Advanced Studies in Biology, Vol. 4, 2012, no. 2, 95 - 101

On the Optimal Reconfiguration Planning for Modular Self-Reconfigurable DNA Nanomechanical Robots

Anna Gorbenko

Department of Intelligent Systems and Robotics
Ural Federal University
620083 Ekaterinburg, Russia
gorbenko.ann@gmail.com

Vladimir Popov

Department of Intelligent Systems and Robotics
Ural Federal University
620083 Ekaterinburg, Russia
Vladimir.Popov@usu.ru

Abstract

In this paper we describe an approach to solve the optimal reconfiguration planning problem of finding the least number of reconfiguration steps to transform between two configurations for chain-type modular self-reconfigurable DNA nanomechanical robots. Our approach is based on constructing logical models for considered problem.

Keywords: self-reconfigurable modular robots, reconfiguration planning, logical models, **NP**-complete, genetic algorithms

Self-reconfiguring robots were first proposed in [1]. They intensively studied (see e.g. [2] and references). In particular, problems of planning of self-reconfiguration received considerable attention (see e.g. [3] and references). In [3] considered the optimal reconfiguration planning problem of finding the least number of reconfiguration steps to transform between two configurations for chain-type modular robots. In the case where robot not necessarily keeps connected throughout the reconfiguration process in [3] proved that the problem is **NP**-complete. DNA, beyond its crucial biological role, has established itself as an important building block for self-assembly in nanotechnology, with

numerous nanodevices being reported over the past decade (see e.g. [4]). It should be noted that recent progress indicates that the field is now heading beyond purely structural elements and towards complex functional systems (see e.g. [5]). For DNA nanomechanical devices it is preferred that the robot keeps connected throughout the reconfiguration process (see e.g. [5], see also [3]). If the robot keeps connected throughout the reconfiguration process, then the problem is also **NP**-complete. To prove this fact, a slightly modified version of the idea from [3] can be used. In this paper we suppose that the robot keeps connected throughout the reconfiguration process and describe an approach to solve this problem. This approach is based on constructing logical models for considered problem (see [4]).

We give only formal definitions (detailed description of the problem and examples see e.g. in [3]). A chain-type modular robot in given configuration can be considered as C-Graph (see e.g. [3]) $G = (V, E)$, $V = \{v[1], \ldots, v[n]\}$, $E = \{e[1], \ldots, e[m]\}$, where each node $v[i] \in V$ represents the set $v[i] = \{v[i, 1], \ldots, v[i, p_i]\}$ of connecting points of ith module, where p_i is the number of connecting points of ith module; each edge $e[j] = (v[i_1, l_1], v[i_2, l_2]) \in E$ represents a connection between module i_1's connector l_1 and module i_2's connector l_2, where $1 \leq i_1 \leq n$, $1 \leq i_2 \leq n$, $1 \leq l_1 \leq p_{i_1}$, $1 \leq l_2 \leq p_{i_2}$. For given two connected C-Graphs $I = (V, E_1)$ and $G = (V, E_2)$ we say that there exists a reconfiguration plan with at most k reconfiguration steps (see e.g. [3]) if and only if there exists a sequence of $r \leq k + 1$ connected C-Graphs $G_1 = (V, F_1), \ldots, G_r = (V, F_r)$ such that $F_1 = E_1$, $F_r = E_2$, and $|(F_{i+1} \backslash F_i) \cup (F_i \backslash F_{i+1})| = 1$.

The decision version of the optimal reconfiguration planning problem of finding the least number of reconfiguration steps to transform between two configurations for chain-type modular robots is formulated as the following problem (see [3]).

Optimal reconfiguration planning problem (ORP):

Instance: *C-Graphs $I = (V, E_1)$ and $G = (V, E_2)$, a given integer k.*

Question: *Whether there exists a reconfiguration plan for C-Graphs I and G with at most k reconfiguration steps?*

Satisfiability is a core problem in mathematical logic and computing theory. Different variants of satisfiability were considered. PSAT: given a boolean expression, is it satisfiable? SAT: given a boolean expression in conjunctive normal form (CNF), is it satisfiable? 3SAT: given a boolean expression in CNF with 3 variables per clause (3-CNF), is it satisfiable? MAXkSAT: given a boolean expression in CNF, such that each clause contains at most k literals, find an assignment to the variables such that a maximum number of clauses is satisfied. EXACT MAXkSAT: given a boolean expression in CNF, such that each clause contains k literals, find an assignment to the variables such that a maximum number of clauses is satisfied. Encoding problems as PSAT

and its variants and solving them with very efficient satisfiability algorithms has recently caused considerable interest (see e.g. [6, 7, 8, 9]).

