Minimum maximum reconfiguration cost problem

Raouf Senhadji-Navarro · Ignacio Garcia-Vargas

Abstract This paper discusses the problem of minimizing the reconfiguration cost of some types of reconfigurable systems. A formal definition of the problem and a proof of its NP-completeness are provided. In addition, an Integer Linear Programming formulation is proposed. The proposed problem has been used for optimizing a design stage of Finite Virtual State Machines.

Keywords Combinatorial optimization · Computational complexity · NP-completeness · Graph · Integer Linear Programming · Reconfigurable computing

1 Introduction

The concept of reconfigurable computing goes back to 1960 when Gerald Estrin proposed a computer made of a standard processor and an array "reconfigurable" hardware [3]. However, the lack of adequate reprogrammable logic chips has delayed its devel-opment for more than three decades. Nowadays, the increasing performance (in terms of logic density, speed and power consumption) of the current programmable devices, like Field Programmable Gate Arrays (FPGAs), has allowed the development of a wide range of reconfigurable applications.

The conventional computing architectures based on application specific integrated circuits and programmable general purpose processors suffer for the lack of flexibility of the dedicated hardware and limited performance of the software solutions, respec-

tively. Reconfigurable computing is a promising approach to overcome the traditional trade-off between flexibility and performance [5,7].

However, there are some challenges that must be overcome to develop efficient applications for reconfigurable computing systems. One of these challenges is the lack of formal models and methodologies for design automation of dynamically reconfigurable systems [1,2]. One of the most critical problems that the designer faces is to minimize the reconfiguration cost with the goal to achieve an optimal performance [10].

In this paper, we have modeled the problem of minimizing the reconfiguration cost of some types of reconfigurable systems. We prove that this problem is NP-complete. We also propose an Integer Linear Programming (ILP) formulation for this problem. We have used the proposed problem for optimizing a design stage of Finite Virtual State Machines (FVSMs) [11].

The remainder of this letter is organized as follows. We provide a formal definition of the proposed problem in Sect. 2 and its NP-Completeness is proved in Sect. 3. The Integer Linear Programming (ILP) formulation is presented in Sect. 4. Experimental results of the ILP formulation is shown in Sect. 5. Finally, a practical application is shown in Sect. 6.

2 Problem statement

A reconfigurable system is composed of a set of reconfigurable elements (REs) whose functional behavior can be changed in order to adapt the functionality of the system to different execution contexts. Let us call *functionality* the functional behavior that can be implemented in a single RE. Each execution context can be modeled as a set of functionalities. The functionalities implemented by all REs of the system for an execution context is called an *instance* of the reconfigurable system. Given a reconfigurable system with n REs, the set of instances related to the different execution contexts is called *n-implementation*. The changes of execution contexts can be modeled as an undirected graph, called *execution context graph*, where vertices represent execution contexts and edges connect execution contexts adjacent in time.

We define an *Homogeneous Reconfigurable System* (HRS) as a reconfigurable system composed by a set of identical REs that verify the following three properties. First, all REs can implement the same functionalities. Second, the behavior of the HRS only depends on the functionalities of their REs but not on the particular assignment of functionalities to REs. Third, those REs that are not required by a particular execution context can implement any functionality without affecting the behavior of the HRS.

In a HRS, when the execution context changes, those REs that already implement a functionality required by the new execution context do not need to be reconfigured. So, the reconfiguration cost is calculated as the number of REs that must be modified. The reconfiguration cost in the worst case determines the performance of the HRS; so, the maximum reconfiguration cost must be minimized. When an execution context requires less REs than available, the reconfiguration cost can be reduced if the unused REs are exploited for implementing some functionalities of the next execution context.

A classical reconfigurable computing architecture composed by a general-purpose processor and a set of identical Coarse-Grained Reconfigurable Processing Elements

(CGRPEs) connected by a bus is a HRS. The general-purpose processor also acts as reconfiguration controller [7]. FPGA devices (or different areas of a same FPGA device when dynamic partial reconfiguration [8] is used) are classical examples of CGRPE. Computer applications require the reconfiguration of CGRPEs for accelerating computationally intensive tasks. For this purpose, CGRPEs implement different functionalities such as digital filter banks, floating-point coprocessors, cryptographic coprocessors, etc. The functionalities required by the application in each period of time are the different execution contexts. In each execution context, the general-purpose processor reconfigures and uses the needed CGRPEs. The reconfiguration process requires to transfer the reconfiguration data of the modified CGRPEs. The reconfiguration latency (i.e., the transfer time of the reconfiguration data) directly depends on the number of modified CGRPEs. In reconfigurable real-time applications, the reduction of the reconfiguration latency is key to improve the response time of the system. In order to meet timing constraints, the reconfiguration latency in the worst case must be minimized.

A HRS is modeled as a 3-tuple $(\mathcal{F}, G, R)$ where $\mathcal{F}$ is a set of functionalities; $G = (V, E)$, an execution context graph; and $R = \{R_1, R_2, \ldots, R_{|V|}\}$, a collection of execution contexts where $R_j \subseteq \mathcal{F}$ for all $R_j \in R$. Each vertex $v_j \in V$ represents the execution context R_j. Let $S = (\mathcal{F}, G, R)$ be a HRS. Let us define an n-implementation of S as a collection of instances $I = \{I_1, I_2, \ldots, I_{|V|}\}$ where $I_j \subseteq \mathcal{F}$ such that $R_j \subseteq I_j$ and $|I_j| = n$ for all $I_j \in I$. Given an n-implementation I, let us define the *reconfiguration cost* between two instances $I_i, I_j \in I$ as $\delta(I_i, I_j) = n - |I_i \cap I_j|$. Note that $\delta(I_i, I_j)$ represents the number of functionalities in which I_i differs from I_j (or I_j from I_i). Let us define the *maximum reconfiguration cost* of I as $\Delta(I) = \max_{\{v_i, v_j\} \in E} \delta(I_i, I_j)$. The Minimum Maximum Reconfiguration Cost (MMRC) problem consists in finding an n-implementation I of a HRS with a minimum $\Delta(I)$.

