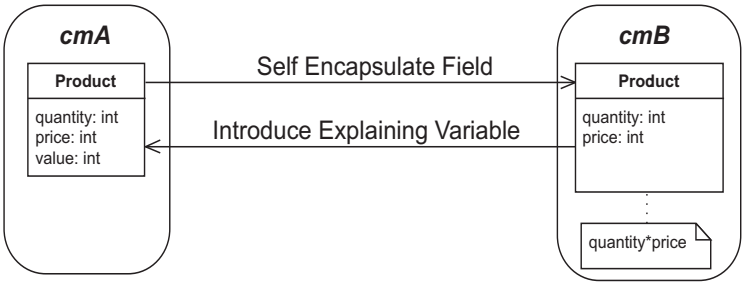


FIGURE 2. Construct variability for the attribute `value`. *cmA* variant is an attribute-based model; *cmB* variant is a method-based model.

A third type of construct variability is represented by the multiple types definition. It consists of introducing or removing specific codes for all the types that are a direct specialization of a generic type. Figure 3 shows a general `Product` that is refined to another ones, more specific: `BOY` and `GIRL`. The *cmA* variant defines specific codes for each concrete product as for *boy* or *girl*, adding a short description for the derived types. On the other hand, the *cmB* variant creates different classes for *boy* or *girl* `products`, providing flexibility for improvements directed to a specific type. Therefore, the definition for the specialized types has not the same semantics for both variants.

2.1.2. *Suggested Refactorings.* In order to switch between variants, there are several types of refactorings that may be applied.

Refactoring a class (entity) to a set of attributes (properties)

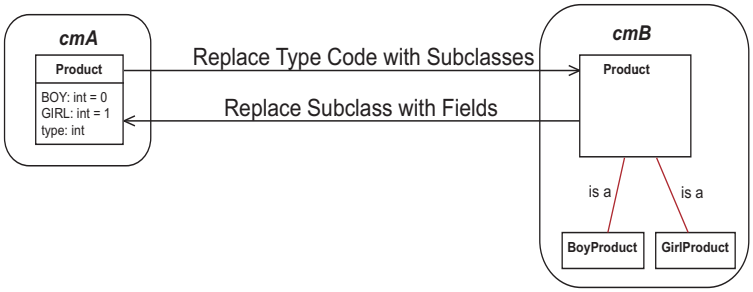


FIGURE 3. Construct variability for multiple types. *cmA* variant is an attribute-based model; *cmB* variant is a entity-based model.

The first recommended transformation is the *InlineClass* refactoring [9]. It is used as a weight reducer since it diminishes the number of classes (entities) by redistributing the responsabilities among the remaining classes (entities). The aspect of moving the attributes and methods to other classes is realized through refactorings like *MoveMethod* or *MoveField*. Figure 1 illustrates how the `price` attribute in the `Product` class is replaced by the attribute contained within the `Price` class, i.e., the `amount` attribute. In order to reverse the effect of the *InlineClass* refactoring, the *ExtractClass* refactoring may be applied to switch between variants. As an immediate effect of the latter application is a raise of the abstraction level, while the overall weight of the UoD increases due to the new class (entity) addition.

Refactoring an attribute to a method

There are many cases where an attribute may be computed from other existing attributes, while the attribute continues to exist. This results in redundancy, a particular case of bad smell [9]. The initialization of the attribute may consist of a formula of which the interpretation gives the correct value of the attribute. Subsequently, the attribute is updated by the corresponding formula. Thus, it may be extracted to a method and the calls to this method will replace the access to the redundant attribute. The latter one being no longer referenced, will be removed. In order to remove this redundancy problem, the *SelfEncapsulateField* refactoring is suggested by Figure 2, that will replace all accesses and updates to the attribute with calls to a newly introduced accessor (*getter*) and modifier (*setter*) method. The *IntroduceExplainingVariable* refactoring allows to return to initial variant by adding an attribute that will be initialized with the corresponding expression.

Refactoring type codes to a set of derived classes

Type codes may be used to model different specializations of the same class. This situation is usually indicated by the presence of `case`-like conditional statements, i.e., `switch`, `if-then-else` constructs. They test the value of the type code and then execute the appropriate code, depending on the value of the type code. A variant to this conceptual model is to expand the class into a class hierarchy in order to emphasize specific types of the base class.

Different refactorings are recommended to be used subsequently. Conditionals that affect the behavior need to be transformed by the *ReplaceConditionalwithPolymorphism* refactoring, that allows to use polymorphism to handle the different behavior in the inherited classes. In order to switch the type codes within a context where the behaviour is not affected the *ReplaceTypeCodewithSubclass* refactoring may be applied. In each cases, the type codes will be replaced with a subclass for each distinct one. Furthermore, there are cases where some features that are relevant only to objects with certain type codes.

Creating such a class hierarchy through this refactoring, then the *PushDownMethod* and the *PushDownField* refactorings may be applied to clarify to which subclass these features are relevant. An important advantage of this switch between variants is the possibility to move the particular behaviour from a client of a class to the class itself. This refactoring ensures a large flexibility of the variant within a continuous changing UoD, through polymorphism.

The reverse process allows to transform subclasses into attributes within a single class, following the *ReplaceSubclasswithFields* refactoring. Figure 3 depicts the way the type codes are changed by the appropriate refactorings. This refactoring situation represents a special case of forward conceptual abstraction. There are cases where the type code may or may not affect the behavior.

2.1.3. *Core Ideas.* The three representative examples demonstrate that refactoring between construct variants is feasible. The effort required to switch between the variants is reduced by the application of a limited number of small refactorings.

There are few, rather limited differences between the variants within the construct variability. Though, the literature retained some work that suggests the evolvability aspect of conceptual models within this type of variability. This would be the case of the third example discussed here, where the flexibility to improvements of the *cmA* variant is reduced. In [1] it is claimed that an entity should be preferred over an attribute if it is likely that the modeled concept in the UoD will take benefit of additional properties in the future.

Though, this claim suggests that is useful to be able to switch between these two variants.

This type of variability is exploited in the shift from object-oriented analysis to design. As a consequence, it is expected that *construct variability* had been already used refactoring in current modeling activities. Furthermore, other relevant results in refactoring conceptual models show that is unlikely that hard obstacles for this transformations between construct variants will be found [6, 18, 7].

2.2. **Vertical Abstraction Variability.**

Definition 2.2. *Vertical Abstraction Variability ([19])*
Vertical abstraction variability refers to the possibility of modeling concepts in the UoD in a more or less generic (abstract) way.

2.2.1. *Types of Vertical Abstract Variants.* There are two ways to navigate over the vertical abstraction variability. The first one refers to the possibility to switch from a general conceptual model to a concrete model, while the other one increases the abstraction level by removing concrete aspects, or by adding various parameters. In [2] refactoring categories needed to switch between models are identified and described.

An example of an abstract vertical variability that may be navigated in both ways, from a generic to a specific conceptual model and vice versa is presented for the `Loan` concept. It cannot be considered like in the construct variability, because its definitions are different within studied variants. The *cmA* variant illustrated by Figure 4 consists of a concrete conceptual model, where a `Loan` is associated with the `Client` to whom it was given to.

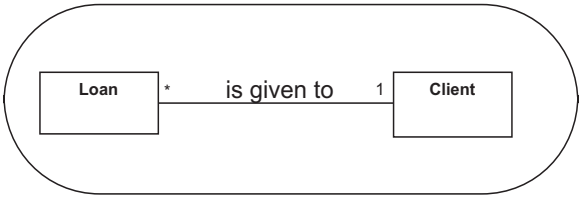


FIGURE 4. *cmA* variant for the concept `Loan`: a *concrete model* within a vertical abstraction.

In the Figure 5, the *cmB* variant defines the `Loan` given to a `Client` when a specific `Action` is achieved, e.g., the client meets some eligibility criteria.

Figure 6 depicts the *cmC* variant, where the `Loan` is given to `Client`, that may be an `Institution` or a `Person`. Moreover, the `Loan` has a type and it is given to the `Client` when some `Action` is fulfilled.

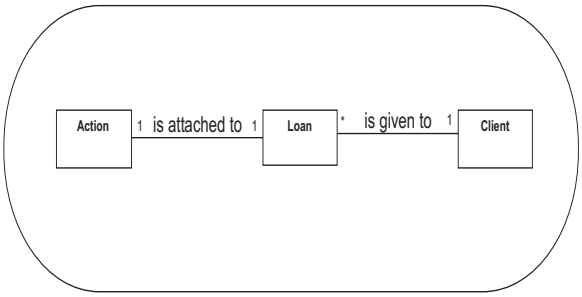


FIGURE 5. *cmB* variant for the concept `Loan`: a *general model* within a vertical abstraction.

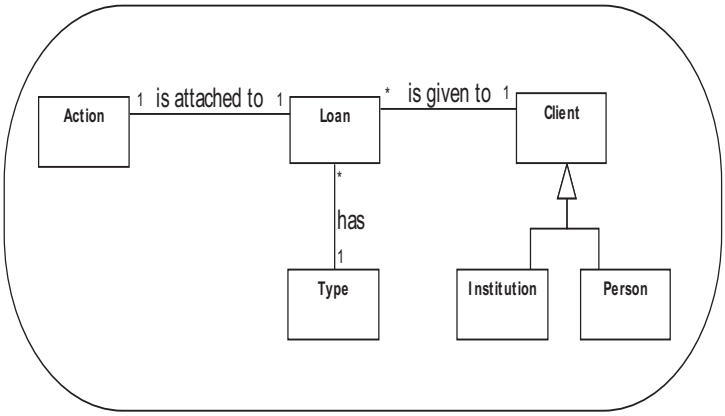


FIGURE 6. *cmC* variant for the concept `Loan`: a *very general model* within a vertical abstraction.

The three variants may be navigated from the most concrete, to the most general and conversely. The *cmC* variant is the most general model, where the `Loan` definition is available for a wide number of situations compared to the first two variants.

2.2.2. *Suggested Refactorings.* In order to switch between the presented variants, the two ways to navigate over the vertical abstraction and instantiation variability were studied and appropriate solutions were provided.

Transforming to a more generic variant

For the `Loan` concept, switching from the most concrete variant (*cmA* variant) to the most general (*cmC* variant) means to increase its flexibility

degree. First, by introducing some `Action` concept that allows to offer a `Loan` to a `Client` (*cmB* variant). The abstraction level is increased by switching from the *cmB* variant to the *cmC* variant, by adding a `Type` to the `Loan` and differentiating a `Client` as `Institution` or `Person`. In order to do that, a more detailed analysis is needed. The forward conceptual abstraction follows to discover new concepts and new associations that make the model more flexible. The refactoring techniques that may be applied are not immediately identified.

Transforming to a more specific variant

In order to obtain the *cmA* variant starting from the *cmC* variant, the flexibility level of the more generic variant has to be reduced by removing all unrequired concepts, associations, and attributes. The refactorings that can be applied in this situation are those specific to inheritance or generality category [9]. They may consists of refactorings like: *PullUpField, PullUpMethod, PushDownMethod, PushDownField, ExtractSubclass, ExtractSuperclass,* or *ExtractInterface.* It is important to underline that from this large refactoring category only the refactorings that work on concepts, associations, or attributes will be applied. This transformations allow to switch from a flexible model to a thin, clear, concrete model, by removing the superfluous information.

2.2.3. *Core Ideas.* The software *genericity* is defined by Parnas [16] as the possibility to use it "without change, in a variety of situations". In [11] *abstraction* is defined as "a view of an object that focuses on the information relevant to a particular focus and ignores the remainder of the information".

In order to identify and implement new concepts and specialized behaviour that transform the concrete models to more generic ones, the forward conceptual abstraction process is needed. Refactoring techniques have limited usage in this navigation way of the vertical abstraction variability. The new variants have the advantage of a raised adaptability and flexibility.

The shift from a more generic to a more specific conceptual model consists of applying refactoring techniques that remove the redundant modeling elements and achieve a lighter variant of the initial model. This type of variability was observed in the process of simplifying the design of the over engineered systems [16, 10, 8, 6]. Changes to the system are made more easily if the conceptual model is more general and consequently, more difficult if the conceptual model is too simple or too concrete.

2.3. **Horizontal Abstraction Variability.**

Definition 2.3. *Horizontal Abstraction Variability ([19])*

Horizontal abstraction variability refers to the possibility of modeling concepts in the UoD based on different properties.

2.3.1. *Types of Horizontal Abstract Variants.* The horizontal abstraction is emphasized within a particular UoD that contains the concepts of an academic management system, like students and teachers. A *student* has a specialty that follows, while a *teacher* has a didactic position. Each of them has a certain *civil status*. The solutions proposed to achieve horizontal variability are presented in [4].

A first direction within the research is represented by the one depicted in Figure 7 where the *person type*, i.e., *student* or *teacher*, is emphasized. The possibility to make visible the person types means to add it as a *primary dimension* [19], that allows to isolate all instances of a certain type of person. In the *cmA* variant, the person type instances are separated in two categories, defined as the `Student` or `Teacher` concepts. `Civil Status` property is shared by both `Student` and `Teacher` concepts. In [19] such a property forms a *secondary dimension* in the concept modeling. Its instances, like *married person* or *single person* are scattered (made not visible) over all instances of the `Student` and `Teacher` types.

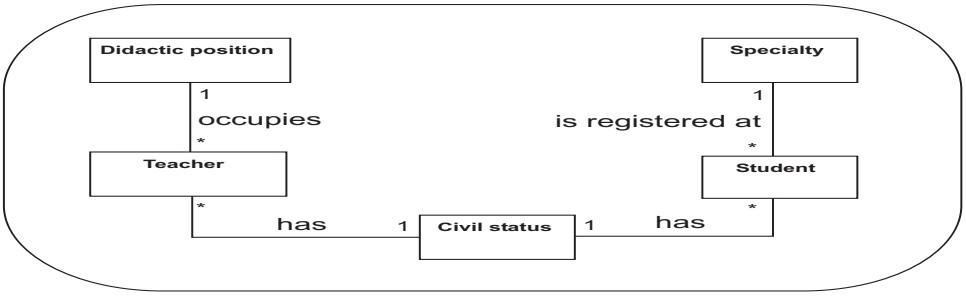


FIGURE 7. *cmA* variant : The `Person` types form a primary dimension, through the `Student` and `Teacher` concepts.