Let $\varphi[q] \rightleftharpoons \wedge_{1\leq i_1\leq n} \vee_{1\leq i_2\leq p_{i_1}, 1\leq i_3\leq n, 1\leq i_4\leq p_{i_3}} x[q, i_1, i_2, i_3, i_4]$. Clearly, boolean function $\varphi[q]$ is satisfiable if and only if C-Graph $G[q]$ is connected. In the case where robot not necessarily keeps connected throughout the reconfiguration process there are reductions from ORP to PSAT and SAT (see [2]). If the robot keeps connected throughout the reconfiguration process, then using essentially the same arguments and boolean function $\varphi[q]$ we can obtain reductions from ORP to different variants of satisfiability.

We use algorithms fgrasp and posit from [10]. Also we design our own genetic algorithm for SAT which based on algorithms from [10].

Consider a boolean function $g(x_1, x_2, \ldots, x_n) = \wedge_{i=1}^{m} \mathcal{C}_i$, where $m \geq 1$, and each of the $\mathcal{C}_i$ is the disjunction of one or more literals. Let $|\mathcal{C}_i|$ be a number of literals in $\mathcal{C}_i$. Let $occ(x_i, g)$ be a number of occurrences of x_i in g. Respectively, let $occ(\neg x_i, g)$ be a number of occurrences of x_i in g. For example, if $g = (x_1 \vee x_2) \wedge (\neg x_2 \vee x_3) \wedge (x_1 \vee x_4) \wedge (\neg x_1 \vee x_5)$, then $occ(x_1, g) = 2$, $occ(\neg x_1, g) = 1$.

We consider a number of natural principles that define importance of a variable x_i for satisfiability of boolean function g. These principles suggest us correct values of variables.

- If $occ(x_i, g) \geq 0$ and $occ(\neg x_i, g) = 0$, then $x_i = 1$.
- If $occ(x_i, g) = 0$ and $occ(\neg x_i, g) \geq 0$, then $x_i = 0$.
- If $occ(x_i, g) > occ(\neg x_i, g)$, then $x_i = 1$.
- If $occ(x_i, g) < occ(\neg x_i, g)$, then $x_i = 1$.
- If $x_i = \mathcal{C}_j$ for some j, then $x_i = 1$.
- If $\neg x_i = \mathcal{C}_j$ for some j, then $x_i = 0$.
- If $\min_{occ(x_i, \mathcal{C}_j)>0} |\mathcal{C}_j| \leq \min_{occ(\neg x_i, \mathcal{C}_j)>0} |\mathcal{C}_j|$, then $x_i = 1$.
- If $\min_{occ(x_i, \mathcal{C}_j)>0} |\mathcal{C}_j| \geq \min_{occ(\neg x_i, \mathcal{C}_j)>0} |\mathcal{C}_j|$, then $x_i = 0$.
- Given positive integers $p_1, p_2, \ldots, p_m$. If

$$\Sigma_{1\leq j\leq m, |\mathcal{C}_j|\leq p_j} occ(x_i, \mathcal{C}_j) \geq \Sigma_{1\leq j\leq m, |\mathcal{C}_j|\leq p_j} occ(\neg x_i, \mathcal{C}_j),$$

then $x_i = 1$.

- Given positive integers $p_1, p_2, \ldots, p_m, q_1, q_2, \ldots, q_m$. If

$$\Sigma_{1\leq j\leq m, |\mathcal{C}_j|\leq p_j} occ(x_i, \mathcal{C}_j) \geq \Sigma_{1\leq j\leq m, |\mathcal{C}_j|\leq q_j} occ(\neg x_i, \mathcal{C}_j),$$

then $x_i = 1$.

- Given positive integers $p_1, p_2, \ldots, p_m, q_1, q_2, \ldots, q_m$ and a set of rational numbers $\{\alpha_{i,u}, \beta_{i,v} \mid 1 \leq i \leq m, 1 \leq u \leq p_i, 1 \leq v \leq q_i\}$. If

$$\Sigma_{1\leq j\leq m, 1\leq u\leq p_j, |\mathcal{C}_j|=u} \alpha_{j,u} occ(x_i, \mathcal{C}_j) \geq \Sigma_{1\leq j\leq m, 1\leq v\leq q_j, |\mathcal{C}_j|=v} \beta_{j,v} occ(\neg x_i, \mathcal{C}_j),$$

then $x_i = 1$.