3 NP-completeness

In order to proof the NP-completeness of the MMRC problem, we formulate the related decision problem as follows.

MINIMUM MAXIMUM RECONFIGURATION COST DECISION PROBLEM (MMRCDP)
INSTANCE: Reconfigurable system $S = (\mathcal{F}, G, R)$, positive integer n, positive integer $K \leq n$.
QUESTION: Is there an n-implementation of S with a maximum reconfiguration cost less or equal to K?

It is easy to see that MMRCDP $\in NP$ since a nondeterministic algorithm needs only to guess an n-implementation and check in polynomial time if its maximum reconfiguration cost is less or equal to K. We prove that MMRCDP $\in$ NP-complete by a reduction from 3SAT [4,9]. The 3SAT can be enunciated as follows.

3- SATISFIABILITY (3SAT)
INSTANCE: Collection $C = \{c_1, c_2, \ldots c_m\}$ of clauses on a finite set U of boolean variables such that $|c_i| = 3$ for $1 \leq i \leq m$.
QUESTION: Is there a truth assignment for U that satisfies all clauses in C ?

Given an arbitrary instance of 3SAT with a collection $C = \{c_1, c_2, \ldots c_m\}$ of clauses on a finite set U of boolean variables, we shall construct a HRS $S = (\mathcal{F}, G, R)$ such that there exists a 4-implementation of S with a maximum reconfiguration cost less or equal to $K = 2$ if and only if C is satisfiable. We construct the HRS by performing the following steps:

1. Given a collection of clauses C, let us define $H(C)$ as a maximal subset of clauses of C which verifies that no pair of clauses of $H(C)$ are defined over exactly the same set of variables. Note that the clauses in $H(C)$ can not share three variables. Let us define $h : C \to \mathbb{Z}^+$ in such a way that for any $c_k \in C$, $c_{h(c_k)}$ is the clause of $H(C)$ defined over exactly the same variables that c_k.
2. Let us define the set of functionalities $\mathcal{F} = \{X, Y, Z\} \cup \bigcup_{i=1}^{|U|} \{T_i, F_i\}$. The functionalities T_i and F_i are called *truth functionalities* of the variable u_i and represent a truth-setting for u_i where T_i and F_i denote the setting of true (T) and false (F) values, respectively. The functionalities X, Y, and Z will be used to impose restrictions on the reconfiguration cost between instances.
3. Let us construct the collection of execution contexts R as follows:
   (a) An execution context R_j^V is created for each $c_j \in C$. Given a clause $c_j = \{l_1, l_2, l_3\} \in C$ where l_i is a literal over $u_{w_i} \in U$ for $i = 1, 2, 3$, let us define $R_j^V = \{X, f_1, f_2, f_3\}$, where

$$f_i = \begin{cases} T_{w_i} & \text{if} \quad l_i = u_{w_i} \\ F_{w_i} & \text{if} \quad l_i = \bar{u}_{w_i} \end{cases} \tag{1}$$

   Note that R_j^V contains the truth functionalities that represent the truth-setting for the variables of c_j that satisfies the literals of c_j.
   (b) Execution contexts $R_j^{C_1}$, $R_j^{C_2}$, and $R_j^{C_3}$ are created for each $c_j \in H(C)$. Given a clause $c_j = \{l_1, l_2, l_3\} \in H(C)$, where l_i is a literal over $u_{w_i} \in U$ for $i = 1, 2, 3$, let us define $R_j^{C_k} = \{T_{w_1}, F_{w_1}, T_{w_2}, F_{w_2}, T_{w_3}, F_{w_3}\} \backslash \{T_{w_k}, F_{w_k}\}$ for $k = 1, 2, 3$.
   (c) An execution context $R_j^U = \{X\}$ is created for each $c_j \in H(C)$. In any 4-implementation I of S with $\Delta(I) \leq 2$, the instance related to R_j^U will contain the truth functionalities that represent a truth-setting for the variables of c_j that satisfies C.
   (d) Execution contexts $R_{j,k}^U$ and $R_{j,k}^C$ are created for each $\{c_j, c_k\} \in H(C) \times H(C)$ with $j \neq k$ such that c_j and c_k share two variables. Let u_p and u_q be the shared variables. Let us define $R_{j,k}^U = \{Y, Z\}$ and $R_{j,k}^C = \{T_p, F_p, T_q, F_q\}$. In any 4-implementation I of S with $\Delta(I) \leq 2$, the instance related to $R_{j,k}^U$ will contain the truth functionalities that represent a truth-setting for the shared variables that satisfies C.
4. Let us construct the execution context graph $G = (V, E)$ as follows. For each created execution context, a vertex $v \in V$ is created. The vertices denoted by V_j^V, $V_j^{C_1}$, $V_j^{C_2}$, $V_j^{C_3}$, V_j^U, $V_{j,k}^U$, and $V_{j,k}^C$ represent the execution contexts R_j^V, $R_j^{C_1}$, $R_j^{C_2}$, $R_j^{C_3}$, R_j^U, $R_{j,k}^U$, and $R_{j,k}^C$, respectively. The set of edges E is created as follows:

(a) Edges $\{V_j^{C_1}, V_j^U\}$, $\{V_j^{C_2}, V_j^U\}$ and $\{V_j^{C_3}, V_j^U\}$ are created for each $c_j \in H(C)$. In any 4-implementation I of S with $\Delta(I) \leq 2$, these edges allow to ensure that the instance related to R_j^U will contain a truth functionality of each variable of c_j.