The second approach consists of highlighting the `Civil Status` property by isolating it as a primary dimension. Therefore, instances of persons are divided in two categories: `Married person` and `Single person`. The person type (*student* or *teacher*) remains as a secondary dimension, being spread over all instances of type `Married Person` and `Single Person`. Figure 8 presents the *cmB* variant where the person type is not visible, but shared between different instances of `Civil Status` types.

In order to shift between the two variants an intermediate *cmC* variant, presented by Figure 9, needs to be build. Therefore, the primary dimension of the person types from the first approach and the primary dimension of the

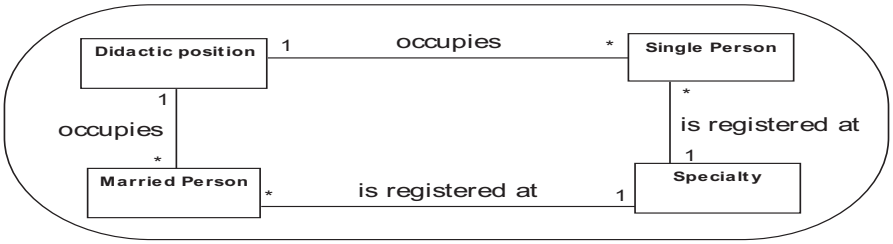


FIGURE 8. *cmB* variant : The `Civil Status` types form a primary dimension, through the `Married Person` and `Single Person` concepts.

civil status types from the second research direction are used. This means that both are isolated and visible.

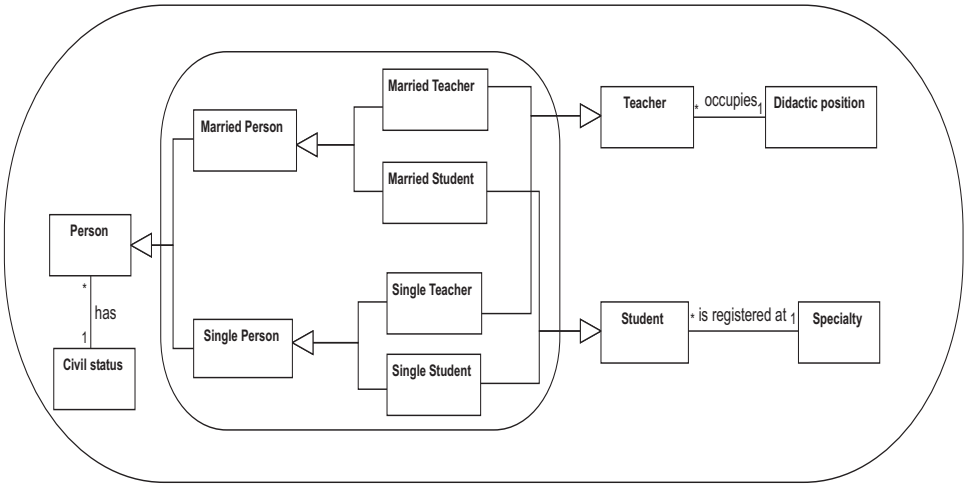


FIGURE 9. *cmC* variant : Both `Person` and `Civil Status` types are primary dimensions.

2.3.2. *Suggested Refactorings.* There are situations where someone may want to switch between variants developed within horizontal abstraction variability. The shift between the *cmA* variant and the *cmB* variant may be done using the *cmC* variant as an intermediate variant. In order to achieve it a two phases process has to be implemented:

- *Step 1:* establish an equivalency relationship between the two dimensions, by transforming the *cmA* variant to the *cmC* variant;

- *Step 2:* allow to keep the requested primary dimension only.

Refactoring to equivalent dimensions (*Step 1*)

In order to accomplish the equivalency between the *cmA* variant and the *cmC* variant, each entity type from the primary dimension of the *cmA* variant will receive the specialization from the primary dimension of the *cmB* variant. This means that a `Student` will become `Married Student` or `Single Student`, while a `Teacher` will be `Married Teacher` or `Single Teacher`. The total number of entity types is computed as cartesian product between the **n**umber **o**f entity types for the **P**rimary **D**imension (**noPD**) within the *cmA* variant (*student*, *teacher*) $noPD_{cmA} = 2$, and the *cmB* variant (*married*, *single*) $noPD_{cmB} = 2$. Therefore, as it is shown in Figure 9 the number of specialized types identified within the *cmC* variant is $noPD_{cmC} = noPD_{cmA} \times noPD_{cmB} = 2 \times 2 = 4$. Transformations required to switch between the *cmA* variant and the *cmC* variant include the following aspects:

(1) *add new classes to specialize.* For each entity type of the primary dimension within the *cmA* variant, i.e., `Student` and `Teacher`, new subclasses that emphasize the primary dimension within the *cmB* variant, i.e., `Married` and `Single`, are added. This means the new variant will be enhanced with the following classes (entities): `Married Student`, `Single Student`, `Married Teacher`, and `Single Teacher`.
(2) *responsibility reassignment.* Specialization for the new added classes (entities) is made by moving or pushing down data (and behaviour) from the initial primary dimension to the new subclasses, using refactorings like: *PushDownMethod* and *PushDownField* [9]. This new variant introduces redundancies, like where a `Single Student` has already a typed property `Civil Status` as `Single`.
(3) *add new class to generalize.* Generalization for the new created subclasses is added from the primary dimension of the *cmB* variant.

Refactoring to a primary dimension (*Step 2*)

To obtain the simplified *cmB* variant starting from the *cmC* variant, the transformations previously applied have to be semantically reversed. This means that primary dimension from the *cmB* variant has to be emphasized by collapsing the existing subclasses in the *cmC* variant. Thus, the entity types related to a `Person` (`Student` and `Teacher`) will be spread within the model through the new primary dimension `Civil Status` (`Married` and `Single`) of the *cmB* variant. The suggested refactorings to obtain a lighter model, concentrated on a specific primary dimension, include the following transformations that collapse the hierarchy:

(1) *responsibility reassignment.* Generalization is realized by moving or pulling up data (and behaviour) from one initial primary dimension to a new class, using refactorings like: *PullUpMethod* and *PullUpField*. This transformations prepare the model to safely remove the superfluous information from the *cmC* variant.
(2) *safely removal of the specialized classes.* Only the primary dimension of the *cmB* variant, represented by the typed property `Civil Status`, through `Married Person` and `Single Person` are kept. Specialized classes related to the primary dimension of the *cmA* variant, as `Married Student`, `Single Student`, `Married Teacher`, and `Single Teacher` may be safely removed from the new model.

Refactorings applied to reduce the model to a single primary dimension, does not ensure the equivalency between the *cmB* variant and the *cmA* variant. There are two possibilities to check that a `Person` is a `Student` or a `Teacher`.

- *implicit relationship usage.* There is an implicit relationship between
  - `Student` and `Specialty` – *only* the `Student` is registered at a faculty `Specialty`;
  - `Teacher` and `Didactic Position` – *only* the `Teacher` occupies a `Didactic Position`;
- *type variable usage.* A new type variable may be introduced to distinguish between the two `Person` types (`Student` and `Teacher`).

2.3.3. *Core Ideas.* Similar to the vertical abstraction variability, in horizontal abstraction variability the concepts in the UoD are modeled using different semantic definitions. But, within the former one, the difference between variants concerns the different levels of generality, while in latter one, the difference between variants bears upon concepts that are modeled based on different properties.

In the horizontal abstraction variability, the properties may be or not *visible* and *isolated*. They are classified as *primary dimension properties* (that can be visible and isolated from others) and *secondary dimension properties* (that are not visible and cannot be isolated from others)[19].

The literature records claims that, as vertical abstraction variability, the horizontal abstraction variability affects evolvability [17]. Within the latter one, the aspect responsible for the evolvability disadvantages is represented by the information hiding highlighted within the secondary dimension.

In order to refactor from the *cmA* variant to the *cmB* variant a new intermediate with equivalent dimensions *cmC* variant is used. The transformation process from a variant with equivalent dimensions to a regular variant implies more resources than the previous step, when there were many constraints and relationships that became implicit in the resulting model.

The transformations number applied through refactoring when switching between variants depends on the number of entity types in the primary dimensions of both variants ($noPD_{cmC} = noPD_{cmA} \times noPD_{cmB}$) and the extent to which these types are used. Thus, it is expected that some range may be taken over, above which the refactoring cost weights to much over its advantages.

Refactoring literature does not record real world application of horizontal abstraction variability, though active research has been developed [15].

3. A Model for the Evolution in Conceptual Modeling Variability

Variability within conceptual modeling outlines an evolutionary process among different models of a specific variability type. This process is similar to the biological evolutionary process presented by Maturana and Varela in [13]. According to them, changes are determined by the structure of an organism and a perturbation. A perturbation itself does not determine how the organism evolves, but it triggers the organism to change its structure. The evolved organism with its new structure affects the outer environment and produces another perturbation. This iterative process of the interaction between the organisms structure and the environment through a perturbation is a driving force of evolution [13].

For a software product, customers may require new functionalities to be implemented. This results in changes that serve as perturbation in the software product evolution. In order to achieve variability within conceptual modeling, changes provided by refactoring, forward conceptual abstraction, conceptual specialization or other evolutionary changes have to be applied. There are two types of evolution in biology: *phylogeny* and *ontogeny* [13]. The former refers to the evolution as species while the latter refers to the evolution of individual living beings.

Two types of evolutions have been identified for the conceptual modeling variability within our research. A first type of evolution is similar to an *ontogenic process* where an individual living being grows. This corresponds to small changes that does not substantially affect the overall conceptual model. Major modifications on the conceptual models that have effect on the entire future development represent a second type of evolution. This is represented by a *phylogenic process* that fundamentally affects every development stage forward. Within a phylogenic evolution, the before and after conceptual models belong to different development stages and have different development approaches.

For the already studied conceptual modeling variability with its three different types, i.e., construct, vertical abstraction, and horizontal abstraction,

an evolution model may be developed. Figure 10, Figure 11, and Figure 12 illustrate how the two ontogenic and phylogenic biological evolution may be modeled in the conceptual modeling variability context.

3.1. Conceptual Modeling Variability as Biological Evolution.

Evolution in Construct Variability

Figure 10(a) depicts the ontogenic evolution for conceptual models that are perturbated by small changes that does not fundamentally affect the developed model. These changes correspond to switches between *attributes and entities* ($cm_{i,j} \rightarrow cm_{i+1,j}$, $cm_{i+1,j} \rightarrow cm_{i,j}$) or *attributes and methods* ($cm_{i+1,j} \rightarrow cm_{i+2,j}$, $cm_{i+2,j} \rightarrow cm_{i+1,j}$) approaches (see Section 2.1.2). They consists of refactorings applied to the conceptual models such that their overall organization remains fundamentally unchanged.

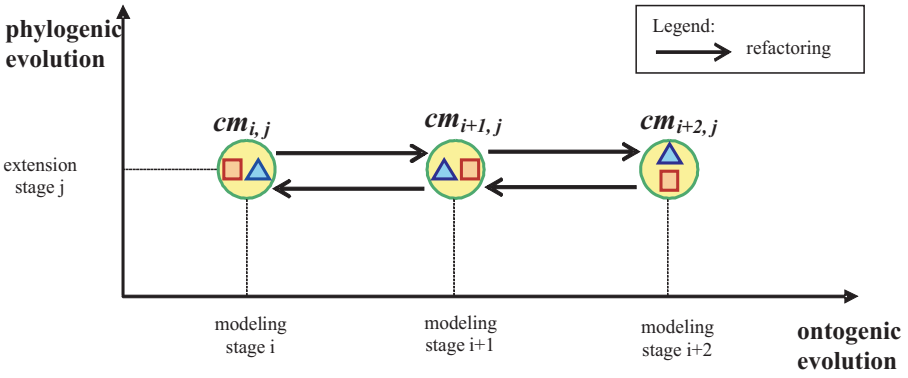
The *multiple types definition* construct variability presented in Section 2.1, outlines a phylogenic evolution by the addition of new types within the conceptual model. Figure 10(b) shows that changes within the model are reflected by a forward conceptual abstraction process, denoted by $- \rightarrow$, for *addition* of new types ($cm_{i,j} - \rightarrow cm_{i+1,j+1}$). The existing types *removal* and their replacement by type codes is achieved by conceptual specialization, relation denoted by $-\bullet$, where $cm_{i+1,j+1} - \bullet\, cm_{i,j}$. They represent small refactorings that decrease the complexity. They may be interpreted as a special case of forward conceptual abstraction inducing a different generality level between the source and the target models.

Evolution in Vertical Abstraction Variability

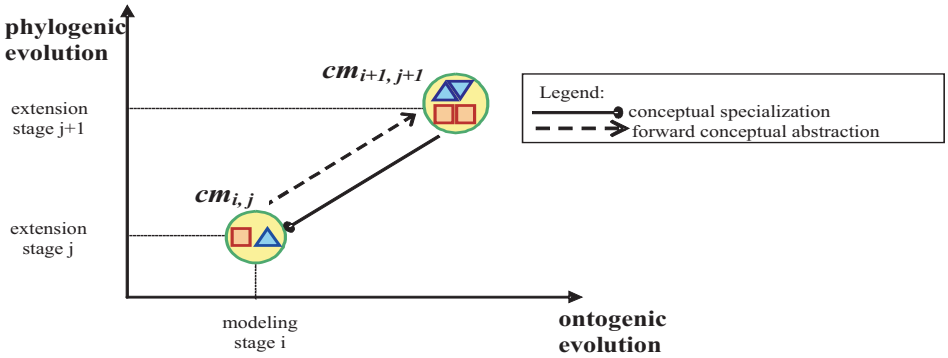
Reducing the abstraction level for a conceptual model means to remove superfluous information in order to shape a more concrete conceptual model. Figure 11 suggests that shifting from a more general to a more concrete model ($cm_{i+1,j+2} - \bullet\, cm_{i,j}, cm_{i+3,j+3} - \bullet\, cm_{i+1,j+2}, cm_{i+1,j+2} - \bullet\, cm_{i+2,j+1}$) results in changes applied to a single model, i.e., the more concrete one, which corresponds to a conceptual specialization, by reducing the model generality. In this way, the simplifying process consists of refactorings that remove the irrelevant information in the target model. On the contrary, raising the abstraction level requires additional information gathered by forward conceptual abstraction. Figure 11 shows that moving to a generic model new extension stages are added ($cm_{i,j} - \rightarrow cm_{i+1,j+2}$, $cm_{i+1,j+2} - \rightarrow cm_{i+3,j+3}$, $cm_{i+2,j+1} - \rightarrow cm_{i+1,j+2}$).