Based on these principles, we can consider the following eleven types of commands: $P_1, \ldots, P_8, P_9[p_1, \ldots, p_m]$, $P_{10}[p_1, \ldots, p_m, q_1, \ldots, q_m]$, $P_{11}[p_1, \ldots, p_m, q_1, \ldots, q_m, \{\alpha_{i,u}, \beta_{i,v} \mid 1 \leq i \leq m, 1 \leq u \leq p_i, 1 \leq v \leq q_i\}]$. Also we consider the following three commands for run algorithms: Try_fgrasp, Try_posit, and Try_ga, where Try_ga runs a simple genetic algorithm. Denote by $\mathcal{R}$ the set of commands of these fourteen types. Arbitrary element of $\mathcal{R}^*$ it is possible to consider as a program for finding values of variables of a boolean function. We assume that such programs are executed on a cluster. Execution of each of commands of type P_i reduces the number of variables of a boolean function by one. Execution of each of commands Try_fgrasp, Try_posit, and Try_ga consists in the run of corresponding algorithm for current boolean function on a separate set of calculation nodes and the transition to the next command. Algorithms fgrasp and posit we run only on one calculation node. Genetic algorithms can be used in parallel execution. We use auxiliary genetic algorithms which determine the number of calculation nodes and, if necessary, scale commands P_9, P_{10}, P_{11}.

Initially, we selected a random subset of $\mathcal{R}^*$. We use a genetic algorithm to select a program from the current subset of $\mathcal{R}^*$ and a genetic algorithm for evolving the current subset of $\mathcal{R}^*$. The evolution of the current subset of $\mathcal{R}^*$ implemented on a separate set of calculation nodes. For every subsequent boolean functions it is used the current subset of $\mathcal{R}^*$ which is obtained by taking into account the results of previous runs.

We use heterogeneous cluster based on three clusters (Cluster USU, Linux, 8 calculation nodes, Intel Pentium IV 2.40GHz processors; umt, Linux, 256 calculation nodes, Xeon 3.00GHz processors; um64, Linux, 124 calculation nodes, AMD Opteron 2.6GHz bi-processors) [11].

We create a generator of special hard and natural instances for ORP. For further computational experiment we create special hard test sets, natural test sets and special DNA nanotechnology test sets. Special hard test sets based on ideas from [12]. Natural test sets based on ideas from [3]. Nearly all structures in DNA nanotechnology make use of branched DNA structures containing junctions, as opposed to most biological DNA which exists in a linear double helix form. One of the simplest branched structures, and the first made, is a four-arm junction which can be made using four individual DNA strands which are complementary to each other in the correct pattern. Unlike in natural Holliday junctions, in the artificial immobile four-arm junction shown below, the base sequence of each arm is different, meaning that the junction point is fixed in a certain position [13], [14]. DNA nanostructures must be designed so that they will assemble into the desired structures. This includes both the design of secondary structure, deciding which parts of which nucleic acid molecules should bind to each other, and primary structure, the specification of the identity of each individual base. Special DNA nanotechnology test sets

based on ideas from [15], [16]. In this sets we try to solve problems of the design of secondary structure of DNA nanostructures. In particular, in special DNA nanotechnology test sets we consider C-Graphs with $|V| \geq 10^5$.

Algorithms fgrasp and posit used only for 3SAT. For natural test sets and special hard test sets used fgrasp, posit, simple genetic algorithm (SGA), and our algorithm (OA). For special DNA nanotechnology test sets we failed to wait for the completion of algorithms fgrasp and posit. Note that for special DNA nanotechnology test sets in some cases we obtain time from day to week. So, our algorithms are relatively good but we need much more performance to work with practical problems of DNA nanotechnology.

Selected experimental results are given in Figures 1 – 4 ($V = 100$).

	fgrasp	posit	SGA	OA
average time	24.7 min	23.6 min	25.1 min	14.8 min
maximum time	16.2 h	16.4 h	19.2 h	12 h
best time	5.7 min	3.2 min	1.4 min	1.2 min

Figure 1: Experimental results for natural test set (3SAT).