(b) An edge $\{V_j^U, V_k^U\}$ is created for each $\{c_j, c_k\} \in H(C) \times H(C)$ with $j \neq k$ such that c_j and c_k share exactly one variable. In any 4-implementation I of S with $\Delta(I) \leq 2$, this edge allows to ensure that the instances related to R_j^U and R_k^U will contain the same truth functionality of the shared variable.

(c) Edges $\{V_j^U, V_{j,k}^U\}$, $\{V_{j,k}^U, V_k^U\}$, and $\{V_{j,k}^U, V_{j,k}^C\}$ are created for each $R_{j,k}^U \in R$. In any 4-implementation I of S with $\Delta(I) \leq 2$, these edges allow to ensure that the instances related to R_j^U and R_k^U will contain the same truth functionalities of the two shared variables (note that $R_{j,k}^U$ is created only if c_j and c_k share two variables).

(d) For each $c_j \in C$ with $h(c_j) = k$, an edge $\{V_j^V, V_k^U\}$ is created. In any 4-implementation I of S with $\Delta(I) \leq 2$, this edge allows to ensure that instance related to R_k^U will contain at least one truth functionality that represent a truth-setting that satisfies c_j.

Figure 1 shows an example of the proposed transformation. Figure 1a shows the given instance of 3SAT; Fig. 1b, the execution contexts (steps from 1 to 3 described above); and Fig. 1c, the execution context graph (step 4).

It is easy to see how the construction can be accomplished in polynomial time. Supposing that m represents the number of clauses of C, the time complexity of the procedure is $O(m^2)$ due to the fact that it only needs to consider the different pairs of clauses of C. All that remains to be shown is that C is satisfiable if and only if there exists a 4-implementation I of S with $\Delta(I) \leq 2$.

Firstly, we will prove that there exists a 4-implementation I of S with $\Delta(I) \leq 2$ if C is satisfiable. Given any satisfying truth assignment $t : U \rightarrow \{T, F\}$, let us create the collection of instances I from R as follows. The instances denoted by I_j^V, $I_j^{C_1}$, $I_j^{C_2}$, $I_j^{C_3}$, I_j^U, $I_{j,k}^U$, and $I_{j,k}^C$ are created from the execution contexts R_j^V, $R_j^{C_1}$, $R_j^{C_1}$, $R_j^{C_2}$, $R_j^{C_3}$, R_j^U, $R_{j,k}^U$, and $R_{j,k}^C$, respectively. Initially, these instances contain the same functionalities as the related execution contexts. Let us define $b : U \rightarrow \mathcal{F}$ as follows:

$$b(u_i) = \begin{cases} T_i & \text{if} \quad t(u_i) = T \\ F_i & otherwise \end{cases} \tag{2}$$

Let us add $\{b(u_p), b(u_q), b(u_s)\} \subset \mathcal{F}$ to each $I_j^U \in I$ where u_p, u_q, and u_s are the variables of c_j. Let us add $\{b(u_p), b(u_q)\} \subset \mathcal{F}$ to each $I_{j,k}^U \in I$ where u_p and u_q are the variables shared between c_j and c_k. For all $R_i \in R$, the related instance $I_i \in I$ verifies that $R_i \subseteq I_i$ and $|I_i| = 4$. Thus, I is a 4-implementation.

We must prove that $\delta(I_i, I_j) \leq 2$ for each pair of instances $I_i, I_j \in I$ such that I_i and I_j are related to adjacent execution contexts in G. Each edge of G belongs to one of the following categories:

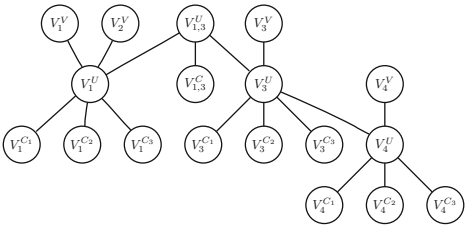
$$C = \{c_1, c_2, c_3, c_4\};\ U = \{u_1, u_2, u_3, u_4, u_5, u_6\}$$

$$c_1 = \{u_1, u_2, \bar{u}_3\};\ c_2 = \{\bar{u}_1, u_2, u_3\};\ c_3 = \{\bar{u}_1, \bar{u}_2, \bar{u}_4\};\ c_4 = \{u_4, u_5, u_6\}$$