Evolution in Horizontal Abstraction Variability

Switching between models developed under horizontal abstraction variability may be done using an intermediate variant. The phylogenic evolution within this type of variability appears at the first shifting step. Figure 12 shows that addition of a new visibility dimension to the model, which drives the complexity of the development process to a higher level through forward conceptual



(a) Ontogenic evolution in construct variability, through refactoring



(b) Phylogenic evolution in construct variability, through forward conceptual abstraction and conceptual specialization

FIGURE 10. Construct variability as ontogenic and phylogenic evolution processes, through refactoring, forward conceptual abstraction, and conceptual specialization

abstraction ($cm_{i,j} - \rightarrow cm_{i+1,j+1}$, $cm_{i+2,j} - \rightarrow cm_{i+1,j+1}$). In order to reduce the number of visible dimensions, refactoring may be applied to a model within a conceptual specialization ($cm_{i+1,j+1} - \bullet cm_{i,j}$, $cm_{i+1,j+1} - \bullet cm_{i+2,j}$). This process is depicted by Figure 12 where the intermediate model $cm_{i+1,j+1}$ is used to achieve the specialization for a single conceptual model.

3.2. **Formal Approach.** In order to formalize the conceptual modeling variability as an ontogenic and phylogenic evolution some definitions are needed.

Definition 3.1. *Conceptual Model ([5])*

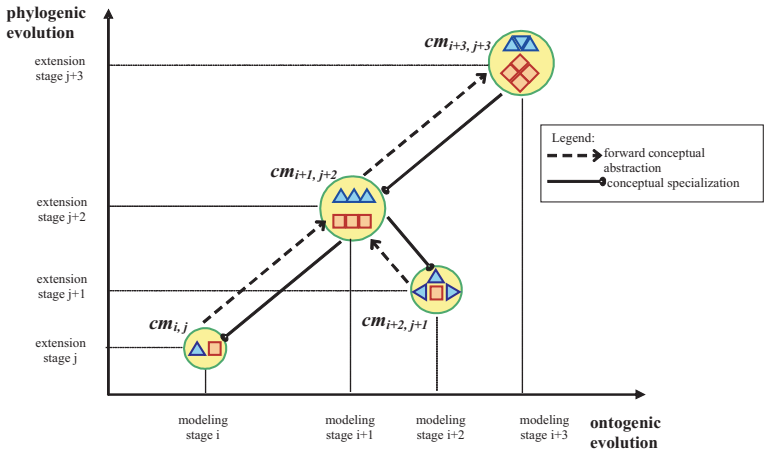


FIGURE 11. Vertical abstraction as phylogenic evolution through forward conceptual abstraction and conceptual specialization

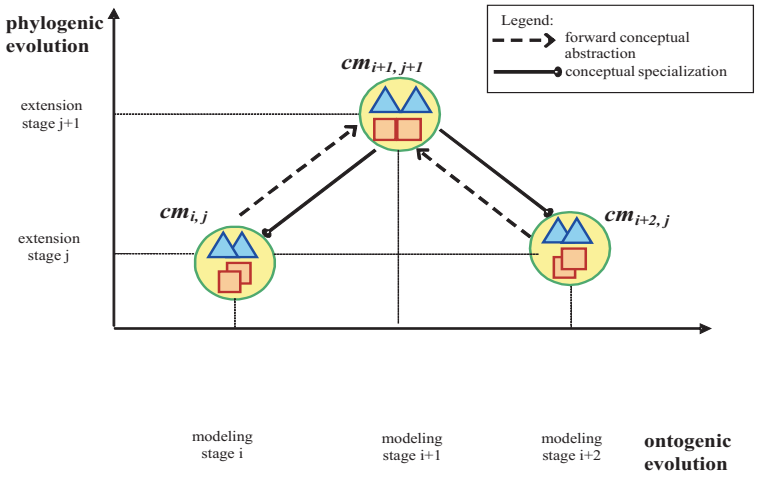


FIGURE 12. Horizontal abstraction as phylogenic evolution through forward conceptual abstraction and conceptual specialization

A ***conceptual model*** *is a triple* $M = (E, S, A)$*, where:*

(i) $E = \{e_1, \ldots, e_m\}$ *is the set of entities or concepts within the model;*

(ii) $S = \{s_{ij} | e_i, e_j \in E,\ \exists e_i\ s_{ij}\ e_j, i, j \in \{1, \ldots, m\}\}$ *is the set of associations between the entities within the model;*

(iii) $Attr_i = \{a_{i_1}, \ldots, a_{i_k}\}$ *is the set of attributes for the entity* $e_i, i = \overline{1, m}$, *and*
$A = \cup_{i=1}^{m} Attr_i$ *is the set of all attributes within the model.*

In what follows, by $\mathcal{P}(X)$ is denoted the power set of X.

Definition 3.2. *Refactoring ([5])*
Let $M_1 = (E_1, S_1, A_1)$ *and* $M_2 = (E_2, S_2, A_2)$ *be two conceptual models. A* ***refactoring*** *is a triple* $r = (e_r, s_r, a_r)$ *that transforms* M_1 *to* M_2. *Furthermore, the following constraints are met:*

(i) $e_r : E_{1_r} \to \mathcal{P}(E_{2_r})$ *maps the affected set of entities of the two conceptual models, where* $E_{1_r} \subseteq E_1, E_{2_r} \subseteq E_2$;

(ii) $s_r : S_{1_r} \to \mathcal{P}(S_{2_r})$ *maps association relationship changes between the two conceptual models, where* $S_{1_r} \subseteq S_1, S_{2_r} \subseteq S_2$;

(iii) $a_r : A_{1_r} \to \mathcal{P}(A_{2_r})$ *maps the affected set of attributes of the two conceptual models, where* $A_{1_r} \subseteq A_1, A_{2_r} \subseteq A_2$.

This is denoted by $M_1 \overset{r}{\to} M_2$.

Definition 3.3. *Forward Conceptual Abstraction ([5])*
Let $M_1 = (E_1, S_1, A_1)$ *and* $M_2 = (E_2, S_2, A_2)$ *be two conceptual models. A* ***forward conceptual abstraction*** *is a triple* $fca = (e_{fca}, s_{fca}, a_{fca})$ *that transforms* M_1 *to* M_2 *(denoted by* $M_1 \overset{fca}{\to} M_2$*). Furthermore, the following constraints are met:*

(i) $e_{fca} : E_{fca} \to \mathcal{P}(E_2)$ *maps a new set of entities* E_{fca} *to the* M_2 *conceptual model, where* $E_1 \cap E_{fca} = \varnothing, E_1 \cup E_{fca} = E_2$;

(ii) $s_{fca} : S_{fca} \to \mathcal{P}(S_2)$ *maps a new set of association relationships* S_{fca} *to the* M_2 *conceptual model, where* $S_1 \cap S_{fca} = \varnothing, S_1 \cup S_{fca} = S_2$;

(iii) $a_{fca} : A_{fca} \to \mathcal{P}(A_2)$ *maps a new set of attributes* A_{fca} *to the* M_2 *conceptual model, where* $A_1 \cap A_{fca} = \varnothing, A_1 \cup A_{fca} = A_2$.

Refactoring allows to switch between variants on the same abstraction level, while forward conceptual abstraction increases the generality of the target conceptual model, placing it on a higher extension stage. In order to reach a lower abstraction level, transformations that reduce complexity and generality are applied. Conceptual specialization decreases the abstraction level, being a special case of forward conceptual abstraction that achieves the transformation in the reverse order. Similar to this, conceptual specialization uses refactoring to move towards a lower extension stage. Therefore, a conceptual specialization is defined as a refactoring that acts as a reversed forward conceptual abstraction.

Definition 3.4. *Conceptual Specialization ([5])*

Let $M_1 = (E_1, S_1, A_1)$ and $M_2 = (E_2, S_2, A_2)$ be two conceptual models. A ***conceptual specialization*** *is a triple $r = (e_{cs}, s_{cs}, a_{cs})$ that transforms M_1 to M_2 (denoted by $M_1 \xrightarrow{cs} M_2$). Furthermore, the following constraints are met:*

(i) $e_{cs} : E_{1_{cs}} \to \mathcal{P}(E_2)$ *maps the set of entities $E_{1_{cs}}$ to the M_2 conceptual model, where $E_{1_{cs}} \subseteq E_1, E_2 \subseteq E_1$;*

(ii) $s_{cs} : S_{1_{cs}} \to \mathcal{P}(S_2)$ *maps the set of association relationships $S_{1_{cs}}$ to the M_2 conceptual model, where $S_{1_{cs}} \subseteq S_1, S_2 \subseteq S_1$;*

(iii) $a_{cs} : A_{1_{cs}} \to \mathcal{P}(A_2)$ *maps the set of attributes $A_{1_{cs}}$ to the M_2 conceptual model, where $A_{1_{cs}} \subseteq A_1, A_2 \subseteq A_1$.*

Following the notions previously introduced, the ontogenic and phylogenic processes are formally defined.

Definition 3.5. *Ontogenic Evolution ([5])*

Let $M_1 = (E_1, S_1, A_1)$ and $M_2 = (E_2, S_2, A_2)$ be two conceptual models, where $E_2 - E_1 = \varnothing$, $S_2 - S_1 = \varnothing$ and $A_2 - A_1 = \varnothing$. An ***ontogenic evolution*** *is a transformation $t_o = (e_{t_o}, s_{t_o}, a_{t_o})$ that transforms M_1 to M_2 (denoted by $M_1 \xrightarrow{t_o} M_2$). The transformation t_o has the following properties:*

(i) *t_o consists of (small) changes that does not affect the semantics of the M_2 model;*

(ii) *t_o is a refactoring, i.e., $M_1 \xrightarrow{t_o} M_2$ is achieved by $M_1 \xrightarrow{r} M_2$, $t_o = r$.*

Definition 3.6. *Phylogenic Evolution ([5])*

Let $M_1 = (E_1, S_1, A_1)$ and $M_2 = (E_2, S_2, A_2)$ be two conceptual models, where $E_2 - E_1 = E_p$, $S_2 - S_1 = S_p$ and $A_2 - A_1 = A_p$. A ***phylogenic evolution*** *is a transformation $t_p = (e_{t_p}, s_{t_p}, a_{t_p})$ that transforms M_1 to M_2 (denoted by $M_1 \xrightarrow{t_p} M_2$). The transformation t_p has the following properties:*

(i) *t_p consists of changes that affect the semantics and the abstraction level of the M_2 model;*

(ii) *if M_2 is a more general model than M_1 then t_p is a forward conceptual abstraction, i.e., $M_1 \xrightarrow{t_p} M_2$ is accomplished by $M_1 \xrightarrow{fca} M_2$, $t_p = fca$;*

(iii) *if M_1 is a more general model than M_2 then t_p is a conceptual specialization, i.e., $M_1 \xrightarrow{t_p} M_2$ is achieved by $M_1 \xrightarrow{cs} M_2$, $t_p = cs$.*

The three types of conceptual modeling variability are defined as biological evolution processes, following the ontogenic and phylogenic principles.

Definition 3.7. *Construct Variability Evolution ([5])*

Let $M_1 = (E_1, S_1, A_1)$ and $M_2 = (E_2, S_2, A_2)$ be two conceptual models. The ***construct variability*** *is a transformation $cVar$ (denoted by $M_1 \overset{cVar}{\to} M_2$). The following statements stay within the construct variability:*

(i) *within the* ***ontogenic evolution*** $M_1 \overset{t_o}{\to} M_2$*:*
   (a) *a refactoring transformation is applied, i.e., $M_1 \overset{r}{\to} M_2$, and $cVar = t_o = r$, where M_1 and M_2 have the same abstraction modeling level;*

(ii) *within the* ***phylogenic evolution*** $M_1 \overset{t_p}{\to} M_2$*:*
   (a) *a forward conceptual abstraction may be applied, i.e., $M_1 \overset{fca}{\to} M_2$, and $cVar = t_p = fca$, where M_1 and M_2 have different abstraction modeling levels and M_2 is a more general model than M_1;*
   (b) *a conceptual specialization is applied, i.e., $M_1 \overset{cs}{\to} M_2$, and $cVar = t_p = cs$, where M_1 and M_2 have different abstraction modeling levels and M_1 is a more general model than M_2.*

Definition 3.8. *Vertical Abstraction Variability Evolution ([5])*

Let $M_1 = (E_1, S_1, A_1)$ and $M_2 = (E_2, S_2, A_2)$ be two conceptual models. The ***vertical abstraction variability*** *is a transformation $vVar$ (denoted by $M_1 \overset{vVar}{\to} M_2$). The following statements stay within the vertical abstraction variability:*

(i) *within the* ***phylogenic evolution*** $M_1 \overset{t_p}{\to} M_2$*:*
   (a) *a forward conceptual abstraction may be applied, i.e., $M_1 \overset{fca}{\to} M_2$, and $vVar = t_p = fca$, where M_1 and M_2 have different abstraction modeling levels and M_1 is converted to a more general model M_2;*
   (b) *a conceptual specialization may be applied, i.e., $M_1 \overset{cs}{\to} M_2$, and $vVar = t_p = cs$, where M_1 and M_2 have different abstraction modeling levels and M_1 is converted to a more specific model M_2.*

Definition 3.9. *Horizontal Abstraction Variability Evolution ([5])*

Let $M_1 = (E_1, S_1, A_1)$ and $M_2 = (E_2, S_2, A_2)$ be two conceptual models. The ***horizontal abstraction variability*** *is a transformation $hVar$ (denoted by $M_1 \overset{hVar}{\to} M_2$). The following statements stay within the horizontal abstraction variability:*

(i) *within the* ***phylogenic evolution*** $M_1 \overset{t_p}{\to} M_2$*:*
   (a) *a forward conceptual abstraction may be achieved, i.e., $M_1 \overset{fca}{\to} M_2$, and $hVar = t_p = fca$, where M_1 and M_2 have different modeling*

abstraction levels and the M_1 model is transformed to the M_2 model by adding a new primary dimension;

(b) *a conceptual specialization may be applied, i.e., $M_1 \overset{cs}{\rightarrow} M_2$, and $hVar = t_p = cs$, where M_1 and M_2 have different abstraction modeling levels and the M_1 model is transformed to the M_2 model by removing a primary dimension.*

The existing relations among various conceptual models within the same or different extension stages of the development process is formalized too.

