Proceedings of the IASTED International Conference
Eleventh IASTED International Conference on Parallel and Distributed Computing and Systems
November 3-6, 1999, Boston, USA

A Method and a Genetic Algorithm for Deriving Protocols for Distributed Applications with Minimum Communication Cost

Khaled El-Fakih[†] Hirozumi Yamaguchi[‡] Gregor v. Bochmann[†]

[†]School of Information Technology and Engineering
University of Ottawa
Ottawa, Ontario, K1N 6N5, CANADA
{kelfakih, bochmann}@site.uottawa.ca

[‡]Graduate School of Engineering Science
Osaka University
Toyonaka, Osaka, 560-8531, JAPAN
h-yamagu@ics.es.osaka-u.ac.jp

Abstract We consider a set of rules for deriving the specification of the protocol of a distributed system from a given specification of services, and define and formulate the message exchange optimization problem using a 0-1 integer programming model. This problem is about determining the minimum number of messages to be exchanged between the physical locations of the distributed system, in order to reduce the communication cost. We then present a genetic algorithm for solving this problem. The main advantage of this algorithm, in comparison with exact algorithms, is that its complexity remains manageable for realistic large specifications. The experimental results show that the minimum number of messages to be exchanged is found in a very reasonable time.

keywords protocol synthesis, service specification, protocol specification, communication cost, 0-1 integer linear programming, genetic algorithm

1 Introduction

Protocol synthesis methods[2, 3, 4, 5] (for survey see [1]) have been used to derive a specification of a distributed system (called a *protocol specification*) automatically from a given specification of the services to be provided by the distributed system to its users (called a *service specification*). The service specification is written like a program of a centralized system, and does not contain any specification of message exchange between different physical locations. However, the protocol specification contains the specification of the communication between the different protocol entities (PE's) at the different locations. Therefore, in protocol synthesis, we first have to decide how messages are exchanged between the different PE's. For this, we use the derivation policy presented in our previous work [4]. Then, we have to decide how to optimize this exchange of messages, since we consider communication costs as a primary cost criteria for distributed systems ([3] started this first, for an EFSM based synthesis method).

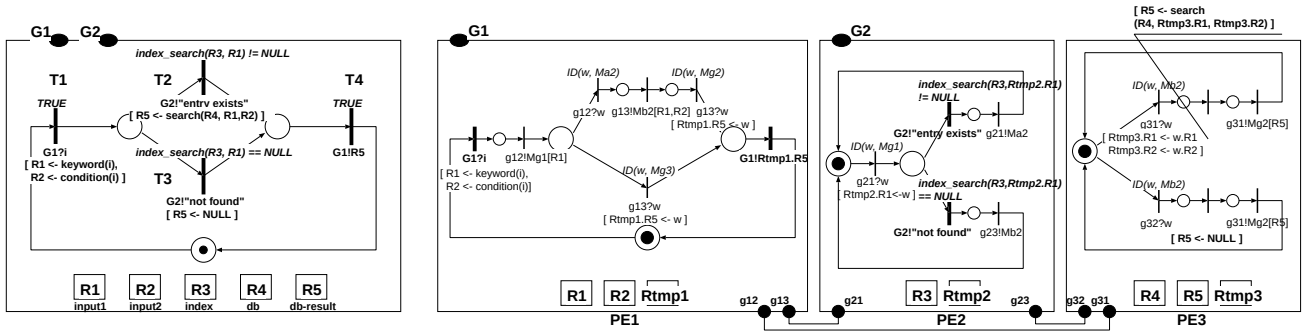
In the context of distributed applications, one also has to deal with the allocation of resources to the different physical locations. As an example, we consider a distributed database application which uses several databases such as a customer database and an account database located on different computers. To complete a transaction in a speedy and efficiently manner, the number of messages exchanged between the databases and the client should be minimized.