(a)

$$\begin{aligned}
&Step\ 1 && : H(C) = \{c_1, c_3, c_4\}\\
&Step\ 2 && : \mathcal{F} = \{X, Y, Z, T_1, F_1, T_2, F_2, T_3, F_3, T_4, F_4, T_5, F_5, T_6, F_6\}\\
&Step\ 3 && : R = \left\{R_1^V, R_2^V, R_3^V, R_4^V, R_1^{C_1}, R_1^{C_2}, R_1^{C_3}, R_3^{C_1}, R_3^{C_2}, R_3^{C_3}, R_4^{C_1}, R_4^{C_2}, R_4^{C_3}, R_1^U, R_3^U, R_4^U, R_{1,3}^U, R_{1,3}^U\right\}\\
&Step\ 3a && : R_1^V = \{X, T_1, T_2, F_3\};\ R_2^V = \{X, F_1, T_2, T_3\};\ R_3^V = \{X, F_1, F_2, F_4\};\ R_4^V = \{X, T_4, T_5, T_6\}\\
&Step\ 3b && : R_1^{C_1} = \{T_2, F_2, T_3, F_3\};\ R_1^{C_2} = \{T_1, F_1, T_3, F_3\};\ R_1^{C_3} = \{T_1, F_1, T_2, F_2\}\\
&\quad && R_3^{C_1} = \{T_2, F_2, T_4, F_4\};\ R_3^{C_2} = \{T_1, F_1, T_4, F_4\};\ R_3^{C_3} = \{T_1, F_1, T_2, F_2\}\\
&\quad && R_4^{C_1} = \{T_5, F_5, T_6, F_6\};\ R_4^{C_2} = \{T_4, F_4, T_6, F_6\};\ R_4^{C_3} = \{T_5, F_5, T_6, F_6\}\\
&Step\ 3c && : R_1^U = \{X\};\ R_3^U = \{X\};\ R_4^U = \{X\}\\
&Step\ 3d && : R_{1,3}^U = \{Y, Z\};\ R_{1,3}^C = \{T_1, F_1, T_2, F_2\}
\end{aligned}$$

(b)



(c)

$$t(u_1) = t(u_2) = t(u_5) = t(u_6) = T \qquad I_1^U = \{X, T_1, T_2, F_3\};\quad I_3^U = \{X, T_1, T_2, F_4\}$$

$$t(u_3) = t(u_4) = F \qquad I_4^U = \{X, F_4, T_5, T_6\};\quad I_{1,3}^U = \{Y, Z, T_1, T_2\}$$

(d) **(e)**

Fig. 1 Example of transformation from 3SAT to MMRCDP: **a** clauses, **b** execution contexts, **c** execution context graph, **d** a satisfying truth assignment for C, and **e** the equivalent 4-implementation (only the instances that differ to the related execution contexts are shown)

- $\{V_j^U, V_j^V\} \in E$. By construction, I_j^V contains X and three truth functionalities each one representing a truth-setting that satisfies a different literal of c_j. The instance I_j^U contains X and at least one truth functionality that represent a truth-setting that satisfies c_j. As C is satisfiable, at least one of the literal of c_j is satisfiable; so, $|I_j^U \cap I_j^V| \geq 2$ and thus $\delta(I_j^U, I_j^V) \leq 2$.
- $\{V_j^U, V_j^{C_i}\} \in E$ for $i = 1, 2, 3$. By construction, each $I_j^{C_i}$ contains all possible truth functionalities of two variables of c_j. The instance I_j^U contains X and the truth functionalities given by the function b for the three variables of c_j. So, $|I_j^U \cap I_j^{C_i}| = 2$ and thus $\delta(I_j^U, I_j^{C_i}) = 2$ for $i = 1, 2, 3$.
- $\{V_j^U, V_k^U\} \in E$. This edge is created when c_j and c_k share a unique variable. The instances I_j^U and I_k^U only share X and the truth functionality given by the function b for the shared variable. So, $|I_j^U \cap I_k^U| = 2$, and thus, $\delta(I_j^U, I_k^U) = 2$.

– $\{V_j^U, V_{j,k}^U\} \in E$. This edge is created when c_j and c_k share exactly two variables. The instances I_j^U and $I_{j,k}^U$ only share the truth functionalities given by the function b for the shared variables. So, $|I_j^U \cap I_{j,k}^U| = 2$ and thus, $\delta(I_j^U, I_{j,k}^U) = 2$.
– $\{V_{j,k}^C, V_{j,k}^U\} \in E$. By construction, $I_{j,k}^C$ contains all possible truth functionalities of the two shared variables. The instance $I_{j,k}^U$ contains the truth functionalities given by the function b for the two shared variables. So, $|I_{j,k}^C \cap I_{j,k}^U| = 2$ and thus $\delta(I_{j,k}^C, I_{j,k}^U) = 2$.

We conclude that $\Delta(I) = 2$. Conversely, we will prove that C is satisfiable if there exists a 4-implementation I of S with $\Delta(I) \leq 2$. Previously, we prove some lemmas.

Lemma 1 *Let $S = (\mathcal{F}, G, R)$ be a HRS obtained from a 3SAT instance and let I be a 4-implementation of S with $\Delta(I) \leq 2$. Then, for all $I_j^U \in I$, there do not exist $T_r, F_r \in \mathcal{F}$ such that $\{T_r, F_r\} \subset I_j^U$.*

Proof By contradiction, let us assume that there exit $T_r, F_r \in \mathcal{F}$ such that $\{T_r, F_r\} \subseteq I_j^U$; so, as $X \in R_j^U$ and $R_j^U \subseteq I_j^U$, $\{X, T_r, F_r\} \subseteq I_j^U$. By construction, there exist $R_j^{C_i} \in R$ such that $\{X, T_r, F_r\} \cap R_j^{C_i} = \emptyset$ and $|R_j^{C_i}| = 4$. Therefore, $\{X, T_r, F_r\} \cap I_j^{C_r} = \emptyset$ and so $I_j^U \cap I_j^{C_r} \subseteq I_j^U \backslash \{X, F_r, T_r\}$. Then $|I_j^U \cap I_j^{C_i}| \leq |I_j^U \backslash \{X, F_r, T_r\}| = 1$ and therefore $\delta(I_j^U, I_j^{C_i}) > 2$ which implies a contradiction. □