Definition 3.10. *Ontogenic Equivalence ([5])*

Let $M_1 = (E_1, S_1, A_1)$ and $M_2 = (E_2, S_2, A_2)$ be two conceptual models. Then:

*M_1 is **ontogenically equivalent** to M_2 (denoted by $M_1 \equiv M_2$) if $\exists\, t_{o_r}$, t_{o_l} two ontogenic transformations such that $M_1 \overset{t_{o_r}}{\rightarrow} M_2$ and $M_2 \overset{t_{o_l}}{\rightarrow} M_1$.*

This means that M_1 and M_2 belong to the same extension stage, while t_{o_r} and t_{o_l} are *refactorings* that transform a conceptual model to another.

Definition 3.11. *Phylogenic Dominance ([5])*

Let $M_1 = (E_1, S_1, A_1)$ and $M_2 = (E_2, S_2, A_2)$ be two conceptual models. Then:

(i) *M_1 is **phylogenically dominated** by M_2 (denoted by $M_1 < M_2$) if $\exists\, t_{p_u}$ a phylogenic transformation such that $M_1 \overset{t_{p_u}}{\rightarrow} M_2$ and M_2 is a more general conceptual model than M_1;*

(ii) *M_1 **phylogenically dominates** M_2 (denoted by $M_1 > M_2$) if $\exists\, t_{p_d}$ a phylogenic transformation such that $M_1 \overset{t_{p_d}}{\rightarrow} M_2$ and M_1 is a more general conceptual model than M_2.*

This means that M_1 and M_2 belong to different extension stages, while t_{p_u} is a *forward conceptual abstraction* and t_{p_d} is a *conceptual specialization* that allows to shift between conceptual models.

4. Conclusions and Future Work

Variability occurs in almost every modeling activity and its exploitation may help modelers to switch between taken decisions and to validate the model equivalence. Refactoring techniques are dedicated to design and implementation phase, but the research shows that their applicability may be extended to the conceptual modeling level.

Even though refactoring was applied to some theoretical but representative conceptual model examples, there is a large confidence that refactoring is reliable when it is used on particular and large UoD. For each type of variability, the specific problems on refactoring between variants were studied.

There are several important ideas that emerge from this analysis:

- construct and vertical abstraction variability and the application of refactorings between them are already recognized in software evolution practice and research [9, 12];
- within vertical abstraction variability, the transformation to a more generic variant requires forward conceptual abstraction, similar to a forward engineering at modeling stage;
- within horizontal abstraction variability shifting between variants requires an intermediate conceptual model, with an inconsistent modeling state and reduced relevance.

A biological evolution model was proposed in order to cope with different types of variability previously identified. Three specific transformations were suggested to shift between *ontogenic* and *phylogenic* conceptual models: refactoring, forward conceptual abstraction, and conceptual specialization.

Furthermore, there are aspects that have to be analyzed in the near future, like: a thoroughly study of the switching process between horizontal abstraction variants and to estimate the refactoring effort between variants.

References

[1] C. Batini, S. Ceri, and S. Navathe. *Conceptual database design: an entity relationship approach*. Benjamin/Cummings, 1992.

[2] M.C. Chisăliţă-Creţu. Efecte ale refactorizării asupra structurii interne a codului. *"Analele Facultăţii", Seria Ştiinţe Economice, Universitatea Creştină "Dimitrie Cantemir" Bucureşti, Facultatea de Ştiinţe Economice Cluj-Napoca, ISSN:1584-5621*, 13(1):214–230, 2005.

[3] M.C. Chisăliţă-Creţu. General aspects of refactoring applicability to conceptual models. In *Proceedings of the Symposium "Colocviul Academic Clujean de INFORMATICĂ"(CACI2005)*, pages 99–104, 2005.

[4] M.C. Chisăliţă-Creţu. Describing low level problems as patterns and solving them via refactorings. *"Studii şi Cercetări Ştiinţifice", Seria Matematică, ISSN: 1224-2519*, (17):29–48, 2007.

[5] M.C. Chisăliţă-Creţu and A. Mihiş. A model for conceptual modeling evolution. In *The 7th International Conference on Applied Mathematics (ICAM 2010), September 1-4, 2010, Baia-Mare, Romania*, 2010.

[6] S. Demeyer, S. Ducasse, and O. Nierstrasz. *Object-Oriented Reengineering Patterns*. Morgan Kaufmann and DPunkt, 2002.

[7] B. Du Bois. Opportunities and challenges in deriving metric impacts from refactoring postconditions. In *In Proceedings of the Fifth International Workshop on Object Oriented Reengineering (WOOR2004), ECOOPworkshop*, 2004.

[8] M. Fayad and D. Schmidt. Object-oriented application frameworks. *Communications of the ACM*, 40(10):32–38, 1997.
[9] M. Fowler. *Refactoring Improving the Design of Existing Code.* Addison-Wesley, 1999.
[10] D. Garlan and D. Perry. Introduction to the special issue on software architecture. *IEEE Transactions on Software Engineering*, 21(4):269–274, 1995.
[11] IEEE. *Standard IEEE Std 610.12-1990: IEEE standard glossary of software engineering terminology, In IEEE standards collection: software engineering.* IEEE Press, 1990.
[12] J. Kerievsky. *Refactoring to Patterns.* Addison-Wesley Professional, 2004.
[13] H. R. Maturana and F.J. Varela. *The Tree of Knowledge: The Biological Roots of Human Understanding.* Shambhala Publiccations,Inc., Boston, MA., USA, 1998.
[14] T. Mens, S. Demeyer, and D. Janssens. Formalising behaviour preserving program transformations. *Graph Transformation of Lecture Notes in Computer Science*, 2505:286–301, 2002.
[15] T. Mens, S. Demeyer, B. Du Bois, H. Stenten, and P. Van Gorp. Refactoring: Current research and future trends. *Electronic Notes in Theoretical Computer Science*, 82(3):483–499, 2003.
[16] D. Parnas. Designing software for ease of extension and contraction. *IEEE Transactions on Software Engineering*, 5(2):128–128, 1979.
[17] H. Schmid. Systematic framework design by generalization. *Communications of the ACM*, 40(10):48–51, 1997.
[18] S. Tichelaar, S. Ducasse, S. Demeyer, and O. Nierstrasz. A meta-model for language-independent refactoring. In *In Proceedings of the International Symposium on Principles of Software Evolution*, pages 157–169. IEEE Computer Society Press, 2000.
[19] J. Verelst. The influence of the level of abstraction on the evolvability of conceptual models of information systems. In *Proceedings of the International Symposium on Empirical Software Engineering (IIESE04), Los Angeles, IEEE CS Press*, pages 17–26, 2004.

Computer Science Department, Faculty of Mathematics and Computer Science, Babeş-Bolyai University, Cluj-Napoca, Romania
E-mail address: cretu@cs.ubbcluj.ro

STUDIA UNIV. BABEŞ–BOLYAI, INFORMATICA, Volume **LVI**, Number 1, 2011

EVOLUTIONARY COMPUTING IN THE STUDY OF COMPLEX SYSTEMS

DAVID ICLĂNZAN[(1)], RODICA IOANA LUNG[(2)], ANCA GOG[(1)], AND CAMELIA CHIRA[(1)]

Abstract. Complex systems and their important principles of emergence, auto-organization and adaptability are being intensively studied by researchers in a variety of fields including physics, biology, computer science, sociology and economics. The aim of this paper is to highlight potential interesting research directions lying at the intersection of nature-inspired computing and complex systems and able to generate new insights on modelling complexity. The field of evolutionary computation comprises classes of nature inspired search, design and optimization methods that can be applied to a variety of complex problems. Complex systems and relevant evolutionary computation methods are reviewed and analysed in this paper. Several aspects relating evolutionary computation to emergent and self-organization phenomena are emphasized.

1. Introduction

A complex system is any system containing a large number of interacting entities (agents, processes, etc.) which are interdependent. The system behaviour cannot be identified by considering each individual entity and combining them, but considering how the relationships between entities affect the behaviour of the whole system. The main features of complex systems include emergence, self-organization, evolution and adaptability. Emergence occurs when the behaviour of a system cannot be reduced to the sum of the behaviour of the parts. Self-organization is the process by which elements interact to create spatio-temporal patterns of behaviour that are not directly imposed by external forces. The formation of complex systems, and the structural/functional change of such systems, is a process of adaptation. Evolution

Received by the editors: March 22, 2011.

2010 *Mathematics Subject Classification.* 68-02.

1998 *CR Categories and Descriptors.* A.1 General Literature [**INTRODUCTORY AND SURVEY**]; I.2.8 Computing Methodologies [**ARTIFICIAL INTELLIGENCE**]:Problem Solving, Control Methods, and Search – *Heuristic methods*.

Key words and phrases. evolutionary computation, complex systems, emergence, self-organization, cellular automata.

is the adaptation of populations through inter-generation changes in the composition of the population. Learning is a similar process of adaptation of a system through changes in its internal patterns. An extensive study of complex systems and cellular automata as important tools in the analysis of complex interactions and emergent systems has been presented in [17].

The development of complex organizations, structures and mechanisms found in nature are explained by Charles Darwin's theory of evolution [19], one of the greatest scientific achievements of all times. This powerful evolutionary paradigm stands behind a class of nature inspired optimization methods, called Evolutionary Computation (EC). Some optimization methods belonging to the EC class share more than the basic principles and operators [39]. Certain high-level phenomena like the Baldwin effect [38, 65, 73, 71, 5, 28], coevolution and arm races [64, 61, 11, 72, 31], parasitism [56, 63], exaptation [24, 33, 35] and speciation [60] are found in methods and models belonging to EC.

Daniel Dennett has underlined that, "evolution will occur whenever and wherever three conditions are met: replication, variation (mutation), and differential fitness (competition)" [26], thus the digital transference of the natural selection paradigm must be also capable of producing self-organization and adaptability. Several such examples are reviewed in this paper based on a broad analysis of complex systems, EC major areas of interest and important aspects that connect emergent self-organizing phenomena with evolutionary computation. The main emerging research areas are discussed and potential interesting directions able to generate new insights on modelling complexity are emphasized.

The paper is organized as follows: Section 2 presents evolutionary techniques as tools for obtaining complex behaviours; Section 3 describes how EC can be used to evolve irreducible complexity; Section 4 presents the significant role that EC plays in many aspects of the Artificial Life domain; Section 5 shows that evolutionary techniques can be applied for automating the design of heuristic search methods and Section 6 contains conclusions and further research directions.

2. Evolving complex behaviours

Inspired by the richness and robustness of behavioural complexity exhibited by living organisms, EC has been used to tackle a broad variety of problems regarding the development of desired behaviours or strategies. Indeed, truly emergent phenomena are those that cannot be controlled or foreseen. Their effect - beneficial or destructive - will only appear during or after the interaction of the components takes place. Therefore, the task of designing

a system presenting desired emergent behaviour or the task of avoiding undesired emergent behaviour becomes complex and challenging [9]. Since it is by definition impossible to know how design choices made on the component level affect the overall system behaviour such tasks may be realized only by means of computational extensive and expensive simulations. This kind of simulations can be approached by evolutionary computation tools.

There are several characteristics of Evolutionary Algorithms (EAs) that make them suitable for dealing with the challenges mentioned above:

- EAs are black box optimization heuristics, they do not impose any restrictions on the fitness function and thus can be used in a complex simulation environment;
- EAs can be efficiently parallelized, even on heterogeneous hardware platforms like computer grids [1];
- Approximation methods for fitness evaluation can significantly reduce the computational complexity of the simulation;
- EAs are known to be able to cope well with uncertainty in evaluation and capable to adapt to changing environments [7];
- The well established field of Evolutionary Multiobjective Optimization provides a range of methods and tools for dealing with multiple objectives which is a realistic approach in studying complex systems and possible emergent behaviour [23];
- EAs are highly adaptable to different solution concepts - using generative relations such as Pareto dominance or Nash ascendancy can lead to different types of solutions - the Pareto frontier or the Nash equilibria of a game respectively [29];
- EAs are known to be adaptive but they can be also interactive - both features making them useful in studying or designing possible emergent behaviour.

EAs have been successfully used in designing complex systems and inducing desired emergent behaviour. One example is the problem of designing en-route caching strategies [8] where genetic programming is engaged to design effective caching strategies. An EA was used with traffic simulation to design a traffic light controller in a multi-objective setting, attempting to minimize travel time as well as the number of stops.

Chellapilla and Fogel [16] used a genetic algorithm (GA) to evolve neural networks that could play the game of checkers. The major breakthrough of the paper is represented by the fact that a competitive strategy could be evolved given only the spatial positions of pieces on the checkerboard and the piece differential. The GA optimized artificial neural networks to evaluate

alternative positions in the game without relying on any specific credit assignment algorithm - a rewarding mechanism that would normally require human expertise.

Evolutionary search has been applied to develop strategies for many different games like Othello [52] or GO [62]. However, these results suggest that the evolutionary principles may be successfully applied also to problems that have not yet been solved by human expertise.

Evolutionary methods were also successfully applied by Andre and Teller to develop a program for controlling a team of robot soccer players [4]. They used a genetic programming algorithm which operated with a set of primitive control functions such as turning, moving and kicking. The fitness function rewarded good play in general, rather than scoring specific tasks. No code or elementary building block was provided to teach the team how to achieve complex objectives, like ball tracing, kicking the ball in the correct direction, keeping the ball on the opponent's side, goal scoring etc. The robot team, called Darwin United, entered the international RoboCup[1] tournament, an annual soccer tournament between teams of autonomous robots. Darwin United performed quite well, outranking half of the human-written, highly specialized entries.

Cellular Automata (CA) are decentralized structures of simple and locally interacting elements (cells) that evolve following a set of rules [74]. Programming CA is not an easy task, especially when the desired computation requires global coordination. CA provides an idealized environment for studying how (simulated) evolution can develop systems characterized by "emergent computation" where a global, coordinated behaviour results from the local interaction of simple components [49].