In this paper, we consider the automatic derivation of a protocol specification from a given service specification when the allocation of the resources to the different protocol entities (PE) is given [4]. More specifically, we deal with the problem of optimizing the number of messages to be exchanged between the PE's during the execution of the protocol, by formulating it as a 0-1 integer linear programming (ILP) problem. Moreover, we present a new hybrid genetic algorithm for solving this optimization problem. The main advantage of this algorithm, compared with other exact algorithms for solving 0-1 ILP problems, is the fact that its complexity remains manageable for realistic large specifications. For example, existing branch-and-bound algorithms have exponential complexity, and spend too much time for solving large problems. Moreover, cutting plane methods sometimes work well, but sometimes not at all, especially for large problem sizes.

2 Service Specification and Protocol Specification

In this paper, we describe service and protocol specifications using a Petri net model extended with registers (representing computational data) and gates (representing service access points), in the following simply called Petri Net with Registers (PNR in short). Each transition t in PNR has a label $\langle \mathcal{C}(t), \mathcal{E}(t), \mathcal{S}(t) \rangle$, where $\mathcal{C}(t)$ is a pre-condition statement (one of the firing conditions of t), $\mathcal{E}(t)$ is an event expression and $\mathcal{S}(t)$ is a set of substitution statements. For example, in Fig. 1(a), $\mathcal{C}(t_1) =$"$TRUE$", $\mathcal{E}(t_1) =$"$G_1?i$" and

This work was partially funded by Communications and Information Technology Ontario (CITO).



(a) Service Specification **(b) Protocol Specification**

Figure 1: *Service and Protocol Specifications*

$\mathcal{S}(t_1)$ ="$R_1 \leftarrow keyword(i)$, $R_2 \leftarrow condition(i)$". Note that G_1 is a gate, R_1 and R_2 are registers and i is an input variable. t_1 can fire if the value $C(t_1)$ is true and if an input is given through gate G_1. If t_1 fires, $\mathcal{E}(t_1)$ is executed (the input value is assigned to input variable i) and then the statements in $\mathcal{S}(t_1)$ are executed simultaneously (the values of R_1 and R_2 are changed).

At a highly abstracted level, a distributed system is regarded as a centralized system which works and provides services as a single "virtual" machine. The virtual machine has all the gates and registers used by the system. The number of actual PE's and communication channels among them are hidden. The specification of the distributed system at this level is called a *service specification* and denoted by *Sspec*.

Fig. 1(a) shows *Sspec* of a simple distributed system written in the PNR model. An input which contains a pair of a keyword and a condition given through G_1 enables t_1 fire. If it fires, the values of the input parameters are stored in R_1 and R_2, respectively. Then either t_2 or t_3 may fire, depending on the value of $index\_search(R_3, R_1)$ (the result of index searching using keyword). t_2 can fire if the value of the pre-condition "$index\_search(R_3, R_1)! = NULL$" is true, otherwise t_3 can. If t_2 fires, the system outputs a message "entry exists" through G_2, and then executes "$R_5 \leftarrow search(R_4, R_1, R_2)$" (searching the database). If t_3 fires, the system outputs a message "not found" through G_2 and then the value of R_5 is set to NULL. Finally, t_4 fires and the value of R_5 is output through G_1. Then the system will have the initial marking again.

Realistically, a distributed system is a communication system which consists of p protocol entities PE_1, PE_2, ... and PE_p. Any pair of PE's, PE_i and PE_j, have a duplex reliable communication channel with buffers of infinite capacity. The PE_i's (PE_j's) side of the communication channel is represented as gate g_{ij} (g_{ji}). Gates and registers (called resources) are allocated to PE's (physical places). We must specify the behavior of each PE in order to implement the distributed system. This indicates the communication between the PE's. A specification of a protocol entity PE_k is called a *protocol entity specification* and denoted by $Pspec_k$. A set of p protocol entity specifications $\langle Pspec_1, ..., Pspec_p \rangle$ is called a *protocol specification* and denoted by $Pspec^{\langle 1,p \rangle}$.