Lemma 2 *Let $S = (\mathcal{F}, G, R)$ be a HRS obtained from a 3SAT instance with a collection C of clauses on a finite set U of boolean variables and let I be a 4-implementation of S with $\Delta(I) \leq 2$. If $c_j \in H(C)$ is defined over $u_p, u_q, u_s \in U$ then $I_j^U = \{X, f_p, f_q, f_s\}$ where $f_i \in \{T_i, F_i\}$.*

Proof Since $\Delta(I) \leq 2$, $\delta(I_j^U, I_j^{C_i}) \leq 2$ for $i = 1, 2, 3$; therefore, $|I_j^U \cap I_j^{C_i}| \geq 2$ for $i = 1, 2, 3$. By construction, there exists a $R_j^{C_i} \in R$ such that $R_j^{C_i} = \{T_p, F_p, T_r, F_r\}$ where $r = q$ or $r = s$ and so, $I_j^{C_i} = \{T_p, F_p, T_r, F_r\}$. By contradiction, let us assume that $\{T_p, F_p\} \cap I_j^U = \emptyset$; so, $I_j^U \cap I_j^{C_i} \subseteq I_j^{C_i} \backslash \{T_p, F_p\} = \{T_r, F_r\}$. As $|I_j^U \cap I_j^{C_i}| \geq 2$, $\{T_r, F_r\} \subseteq I_j^U$ which implies a contradiction by Lemma 1. So, $f_r \in I_j^U$ for all variables $u_r \in c_j$ where $f_r = \{T_r, F_r\}$. Since $R_j^U = \{X\}$ by construction, $I_j^U = \{X, f_p, f_q, f_s\}$. □

Lemma 3 *Let $S = (\mathcal{F}, G, R)$ be a HRS obtained from a 3SAT instance and let I be a 4-implementation of S with $\Delta(I) \leq 2$. For each pair $I_j^U, I_k^U \in I$ with $j \neq k$, there do not exist $T_r, F_r \in \mathcal{F}$ such that $\{T_r, F_r\} \subset I_j^U \cup I_k^U$.*

Proof Let c_j and c_k be the clauses of a 3SAT instance related to I_j^U and I_k^U, respectively. The proof is divided into the following cases:

– c_j and c_k do not share variables. Let u_p be a variable of c_j. By Lemma 1, $\{T_p, F_p\} \not\subset I_j^U$. As c_j and c_k do not share variables, c_k is not defined over u_p. So, it is follows from Lemma 2 that $\{T_p, F_p\} \cap I_k^U = \emptyset$. So, $\{T_p, F_p\} \not\subset I_j^U \cup I_k^U$ for any non-shared variable u_p.

– c_j and c_k share a unique variable u_p. Let u_p, u_q, and u_r be the variables of c_j and let u_p, u_s, and u_t be the variables of c_k where $s \neq q, r$ and $t \neq q, r$. By contradiction, let us assume that $T_p \in I_j^U$ and $F_p \in I_k^U$. By Lemma 2, $I_j^U = \{X, T_p, f_q, f_s\}$ and $I_k^U = \{X, F_p, f_r, f_t\}$ where $f_i \in \{T_i, F_i\}$, then $I_j^U \cap I_k^U = \{X\}$. Since $|I_j^U \cap I_k^U| < 2$, we have $\delta(I_j^U, I_k^U) > 2$ which implies a contradiction because $\Delta(I) \leq 2$. So, $\{T_p, F_p\} \not\subset I_j^U \cup I_k^U$.
– c_j and c_k share two variables u_p and u_q. Firstly, we prove that $X \notin I_{j,k}^U$. By construction, $R_{j,k}^U = \{Y, Z\}$, $\{X, Y, Z\} \cap R_{j,k}^C = \emptyset$, and $|R_{j,k}^C| = 4$. Therefore, $\{Y, Z\} \subset I_{j,k}^U$ and $\{X, Y, Z\} \cap I_{j,k}^C = \emptyset$; so, $I_{j,k}^C \cap I_{j,k}^U \subseteq I_{j,k}^U \backslash \{Y, Z\}$. Since $\Delta(I) \leq 2$, we have $2 = |I_{j,k}^U \backslash \{Y, Z\}| \geq |I_{j,k}^C \cap I_{j,k}^U| \geq 2$ which implies $I_{j,k}^C \cap I_{j,k}^U = I_{j,k}^U \backslash \{Y, Z\}$. Then $X \notin I_{j,k}^U$ because $X \notin I_{j,k}^C$.
By Lemma 2, $I_j^U = \{X, f_p, f_q, f_r\}$ and $I_k^U = \{X, f_p', f_q', f_s\}$ where $f_i, f_i' \in \{T_i, F_i\}$ and $r \neq s$. As $\{Y, Z\} \cap I_j^U = \emptyset$, $I_j^U \cap I_{j,k}^U \subseteq I_{j,k}^U \backslash \{Y, Z\}$. Similarly, as $\{Y, Z\} \cap I_k^U = \emptyset$, $I_k^U \cap I_{j,k}^U \subseteq I_{j,k}^U \backslash \{Y, Z\}$. Since $\Delta(I) \leq 2$, $|I_j^U \cap I_{j,k}^U| \geq 2$ and $|I_{j,k}^U \cap I_k^U| \geq 2$ which implies that $I_j^U \cap I_{j,k}^U = I_{j,k}^U \cap I_k^U = I_{j,k}^U \backslash \{Y, Z\}$. Therefore, if $X \notin I_{j,k}^U$ then $I_{j,k}^U \backslash \{Y, Z\} \subset I_j^U \cap I_k^U$; so, $|I_j^U \cap I_k^U| > |I_{j,k}^U \backslash \{Y, Z\}| = 2$ which implies that $\{f_p, f_q\} = \{f_p', f_q'\}$. So, $\{T_p, F_p\} \not\subset I_j^U \cup I_k^U$ and $\{T_q, F_q\} \not\subset I_j^U \cup I_k^U$.