The most widely studied CA task is the density classification problem (DCT) [48]. The task refers to finding the density most present in the initial cellular state. Packard [57] made the first attempts to use genetic algorithms for finding CA rules for the density task. Genetic programming [41], coevolutionary learning [40] and gene expression programming [30] have also been engaged for this problem. Genetic algorithms for computational emergence in the density classification task have been extensively investigated by Mitchell et al [20, 49, 50, 58]. The human designed Gacs-Kurdyumov-Levin rule with a performance of 81.6% as all other known subsequent human-written rules for this problem, were surpassed by rules developed by simulated evolution. Andre et al. [3] found a rule performing with an 82.23% accuracy by using genetic programming. The best currently reported DCT rule has a performance of 89% [54], which was evolved by a two-tier evolutionary algorithm.

[1] `http://www.robocup.org/`

The potential of evolutionary models to efficiently approach the problem of detecting CA rules to facilitate a certain global behaviour is also confirmed by other current research results [46, 77, 32, 25].

Another CA computational task intensely studied is the synchronization task (ST) [21], where the goal is to find 1D binary CA able to reach a configuration that cycles from all cells 0 in one time step to all cells 1 in next time step (starting from an arbitrary initial configuration). Evolutionary models have been successfully engaged for the synchronization task in several studies [21, 46]. There are several one-dimensional radius-3 CA rules able to solve ST for any arbitrary lattice configuration [21] with an efficacity of approximately 95%. Genetic algorithms proposed in [53, 46] for the synchronization task are able to find radius-2 rules with high efficacy.

Research has also been focused on evolving behaviours in multidimensional CA. Morales et al. [51], Alonso and Bull [2] have studied the DCT in a two dimensional setting. In [10] the authors use GA to evolve behaviour in multidimensional CA for DCT, the checkerboard problem that requires the formation of an alternating simple pattern and finally for generic bitmap evolution. The authors found that symmetrical bitmaps seem to be easier to generate than asymmetric ones and that multidimensional CA can solve certain problems faster than one dimensional CA. Chavoya and Duthen successfully evolve CA to produce predefined 2D and 3D shapes [14]. In a later work they apply a GA to evolve an extended artificial regulatory network to produce predefined 2D cell patterns [15].

The evolutionary discovery of rules that produce global synchronization is significant, since these exemplify the automated development of sophisticated emergent computation in decentralized, distributed systems such as CA. These discoveries are encouraging for the prospect of using EC to automatically evolve behaviour for more complex tasks, like predicting chaotic sequences.

3. Evolving irreducible complexity

An important challenge to evolutionary theory is to explain the origin and development of complex organismal features. Michael Behe, the originator of the term irreducible complexity (IC), defines an IC system as one "composed of several well-matched, interacting parts that contribute to the basic function, wherein the removal of any one of the parts causes the system to effectively cease functioning" [6]. IC is a central argument for proponents of intelligent design, revolving around the belief that such systems demonstrate that modern biological forms could not have evolved naturally.

Evolutionary biologists have shown that through evolutionary mechanism like deletion or addition or multiple parts, change of function or addition of

second function to a part, gradual modification and loss of previously existing scaffolding can contribute to the production of IC systems.

In [34] the authors use a simple GA with variable-length chromosomes over a dynamic fitness function, in order to demonstrate that GAs can produce systems composed of multiple parts contributing to a specified complex function, where all components are critical. Therefore, such a design is irreducible.

The problem to be solved by the GA is a game defined over a 30×14 board, where the goal is to attain fitness scores greater than zero. As better solutions are evolved, the game responds by becoming more and more difficult.

By default, each grid cell in the game board is blank. The genes of individuals code mappings onto the game board; the encoded game cells, called "boxes" have one of four types, designated by the letters *S,A,L,R*. As the game begins, a virtual ball falls through the board, entering at the column index 5, carrying an initial point value of 15. The four box types can modify this value and the direction of travel through the board and can also duplicate the ball. The goal of the game is to multiply game balls and steer them to the sink column, with index 8. Game balls that exit the board on different columns do not affect the fitness score.

The effect of the four box types is the following:

(1) The *Split box* (S-box) duplicates the game ball, each with a point value one less than the original and with a different exit path from the box.
(2) The *Add box* (A-box) increases the point value of a game ball passing through it.
(3) The *Left boxes* (L-boxes) and *Right boxes* (R-boxes) modify the ball's direction of travel (right and left turn).

The fitness score of an individual x is defined as:

$$f(x) = max(0, p(x) - l(x) \cdot P) \tag{1}$$

where $p(x)$ is the number of collected points, $l(x)$ is the chromosome length, and P is a population-wide penalty value which increases with time to make the game more difficult as the individuals evolve. Individuals with fitness zero are considered non-viable, and they do not participate in the tournament selection employed by the GA.

Graham et al. [34] conducted experiments to verify if a GA can construct irreducible complex solutions to this game. An individual was regarded irreducible complex if its fitness was greater than zero, it contained more then five boxes (simple solutions were not of interest) and the removal of any single part (box) from the phenotype resulted in the fitness of the individual dropping to zero. With small population sizes of 50 individuals, they were able to constantly evolve irreducible solutions to this game. Such a highly-evolved solution is depicted in Fig. 1.

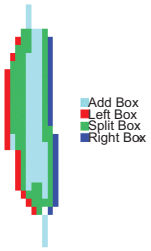


FIGURE 1. An example of a highly-evolved individual as reported in [34]. It can sustain a penalty of $P \cong 1.1 \times 10^{12}$ and collects approximately 3.6×10^{13} points.

The game solution strategies found by the GA are variations of the general pattern observable in Fig. 1. The source and sink column is connected with a cluster of A and S-boxes that multiply the game balls and increase their point value. Each side of the cluster if braced by L and R-boxes, that route the balls back into the cluster and towards the sink column.

The work[2] has demonstrated with the help of a simulated environment that simple evolutionary mechanisms can produce irreducible complexity, consisting of more than 100 parts.

In [18], Clayton depicts how IC systems can be obtained in a simple system that operates on a regular two-dimensional triangular lattice. The nodes in the considered lattice are binary. Whenever two adjacent nodes are set to the value 1 or ON, an edge automatically connects them. A group of connected nodes forms a system which is considered viable if and only if it forms a closed geometric shape and its fitness is directly proportional with its perimeter (larger systems are preferred). Through a simulated evolution, the author demonstrates that irreducible complexity evolves in this model in response to natural selection, favoring the larger systems with fewer parts. A continual increase in complexity was observed in 100 evolutionary steps, resulting in IC systems containing between 6 and 30 parts.

GAs are also used to address the watchmaker analogy, which affirms that the highly complex inner workings of a system (ex. watch) necessitate an intelligent designer. The experiment[3] demonstrates that if the components of a watch are allowed to be combined and the resulting design undergoes natural selection, a functional watch can be evolved. The clocks evolve through a series of transitional forms, with ever increasing complexity. At initialization,

[2]An online demonstration of the game and algorithm can be found at `http://www.stellaralchemy.com/ice/index.php`.

[3]`http://richarddawkins.net/videos/1322-evolution-is-a-blind-watchmaker`

98% of the designs are non functional, the remaining designs being simple pendulums. As they evolve, proto-clocks, more sophisticated clocks gradually develop, starting from 1 hand and resulting in up to 4 hands. The most sophisticated clock obtained had 21 interacting parts.

4. Evolving digital organisms and ecosystems

Natural evolution as well as its simulated variant can produce complex system configurations and behaviours. Therefore, EC plays a significant role in many aspects of the Artificial Life (AL) domain as behaviour strategies, methods of communication, swarm intelligence and many other topics are commonly explored using evolutionary search techniques (see the "From Animals to Animats" or "Artificial Life" conference series). Furthermore, the dynamics of Darwinian evolution and different hypotheses and models of evolution are often studied through digital organisms - artificial life-forms, that are defined as self-replicating digital models that mutate, compete and evolve.

One of the first experiments with digital organisms was conducted by ecologist Thomas S. Ray in the Tierra model [69], where computer programs capable of mutation and self-replication compete for central processing unit time (energy) and access to main memory (resources). The model has been used to conduct experiments regarding evolutionary and ecological dynamics, including dynamics of punctuated equilibrium, host-parasite coevolution and density-dependent natural selection. In the Tierra framework the fitness function is endogenous; there is simply survival or death of a digital organism.

A related framework, AVIDA, where each digital organism lives in its own protected region of memory and is executed by its dedicated virtual CPU, was used to conduct research in the digital evolution of complex features [43]. The digital organisms evolve to perform certain computational tasks, from which the most complicated one is the equality operator - requiring at least 19 simpler, precisely ordered instructions.

Other noteworthy examples of digital organism simulators include:

(1) *Evolve 4.0*[4] a 2D cellular automata where each cell can behave independently as unicellular organism or be a part of a multicellular creature. The digital organisms can grow, movie, feed and replicate.
(2) *Darwinbots* [5], a digital environment of interacting and fighting bots, where the behaviour of the bots is specified by their genome.
(3) *breve*[6], a 3D simulator for multi-agent systems and artificial life, with support for physical simulation and collision detection.

[4] `http://stauffercom.com/evolve4/`
[5] `http://www.darwinbots.com/WikiManual/index.php?title=Main_Page`
[6] `http://www.spiderland.org/breve/`

(4) *Polyworld*[7], an ecosystem of agents which search for food, mate, replicate and hunt. The individuals actions are governed by arbitrary architecture neural networks employing Hebbian learning. The neural network is encoded in each individual's genome that is mutated and replicated. Recent results [75] experimenting with Polyworld had underlined an association between small-world network structures of the controlling networks and complex neural dynamics.
(5) *AnimatLab*[8], is a recently developed simulation tool combining biomechanical simulation and biologically realistic (spiking) neural networks.

Lohn's CA, evolved by a GA to be capable of self-replication [45] and the plant-like biomorphs introduced by Dawkins [22] are another examples of evolutionary AL-forms.

Sims [67, 66] demonstrates the development of animal-like morphologies by simulating Darwinian evolutions of virtual block creatures. The fitness of the initially randomly generated block creatures is measured in their ability to perform a given task, for example swimming in a simulated water environment. The creatures undergoing natural selection and variation developed successful behaviours for swimming, walking, jumping, following, and competing for control of a (resource) cube.

5. Self-* search

Emergent phenomena observed in natural systems have been used as an inspiration for designing many evolutionary computation models. For example Ant colony optimization (ACO) [27] or Particle Swarm Optimization (PSO) [55] methods mimic emergent features mentioned above to solve complex search and optimization problems. Emergent phenomena are carefully observed and used as an inspiration for designing new efficient techniques.

Nevertheless, as evolutionary search is capable of producing highly coadapted complex systems that are often irreducible, there is a growing research interest in evolutionary techniques for automating the (self) design of heuristic search methods. Successful approaches alleviate the need for human experts in the process of designing efficient problem dependent optimization methods (heuristics).

There are two basic approaches to turn simpler methods into self-* algorithms or hyperheuristics: one is built upon machine learning techniques to identify good parameter settings, proper operators and algorithmic building blocks; the second one uses a meta-level search over the parameterization of the base method, where the selection of the good features can be decided in a

[7] `http://beanblossom.in.us/larryy/Polyworld.html`
[8] `http://www.animatlab.com/`

fast greedy way, susceptible of finding (weak) local-optima or by a computationally more expensive evolutionary means.

Regardless of the machine-learning or meta-search based approach, related self-adaptive and meta-level methods revolve around three general processes in automated heuristic design:

(1) Adjusting or tuning the method's control parameters, an approach exemplified by adaptive self-tuning Evolution Strategies [36] or automatically selected perturbation step size in Iterated Local Search [68].
(2) Dynamic selection of existing algorithmic components, ex. managing the search operators in an EC algorithm [47] or the application of various linkage learning techniques for developing competent crossover operators [37, 59, 76, 44].
(3) Generating new heuristics from basic sub-components, an approach implemented by the "Teacher"[9] framework [70].

The literature regarding this field is immense and it can not be covered in this review. For a more in depth discussion, we forward the interested reader to recent reviews on this subject [42, 13, 12].

However, we would like to point out that the evolutionary paradigm can be recursively applied to enhance EC methods. Self-adaptation is an implicit parameter adaptation technique enabling the evolutionary search to tune the strategy parameters automatically by evolution [42].

6. Conclusions

Evolutionary computation techniques have been successfully applied for problems that arise from the study of complex systems principles of emergence, auto-organization and adaptability.

A two way relationship between the two domains can be observed. On the one hand, their interaction gave rise to new efficient optimization techniques inspired by emergent phenomena and helped improving different heuristic methods in terms of tuning control parameters or dynamic selection of components. On the other hand, evolutionary techniques have been used for designing complex systems. For example, complex desired behaviours and strategies have been evolved by means of evolutionary techniques. Cellular automata is a great example of global, coordinated behaviour that results from the local interaction of simple components. Several other examples include caching strategies, traffic controllers, strategies for difficult games etc. Another major application of EC is the production of irreducibly complex systems characterized by the fact that removing any of the systems parts causes the system to cease functioning. We also present the role that EC has in

[9] An acronym for TEchniques for the Automated Creation of HEuRistics

the Artificial Life domain, for aspects like behaviour strategies, methods of communication, swarm intelligence and many other topics.

There are many research perspectives at the intersection of nature-inspired computing and complex systems worth to be further explored. We emphasize the potential of computer simulations using multi-agent modelling and evolutionary computing techniques and investiging complex network and cellular automata models for the analysis of complex systems. Different interaction models at the micro/macro (inidividual/population) level that induce emergent behavior can be studied using evolutionary computation and further explored in modelling complex systems.

7. Acknowledgments

This research is supported by Grant PN II TE 320, Emergence, auto-organization and evolution: New computational models in the study of complex systems, funded by CNCSIS, Romania.

References

[1] E. Alba, C. Cotta, *Parallelism and evolutionary algorithms*, IEEE Transactions on Evolutionary Computation, 6 (2002), pp. 443-462.

[2] R. Alonso-Sanz, L. Bull, *A very effective density classifier two-dimensional cellular automaton with memory.* Journal of Physics A: Mathematical and Theoretical, 42:485101 (2009).