Fig. 1(b) shows $Pspec^{\langle 1,p \rangle}$ of the distributed system written in the same model. An event expression such as "$g_{12}!M_{g1}[R_1]$" at PE_1 means that a message is sent from PE_1 to PE_2, containing an identifier M_{g1} and the value of register R_1. A single message may contain values of several registers and also values of input variables. PE_2 receives the message sent by PE_1 through an event "$g_{21}?w$" (which means that the message is assigned to the input variable w). $ID(M_{g1}, w)$ is a predicate whose value is true *iff* the identifier of the message stored in the input variable w is M_{g1}. PE_1 first checks the value of the pre-condition "$TRUE$" (which is always true) and executes an input event "$G_1?i$", corresponding to the event of t_1 in *Sspec*. The input value is assigned to i. Then the substitution statements of t_1 in *Sspec* are executed and the values of $keyword(i)$ and $condition(i)$ are stored in R_1 and R_2, respectively. Then PE_1 sends a message to PE_2 containing the value of R_1. PE_2 receives this message and checks the value of the pre-conditions "$search(R_3, R_1)! = NULL$" and "$search(R_3, R_1) == NULL$", using the values of R_1 received from PE_1 and its own R_3[1] . Now, assuming that the value of "$search(R_3, R_1)! = NULL$" is true, PE_2 executes the event "G_2!entry exists" corresponding to the event of t_2 in *Sspec*. Then PE_2 sends a message to PE_1 enabling PE_1 to send the values of both R_1 and R_2 to PE_3. After receiving this message, PE_1 sends these values to PE_3 in a single message. PE_3 receives the message, executes the substitution statement "$R_5 \leftarrow search(R_4, R_1, R_2)$" using the values of R_1 and R_2 received from PE_1, and sends the value of R_5 to PE_1. PE_1 receives the message and executes an event "$G_1!R_5$", which is the same event as t_4 in *Sspec*. At this point, all PE's have again their initial markings.

We assume that an allocation of gates and registers to PE's (called a resource allocation) is given. The protocol synthesis method consists of deriving a protocol specification $Pspec^{\langle 1,p \rangle}$ using a given fixed resource allocation, such that the distributed system of PE's provides the same services as a given service specification *Sspec*.

[1] We assume that each PE_i has a register $Rtmp_i$ which temporarily keeps several values given through gates (message contents and user inputs). The values in $Rtmp_i$ can be distinguished by adding the name of the value such as $Rtmp_i.R_3$.

3 Protocol Derivation and Message Optimization

3.1 Derivation Method

A method for deriving a protocol specification $Pspec^{\langle 1,p\rangle}$ with a given resource allocation is described here. It is based on the simulation of each transition of the service specification by corresponding transitions of the PE's in the protocol specification.

For a given transition t of $Sspec$, the PE that has gate G_s used in $\mathcal{E}(t)$ checks the value of $\mathcal{C}(t)$ (pre-condition statement) and executes $\mathcal{E}(t)$ (event expression). After that, the PE sends messages called α-messages to the PE's which have the registers used in the arguments of $\mathcal{S}(t)$ (substitution statements). In response, these PE's send the register values to the PE's which have the registers to be updated in $\mathcal{S}(t)$ as messages called β-messages. The substitution statements are executed and notification messages called γ-messages are sent to those PE's which will start the execution of the next transitions. In Fig. 2, we present our derivation method of the service specification as a set of rules which specify how PE's execute each transition of $Sspec$. The method to construct a protocol specification in PNR model based on this derivation method is described in [4].

For example, for transition t_2 of $Sspec$ in Fig. 1(a), PE_2 checks the value of the pre-condition of t_2 and executes the event of t_2. After that PE_2 sends an α-message to PE_1, to let PE_1 send the value of registers R_1 and R_2 to PE_3 as a β-message. PE_3 receives the β-message and now knows the values of the arguments in the substitution statement of t_2. Therefore PE_3 changes the value of R_5 and sends a γ-message to PE_1, to let PE_1 know that the execution of t_2 has been finished.