	SGA (PS)	OA (PS)	SGA (S)	OA (S)
average time	29.3 min	14.5 min	28.7 min	14.7 min
maximum time	34.9 h	11.4 h	27.1 h	11.8 h
best time	43 sec	1.4 min	59 sec	1.3 min

Figure 2: Experimental results for natural test set (PSAT and SAT).

	SGA (M)	OA (M)	SGA (EM)	OA (EM)
average time	22.1 min	14.4 min	21.9 min	13.2 min
maximum time	15.3 h	11.9 h	15.4 h	10.3 h
best time	36 sec	56 sec	39 sec	1.1 min

Figure 3: Experimental results for natural test set (MAX2SAT and EXACT MAX2SAT).

In this paper we describe an approach to solve ORP problem. This approach is based on constructing logical models for considered problem. Interesting area of future research involves finding an efficient algorithm to search for periodic patterns in genetic sequences [17, 18, 19]. Such algorithm will allow to significantly reduce the size of Boolean functions.

	fgrasp	posit	SGA	OA
average time	1.2 h	1.1 h	2.8 h	45.7 min
maximum time	22.3 h	21.7 h	46.9 h	19 h
best time	11.4 min	9.3 min	4.9 min	4.6 min

Figure 4: Experimental results for special hard test sets (3SAT).

References

[1] T. Fukuda and S. Nakagawa, A Dynamically Reconfigurable Robotic System (Concept of a System and Optimal Configurations), *Proceedings of the 1987 IEEE International Conference on Industrial Electronics, Control, and Instrumentation*, (1987), 588-595.

[2] A. Gorbenko and V. Popov, Programming for Modular Reconfigurable Robots, *Programming and Computer Software*, 38 (2012), 13-23.

[3] F. Hou, W.-M. Shen, On the Complexity of Optimal Reconfiguration Planning for Modular Reconfigurable Robots, *Proceedings of 2010 IEEE International Conference on Robotics and Automation*, (2010), 2791-2796.

[4] S. Murata and M. Stojanovic, DNA-based Nanosystems, *New Generation Computing*, 26 (2008), 297-312.

[5] P. Yin, A. Turberfield, S. Sahu, and J. Reif, Design of an autonomous DNA nanomechanical device capable of universal computation and universal translational motion, *Tenth International Meeting on DNA Computing*, (2005), 426-444.

[6] A. Gorbenko, M. Mornev, and V. Popov, Planning a Typical Working Day for Indoor Service Robots, *IAENG International Journal of Computer Science*, 38 (2011), 176-182.

[7] A. Gorbenko, M. Mornev, V. Popov, and A. Sheka, The problem of sensor placement for triangulation-based localisation, *International Journal of Automation and Control*, 5 (2011), 245-253.

[8] A. Gorbenko and V. Popov, On the Problem of Placement of Visual Landmarks, *Applied Mathematical Sciences*, 6 (2012), 689-696.

[9] A. Gorbenko, V. Popov, and A. Sheka, Localization on Discrete Grid Graphs, *Proceedings of the CICA 2011*, (2012), 971-978.

[10] http://people.cs.ubc.ca/∼hoos/SATLIB/index-ubc.html

[11] http://parallel.imm.uran.ru/mvc_now/hardware/supercomp.htm

[12] J. Navarro and A. Voronkov, Generation of Hard Non-Clausal Random Satisfiability Problems, *Proceedings of the Twentieth National Conference on Artificial Intelligence*, (2005), 436-442.

[13] N. Seeman, Nanotechnology and the double helix, *Scientific American*, 290 (2004), 64-75.

[14] P. Rothemund, Folding DNA to create nanoscale shapes and patterns, *Nature*, 440 (2006), 297-302.

[15] C. Mao, W. Sun, and N. Seeman, Designed Two-Dimensional DNA Holliday Junction Arrays Visualized by Atomic Force Microscopy, *Journal of the American Chemical Society*, 121 (1999), 5437-5443.

[16] V. Popov, *DNA nanomechanical devices: Y-sites*, Ekaterinburg, Business Partner, 2010. (in russian)

[17] J. Sim, C. Iliopoulos, K. Park, and W. Smyth, Approximate periods of strings, *Theoretical Computer Science*, 262 (2001), 557-568.

[18] V. Popov, The approximate period problem for DNA alphabet, *Theoretical Computer Science*, 304 (2003), 443-447.

[19] V. Popov, The Approximate Period Problem, *IAENG International Journal of Computer Science*, 36 (2009), 268-274.

Received: January, 2012