Note that the case of three shared variables is not considered because, by construction, R_j^U (and so I_j^U) is created only if $c_j \in H$. □

The above lemmas allow us to define a truth assignment function $t : U \to \{T, F\}$ as follows:

$$t(u_j) = \begin{cases} T & \text{if} \quad T_j \in I_k^U \quad \text{for all } I_k^U \in I \text{ such that } u_j \in c_k \\ F & \text{if} \quad F_j \in I_k^U \quad \text{for all } I_k^U \in I \text{ such that } u_j \in c_k \end{cases} \tag{3}$$

All that remains to be shown is that t satisfies C. By construction, each truth functionality of R_j^V represents a truth-setting that satisfies c_j and, since $|R_j^V| = 4$, $I_j^V = R_j^V$. For all c_j, we have $\delta(I_{h(c_j)}^U, I_j^V) \leq 2$, then we have $|I_{h(c_j)}^U \cap I_j^V| \geq 2$, hence there exists at least a $f_p \in \{T_p, F_p\} \subset \mathcal{F}$ such that $f_p \in I_{h(c_j)}^U \cap I_j^V$. Then $f_p \in I_j^V$ which implies that the truth value $t(u_p)$ satisfies c_j. So, t satisfies C.

It is proved that C is satisfiable if and only if there exists a 4-implementation I of S with $\Delta(I) \leq 2$. In the example of transformation, Fig. 1d, and e show a satisfying truth assignment for C and the equivalent 4-implementation I, respectively. We conclude that MMRC problem is NP-complete.

4 Integer Linear Programming formulation

Let $S = (\mathcal{F}, G, R)$ be a HRS where $\mathcal{F} = \{f_1, f_2, \ldots, f_p\}$ is a set of functionalities; $G = (V, E)$, an execution context graph; and $R = \{R_1, R_2, \ldots, R_{|V|}\}$, a collection of execution contexts. Let $I = \{I_1, I_2, \ldots, I_{|V|}\}$ be a n-implementation of S. We define the sets of binary variables $x_{i,j} \in \{0, 1\}$ and $y_{i,j,k} \in \{0, 1\}$ as follows:

$$x_{i,j} = \begin{cases} 1 & \text{if} \quad f_i \in I_j, \\ 0 & \text{otherwise.} \end{cases} \qquad i = 1, \ldots, p; \quad j = 1, \ldots, |V| \tag{4}$$

$$y_{i,j,k} = \begin{cases} 1 & \text{if} \quad f_i \in I_j \cap I_k, \\ 0 & \text{otherwise.} \end{cases} \qquad i = 1, \ldots, p; \quad j,k = 1, \ldots, |V| \tag{5}$$

Then, the MMRC problem can be formulated in the following way:

$$\text{minimize } m \tag{6}$$

$$\text{s.t. } x_{i,j} = 1 \quad \forall i, j | f_i \in R_j \tag{7}$$

$$\sum_{i=1}^{p} x_{i,j} = n \quad j = 1, \ldots, |V| \tag{8}$$

$$2y_{i,j,k} \leq x_{i,j} + x_{i,k} \leq 1 + y_{i,j,k} \quad i = 1, \ldots, p; \quad j,k = 1, \ldots, |V| \tag{9}$$

$$m \geq n - \sum_{i=1}^{p} y_{i,j,k} \quad \forall j,k | \{v_j, v_k\} \in E \tag{10}$$

The constraint (7) and (8) ensure that the I is a n-implementation of S. The constraint (9) ensures that the variable y is consistent with the definition (5). The reconfiguration cost $\delta(I_j, I_k)$ can be calculated as $n - \sum_{i=1}^{p} y_{i,j,k}$. So, the constraint (10) ensures that m is an upper bound for the reconfiguration cost between any pair of instances corresponding to adjacent execution contexts. So, the minimum maximum reconfiguration cost of I can be obtained by minimizing m, as shown in (6).

5 Experimental results

The proposed ILP formulation has been solved using the solver Gurobi 4.6 [6]. The experiments have been executed in an Intel Xeon X5660 (4 cores) at 2.80 GHz with 16 GB of RAM running Linux 64 bits (Red Hat Enterprise Linux 6). The test data set consists on 50 randomly generated MMRC instances. The experiments have been executed with a time limit of 7200 s. Table 1 summarizes the obtained results. We refer to the problem instances solved to optimality as "solved instances"; on the other hand, we refer to the instances that could not be solved within the imposed time limit as "non-solved instance". For each problem instance, the table shows the size of the n-implementation (n), the number of functionalities ($|F|$), the number of edges of the execution context graph ($|E|$), the number of vertices of the execution context graph ($|V|$), the mean of the cardinality of the execution contexts ("C-mean"), the standard deviation of the cardinality of the execution contexts ("C-std"), the number of the average final percentage optimality gap ("Gap"), and the amount of time in seconds spent by solved instances ("Time"). The rows of the table are sorted in increasing order of $|E|$. As can be observed, there is a trend of increasing the time with the increasing of the number of edges. The ILP formulation found an optimal solution in the 76 % of the cases (38 instances). The average time for these cases was 605 s.