3.2 Message Optimization Problem

The number of messages exchanged between different PE's for the execution of a transition in $Sspec$ may not be unique because a register may be allocated to more than one PE and several input/register values may be sent in one message. For example, let us assume that there are eight PE's as in Fig. 3(a). PE_6, PE_7 and PE_8 should change the values of registers R_5, R_6 and R_7, respectively. However, they do not have the registers necessary to change these values as PE_2, PE_3, PE_4 and PE_5 do. PE_1 is $PEstart(t)$. Fig. 3(a) shows the optimal way that uses five messages to send the necessary values to change the values of R_5, R_6 and R_7. However, there are many ways to send these values, Fig. 3(b) is one of them, and it uses six messages.

We consider the number of messages as the primary cost criteria for distributed systems. Moreover, if we assume that the size of a message is small, then the overhead for sending it as a packet through high speed network is high. To reduce this cost, the minimum number of messages exchanged for simulating a transition t of $Sspec$ for a given resource allocation has to be determined.

[Action Rules]

(A1) The PE which has the gate G_s appearing in $\mathcal{E}(t_x)$ checks: (a) the value of $\mathcal{C}(t_x)$ is true, (b) the simulation of the previous transitions of t_x has been finished and (c) inputs have been given through G_s, and then executes $\mathcal{E}(t_x)$. This PE is denoted by $PEstart(t_x)$.

(A2) After (A1), PE's which have registers appearing in the right-hands of statements in $\mathcal{S}(t_x)$ execute these statements. The set of these PE's are denoted by $PEsubst(t_x)$.

[Message Exchange Rules]

(M1) $PEstart(t_x)$ may send *α-messages* to PE's after (A1).

(M2) $PEstart(t_x)$ or each PE PE_i which has received an α-message may send (a) a *β-message* to $PE_j \in PEsubst(t_x)$ and (b) a *θ-messages* to $PE_k \in PEstart(t_x \bullet\bullet)$ after (M1). Note that $PEstart(t_x \bullet\bullet)$ is a set of PE's where a transition $t_y \in t_x \bullet\bullet$ exists and $PE_k = PEstart(t_y)$.

(M3) At least one β-message must arrive at each PE $PE_j \in PEsubst(t_x)$ (except $PEstart(t_x)$) before (A2).

(M4) Each PE $PE_j \in PEsubst(t_x)$ must send a γ-messages to each PE $PE_k \in PEstart(t_x \bullet\bullet)$ after (A2).

(M5) At least one θ-message must arrive at each PE $PE_k \in PEstart(t_x \bullet\bullet)$ if $\mathcal{S}(t_x) = \emptyset$ and $PE_k \neq PEstart(t_x)$.

[Message Content Rules]

(C1) Each β-message may contain the values of registers allocated to its sender PE. If the sender PE is $PE_{start}(t_x)$, the β-message can also contain the values of inputs.

(C2) Each $PE_j \in PE_{subst}(t_x)$ may use the values contained in β-messages to execute the statements in $\mathcal{S}^*(t_x)$.

(C3) Each γ-message (θ-message) can contain the values of registers allocated to its sender PE.

(C4) Each $PE_k \in PEstart(t_x \bullet\bullet)$ can use the values contained in the γ-message (θ-message) to check the value of $\mathcal{C}(t_y)$ and execute $\mathcal{E}(t_y)$ where $t_y \in t_x \bullet\bullet$.

Figure 2: *Derivation Method*

Below we formulate this message optimization problem, based on our derivation method, in the form of an 0-1 ILP problem.

3.3 Integer Linear Programming Model for Message Optimization

We introduce the following 0-1 integer variables for each t_x of $Sspec$, which represent the fact that a message is sent from one PE to another.