[3] D. Andre, F.H. Bennett, J.R. Koza, *Discovery by genetic programming of a cellular automata rule that is better than any known rule for the majority classification problem*, In Proceedings of the First Annual Conference on Genetic Programming, GECCO '96, Cambridge, MA, USA (1996), pp. 3-11.

[4] D. Andre, A. Teller, *Evolving team darwin united*, In Minoru Asada and Hiroaki Kitano, editors, RoboCup-98: Robot Soccer World Cup II, Springer Verlag (1999), pp. 346-351.

[5] J. Bala, K. De Jong, J. Huang, H. Vafaie, H. Wechsler, *Using learning to facilitate the evolution of features for recognizing visual concepts.*, Evolutionary Computation, 4 (1996), pp. 297-311.

[6] M.J. Behe, *Darwin's black box: the biochemical challenge to evolution*, Free Press New York (1996).

[7] J. Branke, *Evolutionary Optimization in Dynamic Environments*, Kluwer Academic Publishers, Norwell, MA, USA (2001).

[8] J. Branke, P. Funes, F. Thiele, *Evolving en-route caching strategies for the internet*, In Genetic and Evolutionary Computation GECCO 2004, LNCS Springer Berlin / Heidelberg, vol. 3103 (2004), pp. 434-446.

[9] J. Branke, H. Schmeck, *Evolutionary design of emergent behaviour*, In Organic Computing, volume 21 of Understanding Complex Systems, Springer Berlin / Heidelberg (2008), pp. 123-140.

[10] R. Breukelaar, Th. Bäck, *Using a genetic algorithm to evolve behaviour in multi dimensional cellular automata: emergence of behaviour*, In Proceedings of the 2005 conference on Genetic and evolutionary computation, GECCO '05, New York, NY, USA (2005), pp. 107-114.

[11] A. Bucci, J.B. Pollack, *On identifying global optima in cooperative coevolution*, In Proceedings of the 2005 conference on Genetic and evolutionary computation, GECCO '05, New York, NY, USA (2005), pp. 539-544.

[12] E.K. Burke, M. Hyde, G. Kendall, G. Ochoa, E. Ozcan, R. Qu, *A survey of hyper-heuristics*, Technical Report NOTTCS-TR-SUB-0906241418-2747, School of Computer Science and Information Technology, University of Nottingham (2010).

[13] K. Chakhlevitch and P. Cowling, *Hyperheuristics: recent developments*, Adaptive and Multilevel Metaheuristics (2008), pp. 3-29.

[14] A. Chavoya, Y. Duthen, *Using a genetic algorithm to evolve cellular automata for 2d/3d computational development*, In Proceedings of the 8th annual conference on Genetic and evolutionary computation, GECCO '06, New York, NY, USA (2006), pp. 231-232.

[15] A. Chavoya, Y. Duthen, *Use of a genetic algorithm to evolve an extended artificial regulatory network for cell pattern generation*, In Proceedings of the 9th annual conference on Genetic and evolutionary computation, GECCO '07, New York, NY, USA (2007), pp. 1062-1062.

[16] K. Chellapilla, D.B. Fogel, *Evolving an expert checkers playing program without using human expertise*, IEEE Transactions on Evolutionary Computation, 5:4 (2001), pp. 422-428.

[17] C. Chira, A. Gog, R. Lung, D. Iclanzan, *Complex Systems and Cellular Automata Models in the Study of Complexity*, Studia Informatica series, Vol. LV, No. 4 (2010), pp. 33-49.

[18] S.S. Clayton, *A simple model for the evolution of irreducible complexity*, Citeseer DOI 10.1.1.60.6797 (2006).

[19] C. Darwin, *On the Origin of Species*, John Murray, London, 1859.

[20] R. Das, M. Mitchell, J. P. Crutchfield, *A genetic algorithm discovers particle-based computation in cellular automata*, Parallel Problem Solving from Nature Conference (PPSN-III). Springer-Verlag (1994), pp. 344-353.

[21] R. Das, J.P. Crutchfield, M. Mitchell, J.E. Hanson, *Evolving globally synchronized cellular automata*, In Proceedings of the 6th International Conference on Genetic Algorithms, San Francisco, CA, USA, Morgan Kaufmann Publishers Inc (1995), pp. 336-343.

[22] R. Dawkins, *The Blind Watchmaker*, Penguin Books, 1986.

[23] K. Deb, D. Kalyanmoy, *Multi-Objective Optimization Using Evolutionary Algorithms*, John Wiley & Sons, Inc., New York, NY, USA, (2001).

[24] P. de Oliveira, *Simulation of exaptive behaviour*, In Yuval Davidor, Hans-Paul Schwefel, and Reinhard Mnner, editors, Parallel Problem Solving from Nature - PPSN III, LNCS vol. 866, Springer Berlin / Heidelberg (1994), pp. 354-364.

[25] P.P.B. de Oliveira, J.C. Bortot, G. Oliveira, *The best currently known class of dynamically equivalent cellular automata rules for density classification*, Neurocomputing, 70:1-3 (2006), pp. 35-43.

[26] DC Dennett, *The new replicators*, The encyclopedia of evolution, 1 (2002), pp. 83-92.

[27] M. Dorigo, T. Stutzle, *Ant Colony Optimization*, Bradford Company, Scituate, MA, USA (2004).

[28] K.L. Downing, *The baldwin effect in developing neural networks*, In Proceedings of the 12th annual conference on Genetic and evolutionary computation, GECCO '10, New York, NY, USA (2010), pp. 555-562.

[29] D. Dumitrescu, R.I. Lung, T.D. Mihoc, *Generative relations for evolutionary equilibria detection*, In Proceedings of the 11th Annual conference on Genetic and evolutionary computation, GECCO '09, New York, NY, USA (2009), pp. 1507-1512.

[30] C. Ferreira, *Gene Expression Programming: A New Adaptive Algorithm for Solving Problems*, Complex Systems, 13:2 (2001), pp. 87-129.

[31] S. Ficici, J.B. Pollack, *Challenges in coevolutionary learning: Arms-race dynamics, open-endedness, and mediocre stable states*, In Proceedings of the Sixth International Conference on Artificial Life, MIT Press (1998), pp. 238-247.

[32] A. Gog, C. Chira, *Cellular Automata Rule Detection Using Circular Asynchronous Evolutionary Search*, HAIS 2009, LNCS 5572 (2009), pp. 261-268.

[33] L. Graham and F. Oppacher, *Speciation through exaptation.* In Artificial Life, 2007. ALIFE'07. IEEE Symposium on, IEEE (2007), pp. 433-439.

[34] L. Graham, F. Oppacher, and S. Christensen, *Irreducible complexity in a genetic algorithm*, In IEEE Congress on Evolutionary Computation, IEEE (2007), pp. 3692-3697.

[35] L. Graham, *Exaptation and functional shift in evolutionary computing*, PhD thesis, Carleton University, Ottawa, Ont., Canada, Canada (2009).

[36] N. Hansen and A. Ostermeier, *Completely derandomized self-adaptation in evolution strategies.* Evolutionary computation, 9:2 (2001), pp. 159-195.

[37] G.R. Harik, D.E. Goldberg, *Learning linkage*, In Richard K. Belew and Michael D. Vose, editors, FOGA, Morgan Kaufmann (1996), pp. 247-262.

[38] G.E. Hinton, S.J. Nowlan, *How learning can guide evolution*, Adaptive individuals in evolving populations, Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 0-201-48369-6 (1996), pp. 447-454.

[39] J.H. Holland, *Adaptation in natural artificial systems*, University of Michigan Press, Ann Arbor (1975).

[40] H. Juille, J.B. Pollack, *Coevolving the ideal trainer: Application to the discovery of cellular automata rules*, In John R. Koza, Wolfgang Banzhaf, Kumar Chellapilla, Kalyanmoy Deb, Marco Dorigo, David B. Fogel, Max H. Garzon, David E. Goldberg, Hitoshi Iba, and Rick Riolo, editors, Genetic Programming 1998: Proceedings of the Third Annual Conference, University of Wisconsin, Madison, Wisconsin, USA, Morgan Kaufmann (1998), pp. 519-527.

[41] J. R. Koza, *Genetic Programming: On the Programming of Computers by Means of Natural Selection*, MIT Press (1992).

[42] O. Kramer, *Self-adaptive heuristics for evolutionary computation*, Springer Verlag, ISBN: 3540692800 (2008).

[43] R.E. Lenski, C. Ofria, R.T. Pennock, C. Adami, *The evolutionary origin of complex features*, Nature, 423:6936 (2003), pp. 139-144.

[44] Z. Li, E.D. Goodman, *Learning building block structure from crossover failure*, In GECCO '07: Proceedings of the 9th annual conference on Genetic and evolutionary computation, New York, NY, USA (2007), pp. 1280-1287.

[45] J.D. Lohn, J.A. Reggia, *Discovery of self-replicating structures using a genetic algorithm.* In IEEE International Conference on Evolutionary Computation, vol. 2, IEEE (1995), pp. 678-683.

[46] A. S. Mariano, G. M. B. de Oliveira, *Evolving one-dimensional radius-2 cellular automata rules for the synchronization task*, AUTOMATA-2008 Theory and Applications of Cellular Automata, Luniver Press (2008), pp. 514-526.

[47] J. Maturana, F. Lardeux, F. Saubion, *Autonomous operator management for evolutionary algorithms*, Journal of Heuristics (2010), pp. 1-29.

[48] M. Mitchell, P.T. Hraber, J.P. Crutchfield, *Revisiting the edge of chaos: Evolving cellular automata to perform computations*, Complex Systems, 7 (1993), pp. 89-130.

[49] M. Mitchell, J.P. Crutchfield, R. Das, *Evolving cellular automata with genetic algorithms: A review of recent work*, In Proceedings of the First International Conference on Evolutionary Computation and Its Applications (EvCA'96). Russian Academy of Sciences (1996).

[50] M. Mitchell, M. D. Thomure, N. L. Williams, *The role of space in the Success of Coevolutionary Learning*, Proceedings of ALIFE X - The Tenth International Conference on the Simulation and Synthesis of Living Systems (2006).

[51] F. Jiménez Morales, J. P. Crutchfield, M. Mitchell, *Evolving two-dimensional cellular automata to perform density classification: a report on work in progress*, Parallel Comput., 27 (2001), pp. 571-585.

[52] D.E. Moriarty, R.Miikkulainen, *Discovering complex othello strategies through evolutionary neural networks*, Connection Science, 7 (1995), 195-209.

[53] G. Oliveira, P. de Oliveira, N. Omar, *Definition and applications of a five-parameter characterization of one-dimensional cellular automata rule space*, Artificial Life, 7:3(2001), pp. 277-301.

[54] G.M.B. Oliveira, L.G.A. Martins, L.B. de Carvalho, E. Fynn, *Some investigations about synchronization and density classification tasks in one-dimensional and two-dimensional cellular automata rule spaces*, Electron. Notes Theor. Comput. Sci., 252 (2009), pp. 121-142.

[55] A.E. Olsson, *Particle Swarm Optimization: Theory, Techniques and Applications*, Nova Science Publishers, Inc., Commack, NY, USA (2010).

[56] B. Olsson, *A host-parasite genetic algorithm for asymmetric tasks*, In Claire Ndellec and Cline Rouveirol, editors, Machine Learning: ECML-98, LNCS vol. 1398, Springer Berlin / Heidelberg (1998), pp. 346-351.

[57] N. H. Packard, *Adaptation toward the edge of chaos. Dynamic Patterns in Complex Systems*, World Scientific (1988), pp. 293-301.

[58] L. Pagie, M. Mitchell, *A comparison of evolutionary and coevolutionary search*, Int. J. Comput. Intell. Appl., 2:1 (2002), pp. 53-69.

[59] M. Pelikan, D.E. Goldberg, E. Cantú-Paz, *Linkage problem, distribution estimation, and bayesian networks*, Evolutionary Computation, 8:3 (2000), pp. 311-340.

[60] Z.A. Perry, *Experimental study of speciation in ecological niche theory using genetic algorithms*, PhD thesis, University of Michigan, Ann Arbor, MI, USA (1984), AAI8502912.

[61] M.A. Potter, K.A. De Jong, *Cooperative coevolution: An architecture for evolving coadapted subcomponents*, Evolutionary Computation, 8 (2000), pp. 1-29.

[62] N. Richards, D.E. Moriarty, R. Miikkulainen, *Evolving neural networks to play Go*, Applied Intelligence, 8:1 (1998), pp. 85-96.

[63] P. Robbins, *The effect of parasitism on the evolution of a communication protocol an artificial life simulation*, In Proceedings of the third international conference on Simulation of adaptive behaviour: from animals to animats 3: from animals to animats 3, Cambridge, MA, USA, MIT Press (1994), pp. 431-437.

[64] C.D. Rosin, R.K. Belew, *Methods for competitive co-evolution: Finding opponents worth beating*, In Proceedings of the Sixth International Conference on Genetic Algorithms, Morgan Kaufmann (1995) pp. 373-380.

[65] G.G. Simpson, *The Baldwin effect*, Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA (1996), pp. 99-109.

[66] K. Sims, *Evolving 3D morphology and behaviour by competition*, Artificial Life, 1:4 (1994), pp. 353-372.

[67] K. Sims, *Evolving virtual creatures*, In Proceedings of the 21st annual conference on Computer graphics and interactive techniques, ACM (1994), pp. 15-22.

[68] D. Thierens, *Adaptive operator selection for iterated local search*, In Thomas Sttzle, Mauro Birattari, and Holger Hoos, editors, Engineering Stochastic Local Search Algorithms. Designing, Implementing and Analyzing Effective Heuristics, LNCS vol. 5752, Springer Berlin / Heidelberg (2009), pp. 140-144.

[69] E. Thro, *Artificial life explorer's kit*, chapter Natural A-Life. Sams Publishing, ISBN 0672303019 (1993).

[70] B.W. Wah, A. Ieumwananonthachai, *Teacher: A Genetics-Based System for Learning and for Generalizing Heuristics*, vol. Evolutionary Computation: Theory and Applications, World Scientific Publishing Co. Pte. Ltd (1999), pp. 179-211.

[71] D. Whitley, V. Gordon, K. Mathias, *Lamarckian evolution, the baldwin effect and function optimization*, In Yuval Davidor, Hans-Paul Schwefel, and Reinhard Mnner, editors, Parallel Problem Solving from Nature - PPSN III, LNCS vol. 866, Springer Berlin / Heidelberg (1994), pp. 5-15.