Table 1 Experimental results

n	$\|F\|$	$\|E\|$	$\|V\|$	C-mean	C-std	Gap	Time
54	134	55	22	24.77	16.35	–	0.60
33	123	57	21	13.81	8.26	–	3.88
48	100	59	25	24.64	14.27	–	0.48
21	90	93	26	12.77	5.85	–	0.35
48	160	102	30	21.77	14.66	–	2.57
47	148	105	33	14.79	9.63	–	49.36
38	79	107	35	19.80	10.54	–	1.70
46	94	148	40	17.07	10.33	6.67	–
59	187	158	35	24.94	15.51	–	49.72
6	22	194	43	3.30	1.57	–	0.09
53	134	203	49	20.04	14.74	–	31.06
31	143	221	47	9.96	6.60	6.25	–
34	92	261	49	13.90	7.23	–	153.77
30	78	264	52	13.69	8.04	–	7.17
34	114	270	58	13.59	8.69	–	157.73
8	31	326	53	4.32	2.05	–	0.53
45	131	347	74	14.88	11.94	–	72.31
58	237	394	57	30.60	15.90	–	98.46
34	142	433	60	16.82	8.87	–	22.51
19	67	458	59	6.34	4.04	–	776.90
22	68	503	80	10.94	5.71	–	155.44
57	221	550	69	23.70	15.54	–	1238.63
18	66	562	84	9.52	5.31	–	9.33
6	10	601	77	3.06	1.57	–	2.53
8	14	633	73	4.63	2.31	–	0.16
52	169	638	70	27.70	13.38	–	551.91
50	204	645	75	21.59	12.12	3.12	–
44	188	687	92	19.96	12.17	–	174.31
57	145	870	87	24.00	14.54	18.75	–
36	86	891	115	14.88	8.97	5.00	–
48	96	926	87	19.63	11.26	17.39	–
49	186	980	91	19.36	11.90	9.38	–
15	54	1038	130	6.92	3.86	–	102.23
21	96	1092	132	9.76	5.69	–	247.30
40	183	1125	122	16.16	10.62	3.45	–
54	221	1131	96	26.83	16.10	–	2046.31
26	110	1146	104	11.95	7.52	–	379.87
33	147	1333	105	15.13	8.49	–	2171.91
37	153	1362	133	16.79	10.06	–	2595.82
31	83	1440	143	14.99	8.59	5.00	–
6	22	1478	120	3.50	1.56	–	0.68
38	68	1657	115	20.44	10.52	–	4626.96

Table 1 continued

25	105	1679	132	13.29	6.54	–	2090.29
35	53	1695	149	16.51	10.24	–	4626.65
18	77	1697	121	6.74	4.27	6.67	–
11	46	1827	129	5.49	3.13	–	70.51
58	171	2123	134	20.67	13.40	–	191.80
23	72	2123	136	10.40	6.17	5.56	–
35	106	2662	146	13.80	8.78	–	–
43	197	2667	145	16.84	11.49	–	280.11

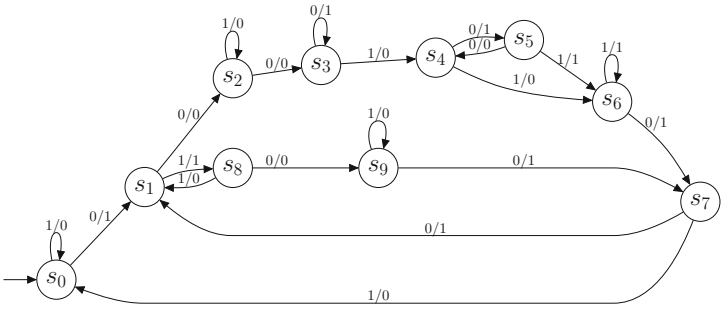
The average "Gap" for the non-solved instances was 7.93 % (one non-solved instance has been excluded from the average because the solver could not calculate the root relaxation solution within the imposed time limit; see the next-to-last row of Table 1).

6 Practical application

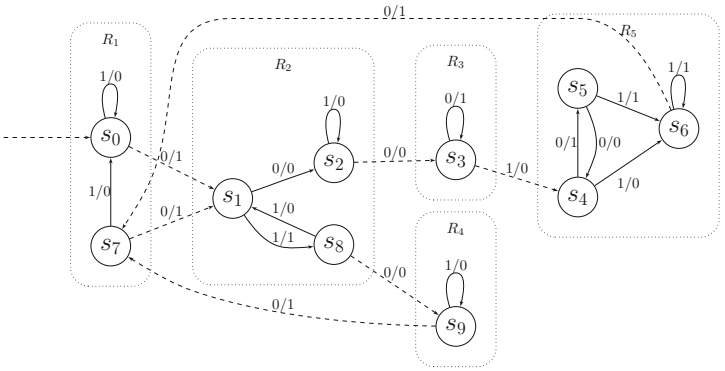
Finite State Machines (FSMs) are traditionally implemented in FPGA by using flip-flops and Look-up-Tables (LUTs). However, there is an increasing interest for implementing FSMs using memory resources because the current FPGAs include a very large number of these resources. In memory-based implementations, the transition and output functions are mapped to a conventional memory whose depth depends on the number of FSM inputs and states. FSMs with a large number of states require large memories that are inefficiently implemented in FPGAs [12].