- $\alpha_{u,i}$: its value is one *iff* PE_u which has gate G_s used in $\mathcal{E}(t_x)$ sends an α-message to PE_i; otherwise zero.
- $\beta_{i,j}$: its value is one *iff* PE_i sends a β-message to PE_j; otherwise zero.
- $\beta_{i,j}[R_w]$ ($\beta_{i,j}[i_z]$): its value is one *iff* the β-message sent from PE_i to PE_j contains the value of register R_w (input variable i_z); otherwise zero.

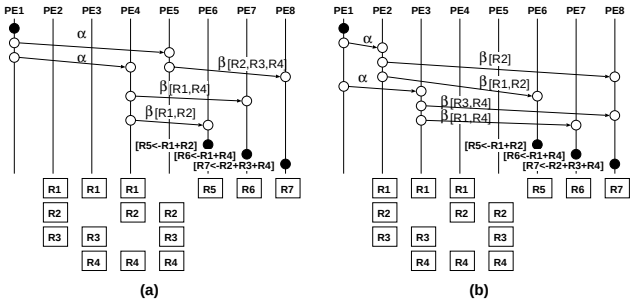


Figure 3: *(a) Optimal and (b) Non-Optimal Message Exchange*

• $\theta_{i,j}$: its value is one *iff* PE_i sends a θ-message to PE_j; otherwise zero.
• $\theta_{i,j}[R_w]$: its value is one *iff* the θ-message sent from PE_i to PE_j contains the value of register R_w; otherwise zero.
• $\gamma_{i,j}[R_w]$: its value is one *iff* the γ-message sent from PE_i to PE_j contains the value of register R_w; otherwise zero.

Using the above variables, we formulate the message optimization problem using several constraints and the following objective function Z.

$$(min)Z = \sum_i \alpha_{u,i} + \sum_i \sum_j \beta_{i,j} + \sum_i \sum_k \theta_{i,k}$$

The following three constraints are necessary according to the definition of variables. For example, $\beta_{i,j}$ must be one if the value of $\beta_{i,j}[R_w]$ is one.

$$\beta_{i,j} - \beta_{i,j}[R_w] \geq 0 \quad (1)$$

$$\beta_{i,j} - \beta_{i,j}[i_z] \geq 0 \quad (2)$$

$$\theta_{i,j} - \theta_{i,j}[R_w] \geq 0 \quad (3)$$

The following constraints represent policy (M2).

$$\alpha_{u,i} - \beta_{i,j} \geq 0 \quad (4)$$

$$\alpha_{u,i} - \theta_{i,k} \geq 0 \quad (5)$$

The following constraint represents policy (M3).

$$\sum_i \beta_{i,j} \geq 1 \quad (6)$$

The following constraint represents policy (M5).

$$\sum_i \theta_{i,k} \geq 1 \quad (7)$$

Constraints (8) and (9) are necessary to represent policy (C2), if R_w is not allocated to PE_j and needed by PE_j to execute statements in $\mathcal{S}(t_x)$, and if $PE_j \neq PEstart(t_y)$ but is required by PE_j to execute statements in $\mathcal{S}(t_x)$, respectively.

$$\sum_i \beta_{i,j}[R_w] \geq 1 \quad (8)$$

$$\beta_{u,j}[i_z] = 1 \quad (9)$$

The following constraint is necessary to represent policy (C4) if R_w is not allocated to PE_k but is required by PE_k to check the value of $\mathcal{C}(t_y)$ or to execute $\mathcal{E}(t_y)$ where $t_y \in t_x \bullet\bullet$.

$$\sum_i (\theta_{i,k}[R_w] + \gamma_{i,k}[R_w]) \geq 1 \quad (10)$$

4 A Genetic Algorithm for Solving the Message Optimization Problem

Genetic algorithms are based on the mechanics of natural evolution [8]. Throughout their artificial evolution, successive generations each consisting of a population of possible solutions, called individuals (or chromosomes, or vectors of genes), search for beneficial adaptations to solve the given problem. This search is carried out by applying the Darwinian principles of "reproduction and survival of the fittest" and the genetic operators of crossover and mutation which derive the new offspring population from the current population.