[72] R.P. Wiegand, M.A. Potter, *Robustness in cooperative coevolution*, In Proceedings of the 8th annual conference on Genetic and evolutionary computation, GECCO '06, New York, NY, USA (2006), pp. 369-376.

[73] J. Wiles, J.R. Watson, *How learning can guide evolution in hierarchical modular tasks*, In Proceedings of the 23rd Annual Conference of the Cognitive Science Society (2001), pp. 1130-1135.

[74] S. Wolfram, editor, *Theory and Applications of Cellular Automata*, Advanced series on complex systems, vol. 1, World Scientific Publishing, P. O. Box 128, Farrer Road, Singapore 9128 (1986).

[75] L. Yaeger, O. Sporns, S. Williams, X. Shuai, S. Dougherty, *Evolutionary selection of network structure and function*, In Proc. of the Alife XII Conference, Odense, Denmark (2010), pp. 313-320.

[76] T.L. Yu, K. Sastry, D.E. Goldberg, *Linkage learning, overlapping building blocks, and systematic strategy for scalable recombination*, In GECCO '05: Proceedings of the 2005 conference on Genetic and evolutionary computation, New York, NY, USA (2005), pp. 1217-1224.

[77] Y. Zhao, S. A. Billings, *Identification of the Belousov-Zhabotinsky Reaction using Cellular Automata Models*, International Journal of Bifurcation and Chaos, 17:5 (2007), pp. 1687-1701.

(1)Department of Computer Science, Babes-Bolyai University, Kogalniceanu 1, 400084 Cluj-Napoca, Romania

E-mail address: david.iclanzan@gmail.com, {anca,cchira}@cs.ubbcluj.ro

(2)Faculty of Economics and Business Administration, Babes-Bolyai University, Kogalniceanu 1, 400084 Cluj-Napoca, Romania

E-mail address: rodica.lung@econ.ubbcluj.ro

STUDIA UNIV. BABEŞ–BOLYAI, INFORMATICA, Volume **LVI**, Number 1, 2011

ADVANCED FUNCTOR FRAMEWORK FOR C++ STANDARD TEMPLATE LIBRARY

NORBERT PATAKI

Abstract. The *C++ Standard Template Library (STL)* is the most popular library based on the *generic programming paradigm*. STL is widely-used, because it consists of many useful generic data structures and generic algorithms that are fairly irrespective of the used container. Iterators bridge the gap between containers and algorithms. As a result of this layout the complexity of the library is reduced and we can extend the library with new containers and algorithms simultaneously.

Function objects (also known as *functors*) make the library much more flexible without significant runtime overhead. They parametrize user-defined algorithms in the library, for example, they determine the comparison in the ordered containers or define a predicate to find. Requirements of relations are specified, for instance, associative containers need strict weak ordering. However, these properties are tested neither at compilation-time nor at run-time. If we use a relation that is not a strict weak ordering, the containers become inconsistent. Only *adaptable functors* are able to work together with function adaptors. Unfortunately, the adapted functors type requirements come from special typedefs. If these typedefs are errounous ones, the adapted functor does not work perfectly.

In this paper we present our framework that aims at developing safe adaptable functors. One of the characteristics tested at runtime, some of them handled at compilation-time. This framework is based on the object-oriented and generative features of C++. Our aim is to develop a safe, efficient, multicore version of C++ STL in the future.

1. Introduction

The *C++ Standard Template Library* (STL) was developed by *generic programming* approach [3]. In this way containers are defined as class templates

Received by the editors: March 24, 2011.

2010 *Mathematics Subject Classification.* 68N15, 68N19.

1998 *CR Categories and Descriptors.* D.2 [**Software Engineering**]: D.2.5 Testing and Debugging – *STL functors*; D.3 [**Programming Languages**]: D.3.2 Language Classification – *C++*.

Key words and phrases. C++, STL, functors.

and many algorithms can be implemented as function templates. Furthermore, algorithms are implemented in a container-independent way [15], so one can use them with different containers [24]. C++ STL is widely-used because it is a very handy, standard C++ library that contains beneficial containers (like list, vector, map, etc.), large number of algorithms (like sort, find, count, etc.) among other utilities.

The STL was designed to be extensible. We can add new containers that can work together with the existing algorithms. On the other hand, we can extend the set of algorithms with a new one that can work together with the existing containers. Iterators bridge the gap between containers and algorithms [5]. The expression problem [26] is solved with this approach. STL also includes adaptor types which transform standard elements of the library for a different functionality [1].

However, the usage of C++ STL does not mean bugless or error-free code [8]. Contrarily, incorrect application of the library may introduce new types of problems [23].

One of the problems is that the error diagnostics are usually complex, and very hard to figure out the cause of a program error [27, 28]. Violating requirement of special preconditions (e.g. sorted ranges) is not tested, but results in runtime bugs [11, 19]. A different kind of stickler is that if we have an iterator object that pointed to an element in a container, but the element is erased or the container's memory allocation has been changed, then the iterator becomes *invalid* [9]. Another common mistake is related to removing algorithms. The algorithms are container-independent, hence they do not know how to erase elements from a container, just relocate them to a specific part of the container, and we need to invoke a specific erase member function to remove the elements phisically. Since, for example the `remove` algorithm does not actually remove any element from a container [14].

C++ STL is very efficient in a sequential realm, but it is not aware of multicore environment [4]. For example, the Cilk++ language aims at multicore programming. This language extends C++ with new keywords and one can write programs for multicore architectures easily. But the language does not consist of an efficient multicore library, but the C++ STL only which is an efficiency bottleneck in multicore environment. We develop a new STL implementation for Cilk++ to cope with the challenges of multicore architectures. This new implementation can be safer solution, too. Hence, our safety extensions will be included in the new implementation. However, the advised techniques presented in this paper concern to the original C++ STL, too.

Most of the properties are checked at compilation time. For example, the code does not compile if one uses sort algorithm with the standard list container, inasmuch as list's iterators do not offer random accessibility [12].

Other properties are checked at runtime. For example, the standard vector container offers an `at` method which tests if the index is valid and it raises an exception otherwise [18].

Unfortunately, there is still a large number of properties that are tested neither at compilation-time nor at run-time. Observance of these properties is in the charge of the programmers.

Functor objects make STL more flexible as they enable the execution of user-defined code parts inside the library. Basically, functors are usually simple classes with an `operator()`. Inside the library `operator()`s are called to execute user-defined code snippets. This can called a function via pointer to functions or an actual `operator()` in a class. Functors are widely used in the STL because they can be inlined by the compilers and they cause no runtime overhead in contrast to function pointers. Moreover, in case of functors extra parameters can be passed to the code snippets via constructor calls.

Functors can be used in various roles: they can define predicates when searching or counting elements, they can define comparison for sorting elements and properly searching, they can define operations to be executed on elements.

Associative containers (e.g. `multiset`) use functors exclusively to keep their elements sorted. Algorithms for sorting (e.g. `stable_sort`) and searching in ordered ranges (e.g. `lower_bound`) typically used with functors because of efficiency. These containers and algorithms need *strict weak ordering* [2].

A relation is strict weak ordering, if it is irreflexive, antisymmetric and transitive. A relation is irreflexive, if no element is related to itself. A relation is antisymmetric, if two elements are in relation and these two elements are in relation in reverse order means that two elements are the same. A relation is transitive if whenever an element a is related to an element b, and b is in turn related to an element c, then a is also related to c. For example, internal types' `operator<` is strict weak ordering relation in contrast to their `operator>=`, that are not strict weak ordering.

The rest of this paper is organized as follows. In section 2 we show the necessity of a framework to develop safe functors. We present our approach for testing functors at runtime in section 3. We detail how to develop safe adaptable functors with our framework in section 4. Finally, we conclude our results in section 5.

2. Motivation

In this section we present motivating examples. We detail the background of the problems because they are not evident.

All standard associative containers (`std::set`, `std::multiset`, `std::map`, `std::multimap`) keep elements ordered. These containers are template classes – among other template parameters they have a type parameter which stands for the type of comparison functor. That is, in this case it is not possible to pass function pointers. We must use functor types instead. From this type the containers create object to evaluate the comparison between elements. This parameter has default value which is `std::less<T>`, where T is the key type of the container. The functor must be strict weak ordering, but this property is not checked, so it is not difficult to write an erroneous one [16], for example:

```
struct Compare :
         std::binary_function<int, int, bool>
{
  bool operator()( int i, int j ) const
  {
    return !(i < j);
  }
};

struct StringLengthLess :
         std::binary_function<std::string, std::string, bool>
{
   bool operator()( const std::string& a,
                    const std::string& b ) const
   {
      return a.length() <= b.length();
   }
};
```

These functor types can be compiled, and they do not raise exception or assertation at runtime, too. On the other hand, neighter is strict weak ordering, especially as they do not meet the requirement of irreflexivity. The usage of these functors makes associative containers inconsistent:

```
std::set<int, Compare> sc;
sc.insert( 3 );
sc.insert( 3 );
// sc.size() == 2
// sc.count( 3 ) == 0

std::multiset<int, Compare> mc;
mc.insert( 7 );
// mc.count( 7 ) == 0
```

Many standard algorithms can be used for searching in ordered ranges, for instance `binary_search`, `lower_bound`, etc. All of these algorithms are overloaded and they can take a comparison functor as a parameter. These algorithms require strict weak ordering as well.

```
std::vector<int> v;
v.push_back( 4 );
v.push_back( 5 );
// ...
std::sort( v.begin(), v.end(), Compare() );

std::vector<int>:: iterator i =
  std::lower_bound( v.begin(), v.end(), 4, Compare() );

if ( i != v.end() )
{
  std::cout << "Not found";
}
```

When we use the erroneous functors, these algorithms cannot find elements in containers, hence they seem to be defective.

However, these examples are the most simplest ones. In case of more complex classes it is much easier to make a mistake.

As a general rule, two comparisons are used in the STL. One of them is called *equality*, and this is based on `operator==`. Algorithms, like `find` use equality when searching. Another one is called *equivalence* and this one is based on the relative ordering of object values in a sorted range. Two objects `x` and `y` have equivalent values with respect to the sort order if neither precedes the other in the sort order, so the following expression is evaluated: `!keycomp()(x,y) && !keycomp()(x,y)`, where `keycomp` is the type of the functor.

Let us consider what happens if the strict weak ordering requirement is violated, for example, the functor is based on `operator<=`. When elements are inserted into a `std::set`, the container has to examine if the element is already in the container. The container's `insert` method uses *equivalence*.

```
s.insert( 3 );
s.insert( 3 );
```

The second call of `insert` evaluates the following expression to check if value 3 is in the set:

```
!(3<=3) && !(3<=3)
```

The result of this expression is false, that can be interpreted as "3 is not equal to 3". This fact accounts for the previous strange behaviour of associative

containers. The container finds that 3 is not in the set, so it inserts once more. The set is not set anymore. The set's `count` member function cannot find any element that is equal to its parameter. The associative containers become inconsistent this way [pirkelbauer:runtime. This problem itself is not specific to C++, similar mistakes can be made in Java [21].

Adaptable functors are special functors which can be used with functor adaptors, such as `not1` or `bind2nd` [14]. Binder allow us to convert a binary function to a unary function, by binding one of the arguments to a given value (given at runtime). `not1` negates the unary predicate, `not2` negates the binary predicate, respectively.

Adaptable functors need some extra typedefs, and the adapted functors are generated based on these typedefs. The standard base type templates `unary_function` and `binary_function` are responsible to guarantee these necessary typedefs. Thus, the author of a functor class is responsible to make sure if the adapted works properly. It would be more elegant if the these typedefs come from the functor's `operator()`. If the template arguments of `unary_function` or `binary_function` disagree to the functor's `operator()` it may also results in compilation-time or run-time errors.

Let us consider the following predicate:

```
struct AnotherBadPredicate: std::unary_function<int, bool>
{
  double x;

  AnotherBadPredicate( const double& d ): x( d ) {}

  bool operator()( const double& a ) const
  {
    return a < x;
  }
};
```

This functor works perfectly, unless it is used without adaptors. However, if we have a vector, and we try to find the first element that is *not* less than a given value, we can negate the previous predicate:

```
std::vector<double> v;
v.push_back( 2.5 );
v.push_back( 8.3 );

std::vector<double>::iterator i =
  std::find_if( v.begin(),
                v.end(),
```

```
                    std::not1( AnotherBadPredicate( 2.3 ) );

if ( i != v.end() )
{
  std::cout << *i << std::endl;
}
```

This code snippets highlights that 8.3 is the first element in the vector which is not less than 2.3. This result is faulty because 2.5 is the first element that is not less than 2.5. However, the problem itself the template argument of base template class `std::unary_function<int, bool>` inasmuch as the generated negated predicate take the parameter as integer value. In the negated functor 2.5 as integer is 2 and it calls the original functor with this value. The original functor takes the parameter 2.0 as double value and `2.0 < 2.5` is true, so it returns true. The generated functor negates this result, thus `std::not1( AnotherBadPredicate( 2.3 )` return false when it takes 2.5 value. However, this root cause of this problem is the duplication. The author has to repeat the type of `operator()`'s argument types. If it disagrees it could make the negated functor erronous. The compiler should deduce the parameters of `operator()` to avoid this kind of problems. However, if the two different types cannot be converted it results in compilation errors. One of the aims of our framework is detect this problem at compilation time.

3. Safe functors

In this section we present our framework approach to test functors at runtime if they are strict weak ordering.

Our approach is based on object-orientation and inheritance because functors typically are inherited from `std::binary_function` or `std::unary_function` class templates, therefore this approach is plausible. Functor adaptors, such as `std::not2` to negate predicates, take advantage of some typedefs that comes from the base type. For instance, standard functor types are written this way. So we can take advantage of the automatic call of the base class's default constructor.