FVSM is a new model of Finite State Machine (FSM) proposed with the aim of improving the performance of memory-based implementations of FSMs with a large number of states [11]. The key idea is that only a subset of the states is needed during a period of time. Following this idea, an FVSM equivalent to a given FSM can be obtained by decomposing the FSM into different subFSMs (this process is called *virtualization*). The FVSM architecture presented in [11] is a two-level memory hierarchy that include a main memory and a secondary memory. The main memory implements a generic memory-based FSM that contains the active subFSM. When the active subFSM must be changed for another one (called update process), the required states are transferred from the secondary memory to the main memory without stopping or altering the FSM operation. The active subFSM generates the FSM output and controls the transfer between memories.

The performance of a FVSM implementation mainly depends on the maximum number of states that must be transfered between memories in an update process. So, the maximum number of states in which two adjacent subFSMs differ must be minimized. The MMRC problem have been used for this purpose. In this case, functionalities ($\mathcal{F}$) and execution contexts (R_i) represent states and subFSMs, respectively. This procedure helps to obtain FVSM implementations with higher performance. Figure 2 shows an example. The FVSM shown in Fig. 2b is equivalent to the FSM shown

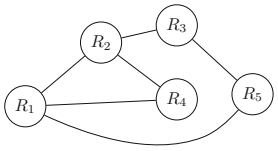


(a)



(b)

$$n = 4; \quad \mathcal{F} = \{s_0, s_1, \ldots, s_9\}$$
$$R_1 = \{s_0, s_7\}; \quad R_2 = \{s_1, s_2, s_8\}; \quad R_3 = \{s_3\}$$
$$R_4 = \{s_9\}; \quad R_5 = \{s_4, s_5, s_6\}$$



(c)

$$I_1 = \{s_0, s_7, s_1, s_6\}; \quad I_2 = \{s_1, s_2, s_8, s_0\}; \quad I_3 = \{s_3, s_0, s_6, s_8\}$$
$$I_4 = \{s_9, s_0, s_7, s_8\}; \quad I_5 = \{s_4, s_5, s_6, s_0\};$$

(d)

Fig. 2 Practical application: **a** FSM, **b** equivalent FVSM, **c** MMRCP problem, and **d** optimal solution with $\Delta(I) = 2$

in Fig. 2a. The slashed arcs denote update processes. Figure 2c shows the MMRC problem related to the FVSM. The number of states stored on main memory must be power of 2, then $n = 4$. Figure 2d shows an optimal solution with $\Delta(I) = 2$.

References

1. Cardoso, J.M.P., Diniz, P.C., Weinhardt, M.: Compiling for reconfigurable computing: a survey. ACM Comput. Surv. **42**, 13:1–13:65 (2010)
2. El-Araby, E., El-Ghazawi, T., Gaj, K.: A system-level design methodology for reconfigurable computing applications. In: IEEE International Conference on Field-Programmable Technology, 2005. Proceedings, pp. 311–312 (2005)
3. Estrin, G.: Organization of computer systems: the fixed plus variable structure computer. Papers presented at the May 3–5. 1960, western joint IRE-AIEE-ACM computer conference, IRE-AIEE-ACM '60 (Western), pp. 33–40. ACM, New York, NY, USA (1960)
4. Garey, M.R., Johnson, D.S.: Computers and Intractability: a Guide to the Theory of NP-Completeness. W. H. Freeman and Co., New York (1979)
5. Gokhale, M.B., Graham, P.S.: Reconfigurable Computing: Accelerating Computation with Field-Programmable Gate Arrays. Springer, Berlin (2005)
6. Gurobi Optimization, I.: Gurobi optimizer reference manual (2012). http://www.gurobi.com. Accessed 3 Apr 2015
7. Hauck, S., DeHon, A.: Reconfigurable Computing: The Theory and Practice of FPGA-Based Computation. Morgan Kaufmann Publishers Inc., San Francisco (2007)
8. Kao, C.: Benefits of partial reconfiguration. Xcell, Fourth Quarter 2005. http://www.xilinx.com (2005). Accessed 3 Apr 2015
9. Karp, R.: Reducibility among combinatorial problems. In: Miller, R., Thatcher, J. (eds.) Complexity of Computer Computations, pp. 85–103. Plenum Press, USA (1972)
10. Raghuraman, K., Wang, H., Tragoudas, S.: Minimizing FPGA reconfiguration data at logic level. In: 7th International Symposium on Quality Electronic Design, 2006. ISQED '06, pp. 219–224 (2006)
11. Senhadji-Navarro, R., Garcia-Vargas, I.: Finite virtual state machines. IEICE Transactions on Information and Systems E95.D, pp. 2544–2547 (2012)
12. Senhadji-Navarro, R., Garcia-Vargas, I., Guisado, J.: Performance evaluation of RAM-based implementation of finite state machines in FPGAs. In: 19th IEEE International Conference on Electronics, Circuits and Systems (ICECS), pp. 225–228 (2012). doi:10.1109/ICECS.2012.6463760