Reproduction involves selecting, in proportion to its fitness level, an individual from the current population and allowing it to survive by copying it to the new population of individuals. The individual's fitness level is usually based on the cost function given by the problem under consideration. Then, crossover and mutation are carried on two randomly chosen individuals of the current population creating two new offspring individuals. Crossover involves swapping two randomly located sub-chromosomes (within the same boundaries) of the two mating chromosomes. Mutation is applied to randomly selected genes, where the values associated with such a gene is randomly changed to another value within an allowed range. The offspring population replaces the parent population, and the process is repeated for many generations.

Typically, the best individual that appeared in any generation of the run (i.e. best-so-far individual) is designated as the result produced by the genetic algorithm.

In the following sections, we describe how genetic algorithms can be adapted for solving the message optimization problem. We present the components of a hybrid genetic algorithm (GA) for minimizing the cost function Z defined in Section 3.3. The algorithm is hybridized by procedures and design choices that account for both the likelihood of producing infeasible individuals as a result of crossover and mutation, and for the premature convergence to a local optima.

4.1 Population and Chromosomal Representation

GAs population is an array of POP individuals. An individual in the population is encoded as an $(n+m)$-element vector $[X_1, X_2, ..., X_n, X_{n+1}, ..., X_{n+m}]$ where n is the number of α-, β-, and γ-messages of the individual, and the sub-vector $X_1, X_2, ..., X_n$ corresponds to a candidate solution of the optimization problem. The other sub-vector $X_{n+1}, ..., X_{n+m}$ corresponds to those m variables of the given problem which only appear in the problem constraints, for example $\beta_{ij}[Rw]$ variables. An element (gene) $X_j = 1$ (or 0), for $j \in [1...n]$, indicates the inclusion (or exclusion) of message j from the selected subset of messages to be exchanged. The initial population of individuals is randomly generated.

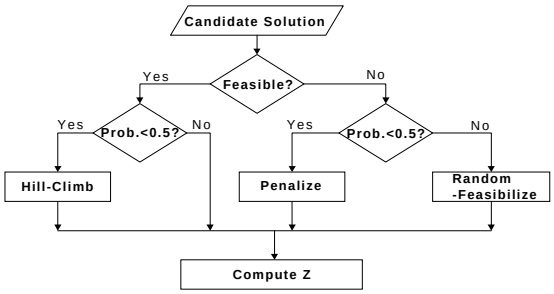


Figure 4: *Feasibilization, Penalizing and Hill-climbing*

4.2 Objective Function Evaluation

The fitness of an individual Z is evaluated by adding the genes of the first n-elements of an individual. Henceforth, the optimal number of messages to be exchanged between different PE's corresponds to the minimum value of the fitness Z of all feasible individuals.

4.3 Reproduction Scheme and Elitism

In GA, the whole population is considered a single reproduction unit within which random selection is performed. Our reproduction scheme involves elitist ranking, followed by random selection of mates from the list of reproduction trials (or copies) assigned to the ranked individuals. In the ranking scheme [6], the individuals are sorted by their fitness value. After sorting, each individual is assigned a rank based on a scale of equidistant values for the population. The ranks assigned to fittest and least-fit individuals are 1.2 and 0.8, respectively. Individuals with ranks greater than 1 are first assigned single copies. Then, the fractional part of their ranks and the ranks of the lower half of individuals are treated as probabilities for random assignment of copies.