The type of functor is passed to the `strict_weak_ordering` class template as well as the type of parameter of its `operator()`. We force the base object to be the instance of subtype's class, which is the type of functor actually. We take advantage of `static_cast` which is able to do casts between pointer types. This is not a problem in this case because we designed this class to be superclass. We can call the functor's `operator()` this way. We assume, that the type `T` has default constructor. It is not serious restriction because most classes do have default constructor. Note, we cannot take advantage

of virtual functions or `dynamic_cast` because in constructors only the static type is available. Unfortunately, we cannot use `static_cast` without pointers because it would create a new object which has to be evaluated with our approach. This means an infinite recursion. Our approach is based on *curiously recurring template pattern (CRTP)* [1]. This pattern replaces the dynamic polymorphism with compile-time mechanism [7].

This way we create a test case that is evaluated when the functor object is constructed. If the testcase fails it throws an exception, otherwise, it does not mean necessarily that the functor itself is perfect. However, most of erronous functors fail on this test because typically the irreflexevity is violated, and our approach focuses on this characteristic. Moreover, specializations can be created for more complex testcases.

Only the `operator()` is called after a `static_cast` operator. This is quite reasonable and negligible overhead for safety functors [20].

So, First, we create a new exception type:

```
struct bad_functor_exception
{
  // ...
};
```

After we develop the essence of our approach:

```
template <class T, class functor_to_check>
struct strict_weak_ordering
{
  strict_weak_ordering()
  {
    if ( static_cast
           <functor_to_check*>( this )->
             operator()( T(), T() )
       )
    {
      throw bad_functor_exception();
    }
  }
};
```

The `strict_weak_ordering` template class is easy to use when one writes a new functor type. The new functor type must be inherited from the instantiated `strict_weak_ordering` class. This way this approach is non-intrusive.

```
struct Compare :
         std::binary_function<int, int, bool>,
         strict_weak_ordering<int, Compare>
```

```
{
  bool operator()( int i, int j ) const
  {
    // ...
  }
};
```

As mentioned before, specializations can be created for specific types. In the specializations more complex test cases can be evaluated. But if we increase the number of test cases the runtime overhead increases, too. Let us consider the following examples:

```
template <class functor_to_check>
struct strict_weak_ordering<int, functor_to_check>
{
  strict_weak_ordering()
  {
    functor_to_check* p = static_cast<functor_to_check*>( this );
    if ( p->operator()( 3,  3  ) ||
         p->operator()( 22, 22 ) )
    {
      throw bad_functor_exception();
    }
  }
};

template <class functor_to_check>
struct strict_weak_ordering<std::string, functor_to_check>
{
  strict_weak_ordering()
  {
    const std::string test = "Hello World";
    if ( static_cast
           <functor_to_check*>( this ) ->
             operator()( test, test ) )
    {
        throw bad_functor_exception();
    }
  }
};
```

External testing frameworks are also able to check these properties [17]. But external testing frameworks typically need external tools [6]. Our approach does not require any external tool, programmers have to design functors by inheritance which is straightforward as mentioned before. With our approach the average user of the library can write functors safely without any testing framework and difficult mathematical background. Our approach works perfectly automatically, too.

4. Adaptable functors

In this section we present how our framework can be used to write perfect adaptable functors.

Compilers cannot emit warnings based on the erroneous usage of the library. STLlint is the flagship example for external software that is able to emit warnings when the STL is used in an incorrect way [10]. We do not want to modify the compilers, so we have to enforce the compiler to indicate if an adaptable functor type is defective. However, `static_assert` as a new keyword is introduced in C++0x to emit compilation errors based on conditions, but no similar construct is designed for warnings.

```
template <class T>
inline void warning( T t )
{
}

struct IMPROPER_FUNCTOR_BASE
{
};

// ...

warning( IMPROPER_FUNCTOR_BASE() );
```

When the `warning` function is called, a dummy object is passed. This dummy object is not used inside the function template, hence this is an unused parameter. Compilers emit warning to indicate unused parameters. Compilation of `warning` function template results in warning messages, when it is referred and instantiated. No warning message is shown if it is not referred. In the warning message the template argument is referred.

Different compilers emit this warning in different ways. For instance, Visual Studio emits the following message:

```
warning C4100: 't' : unreferenced formal parameter
...
```

```
see reference to function template instantiation 'void
warning<IMPROPER_FUNCTOR_BASE>(T)'
being compiled

        with
        [
            T=IMPROPER_FUNCTOR_BASE
        ]
```

And g++ emits the following message:

```
In instantiation of 'void warning(T)
      [with T = IMPROPER_FUNCTOR_BASE]':
... instantiated from here
... warning: unused parameter 't'
```

Unfortunately, implementation details of warnings may differ, thus no universal solution to generate custom warnings.

This approach of warning generation has no runtime overhead inasmuch as the compiler optimizes the empty function body. On the other hand – as the previous examples show – the message refers to the warning of unused parameter, incidentally the identifier of the template argument type is appeared in the message.

C++ metaprogramming facilities are able to detect if the parameter types of the `operator()` suit to the template arguments of the base class [22]. They are not necessarily the same because references and constant references can be used as functor arguments, but in this case no references or constant references given as template arguments [14]. We can generate warnings with the previous approach [25].

First, we present the framework for unary functors that checks if the base type is proper for the `operator()`:

```
template<bool b, class Fun>
struct __WARNING
{
  __WARNING()
  {
    warning( IMPROPER_FUNCTOR_BASE() );
  }
};

template <class Fun>
struct __WARNING<true, Fun>
{
```

```
};

template <class Fun>
class __check_unary_adaptability
{
  typedef BOOST_TYPEOF(&Fun::operator()) f_type;

  typedef typename
    boost::mpl::at_c<
      boost::function_types::parameter_types<f_type>, 1>::type
    arg_type;

  __WARNING< boost::is_same<
      typename boost::remove_const<
        typename boost::remove_reference<arg_type>::type>::type,
      typename Fun::argument_type>::value, Fun > w;
};

#define CHECK_UNARY_FUNCTOR(F) __check_unary_adaptability<F>();
```

The utility class template `__WARNING` takes a compile-time boolean parameter, and if value of this parameter is true `__WARNING` does *not* generate warning. Otherwise it instantiates the warning generator function template, thus programmer gets a compilation warning. It is also takes the type of the functor to be presented in the generated error warning. The core class template is `__check_unary_adaptability` which extracts the type of parameter of the functor's `operator()` and named as `arg_type` [13]. It also retrieves the functor's inner typedef called `argument_type` which set by `unary_function`. Set type is template argument of `unary_function`. After that, it passes the condition of the two types are proper to the `__WARNING`. The proper parameter type means, that we should remove the `const` and `&` modifiers from the declaration of the parameter of `operator()`. Unfortunately, we have to start the verification manually, therefore a comfortable macro is present.

The following code snippets aims at the verification of binary functors:

```
template <class Fun>
class __check_binary_adaptability
{
  typedef BOOST_TYPEOF(&Fun::operator()) f_type;

  typedef typename
    boost::mpl::at_c<
```

```
      boost::function_types::parameter_types<f_type>, 1>::type
    arg1_type;

  typedef typename
    boost::mpl::at_c<
      boost::function_types::parameter_types<f_type>, 2>::type
    arg2_type;

  __WARNING< boost::is_same<
      typename boost::remove_const<
        typename boost::remove_reference<arg1_type>::type>::type,
      typename Fun::first_argument_type>::value, Fun > w1;

  __WARNING< boost::is_same<
      typename boost::remove_const<
        typename boost::remove_reference<arg2_type>::type>::type,
      typename Fun::second_argument_type>::value, Fun > w2;
};

#define CHECK_BINARY_FUNCTOR(F) __check_binary_adaptability<F>();
```

This class template is similar to the previous one, but this one uses `first_argument_type` and `second_argument_type` set by `binary_function`.

With our framework adaptable functors can be written safer because if the base class does not suit to the definition of `operator()` compilation warning is generated. The only limitation is the user of the framework has to start the verification manually. Our future work is to eliminate this limitation.

5. Conclusion

STL is widely-used standard C++ library based on the generic programming paradigm. STL increases efficacy of C++ programmers mightily because it consists of expedient containers and algorithms. On the other hand, improper application of the library results in undefined or strange behaviour.

Functors play an important role in the STL because they enable to execute user-defined code snippets in the library without significant overhead.

In this paper we detail a typical approach that results in a quite incomprehensible behaviour based on the functors' requirements. We show the background of the problem and argue for a non-intrusive approach as a plausible solution. Our solution has minimal overhead at runtime, but makes the usage of functors much safer.

Our framework also includes utilities that make the development of adaptable functors safer. Our approach is able to detect if the base class of the functor does not suit the functor at compilation time. Compilation warning is emitted, if a mistake is detected.

Acknowledgement

This research is supported by the European Union and co-financed by the European Social Fund (grant agreement no. TÁMOP 4.2.1./B-09/1/KMR-2010-0003).

References

[1] A. Alexandrescu, *Modern C++ Design*, Addison-Wesley, 2001.
[2] ANSI/ISO C++ Committee. Programming Languages – C++. ISO/IEC 14882:1998(E). American National Standards Institute, 1998.
[3] M. H. Austern, *Generic Programming and the STL: Using and Extending the C++ Standard Template Library,* Addison-Wesley, 1998.
[4] M. H. Austern, R. A. Towle, A. A. Stepanov, *Range partition adaptors: a mechanism for parallelizing STL*, in ACM SIGAPP Applied Computing Review 1996 **4(1)**, pp. 5–6,
[5] T. Becker, *STL & generic programming: writing your own iterators*, C/C++ Users Journal 2001 **19(8)**, pp. 51–57.
[6] M. Biczó, K. Pócza, Z. Porkoláb, I. Forgács, *A new concept of effective regression test generation in a C++ specific environment*, In Acta Cybern., **18** (2008), pp. 481–512.
[7] K. Czarnecki, U. W. Eisenecker, *Generative Programming: Methods, Tools and Applications,* Addison-Wesley, 2000.
[8] G. Dévai, N. Pataki, *A tool for formally specifying the C++ Standard Template Library*, In Ann. Univ. Sci. Budapest., Comput. **31**, pp. 147–166.
[9] G. Dévai, N. Pataki, *Towards verified usage of the C++ Standard Template Library*, In Proc. of The 10th Symposium on Programming Languages and Software Tools (SPLST) 2007, pp. 360–371.
[10] D. Gregor, S. Schupp, *Stllint: lifting static checking from languages to libraries*, Software - Practice & Experience, **36(3)** (2006) 225-254.
[11] D. Gregor, J. Järvi, J. Siek, B. Stroustrup, G. Dos Reis, A. Lumsdaine, *Concepts: linguistic support for generic programming in C++*, in Proc. of the 21st annual ACM SIGPLAN conference on Object-oriented programming systems, languages, and applications (OOPSLA 2006), pp. 291–310.
[12] J. Järvi, D. Gregor, J. Willcock, A. Lumsdaine, J. Siek, *Algorithm specialization in generic programming: challenges of constrained generics in C++*, in Proc. of the 2006 ACM SIGPLAN conference on Programming language design and implementation (PLDI 2006), pp. 272–282.
[13] B. Karlsson: *Beyond the C++ Standard Library: An Introduction to Boost,* Addison-Wesley, 2005.
[14] S. Meyers, *Effective STL*, Addison-Wesley, 2003.
[15] D. R. Musser, A. A. Stepanov, *Generic Programming*, in Proc. of the International Symposium ISSAC'88 on Symbolic and Algebraic Computation, Lecture Notes in Comput. Sci., **358** 1988, pp. 13–25.

[16] N. Pataki, *C++ Standard Template Library by Safe Functors*, in Proc. of The 8-th Joint Conference on Mathematics and Computer Science, Selected Papers, pp. 357–368.

[17] N. Pataki, *Testing by C++ Template Metaprograms*, in Acta Univ. Sapientiae, Inform., **2**(2), pp. 154–167.

[18] N. Pataki, Z. Porkoláb, Z. Istenes, *Towards Soundness Examination of the C++ Standard Template Library*, In Proc. of Electronic Computers and Informatics, ECI 2006, pp. 186–191.

[19] N. Pataki, Z. Szűgyi, G. Dévai, *C++ Standard Template Library in a Safer Way*, In Proc. of Workshop on Generative Technologies 2010 (WGT 2010), pp. 46–55.

[20] P. Pirkelbauer, S. Parent, M. Marcus, B. Stroustrup, *Runtime Concepts for the C++ Standard Template Library*, In Proc. of the 2008 ACM Symposium on Applied Computing, pp. 171–177.

[21] D. Rayside, Z. Benjamin, R. Singh, J. P. Near, A. Milicevic, D. Jackson, *Equality and hashing for (almost) free: Generating implementations from abstraction functions*, In Proceedings of the 31st IEEE International Conference on Software Engineering (ICSE 2009) pp. 342–352.

[22] Z. Porkoláb, *Functional Programming with C++ Template Metaprograms* in Proc. of Central European Functional Programming School, Revised Selected Lectures, Lecture Notes in Comput. Sci., **6299**, pp. 306–353.

[23] Z. Porkoláb, Á. Sipos, N. Pataki, *Inconsistencies of Metrics in C++ Standard Template Library*, In Proc. of 11th ECOOP Workshop on Quantitative Approaches in Object-Oriented Software Engineering QAOOSE Workshop, ECOOP 2007, Berlin, pp. 2–6.

[24] B. Stroustrup, *The C++ Programming Language - Special Edition*, Addison-Wesley, 2000.

[25] Z. Szűgyi, Á. Sinkovics, N. Pataki, Z. Porkoláb, *C++ Metastring Library and its Applications*, In Proc. of Generative and Transformational Techniques in Software Engineering 2009, Lecture Notes in Comput. Sci., **6491**, pp. 461–480.

[26] M. Torgersen, *The Expression Problem Revisited – Four New Solutions Using Generics*, in Proc. of European Conference on Object-Oriented Programming (ECOOP) 2004, Lecture Notes in Comput. Sci., **3086**, pp. 123–143.

[27] L. Zolman, *An STL message decryptor for visual C++*, In C/C++ Users Journal, 2001 **19(7)**, pp. 24–30.

[28] I. Zólyomi, Z. Porkoláb, *Towards a General Template Introspection Library*, in Proc. of Generative Programming and Component Engineering: Third International Conference (GPCE 2004), Lecture Notes in Comput. Sci., **3286**, pp. 266–282.

Dept. of Programming Languages and Compilers, Fac. of Informatics, Eötvös Loránd University, Pázmány Péter sétány 1/C, H-1117 Budapest, Hungary

E-mail address: patakino@elte.hu