It has been found that ranking based selection with a maximum rank of 1.2 produces individual survival percentage of 92 to 98% in different generations[6]. This helps in maintaining population diversity and controlling premature convergence. Elitism is used to exploit good building blocks and to ensure that good candidate solutions are saved if the search is to be truncated at any point. Preservation of the fittest individual is done by replacing the fittest-so-far individual in place of the least-fit individual if it is better than the current-fittest.

4.4 Genetic Operators

The genetic operators employed in GA are crossover and mutation. Pairs of individuals are randomly selected from the mating pool. Each pair of these strings undergoes crossover as follows: an integer position k along the string is selected at random between $[1..(n+m)]$, where $n+m$ is the string length. The two new strings are created by swapping all characters (genes) between $k+1$ and $n+m$ inclusively. The standard mutation operator is employed. Strings and gene positions where the alteration of the value is going to occur are selected randomly. A mutation rate of 0.02 and crossover rate of 0.7 [9] are used in our implementation.

4.4.1 Feasibilization, Penalizing, and Hill Climbing

Standard genetic operators like crossover and mutation frequently produce infeasible solutions for constrained optimization. Our GA uses the structure of the technique described in [7] which is depicted in Figure 4 and presents new penalization, random feasibilization, and hill-climbing procedures as described in detail in this subsection.

At the detection of an infeasible string i, our GA uses the suggestion of [10] and computes a penalty function, p, that starts with relaxed penalties and tightens them as the run progresses. We used:

$$p = k^q \left(\frac{t}{T}\right) f \sum_{i=1}^{n} d_i$$

where k is the number of objective function variables, q is a parameter, f is the average fitness of the population at the current generation t, T is the maximum number of generations, n is the number of problem constraints, and d_i returns the degree of violation of constraint i (absolute value of difference between the left and right hand sides of constraint i) . We experimented with $q = 0.3$, $T = 3000$.

The value of this function is added to Z. The infeasible string is then allowed to be part of population intact. In this way the penalized infeasible string will have a lower probability of survival. However, 50% of the infeasible offspring are randomly selected for heuristic completion.

A random feasibilization heuristic is applied by randomly selecting an infeasible individual, then genes which are capable of reducing this violation are selected. This selection is done by navigating throughout all the problem constraints and allocating those genes which make them satisfiable. In our case, this means that the algorithm increases, by the number of selected genes the number of messages to be exchanged between the different PE's. This helps to ensure that the population includes some feasible candidates that could be exploited in subsequent generations, and helps maintain the locus of the search near the feasible region.

To refine the solution quality, a simple problem-specific hill-climbing procedure which may decrease the individual's fitness values is incorporated. Our GA randomly selects one-half of all feasible solutions formed in each generation and applies the following fast hill-climbing procedure. The procedure search the space nearby an individual solution by selecting all those genes which are capable of reducing the fitness value while preserving the feasibility status. This selection is done by navigating throughout the problem constraints in an ordered way respecting the relationship between the different problem variables. Then, the algorithm decrements the values of the selected genes, that is the number of messages exchanged is decremented by the number of selected genes. This hill-climbing procedure enables individuals to rapidly climb the peaks which speeds up the evolution process.

Table 1: *The Execution Times of GA for T_1, ..., T_{20}*

Tr.	Service Spec.			Alloc.		ILP		Result	
	#R	#SB	#US	#PE	#AP	#VR	#CT	#MS	TM(sec)
T_1	10	4	4.3	10	2.0	229	245	28	0.52
T_2	10	4	4.0	15	1.5	205	204	21	0.38
T_3	10	4	4.0	10	2.7	233	243	24	0.47
T_4	15	5	4.4	15	3.2	836	877	46	0.93
T_5	20	5	4.4	20	3.0	825	863	51	0.75
T_6	15	5	4.4	30	2.2	840	852	52	0.81
T_7	15	5	4.6	20	4.3	1480	1514	68	2.27
T_8	20	6	4,3	20	4.4	1566	1617	70	2.70
T_9	15	6	4.3	18	5.1	1930	1980	62	3.69
T_{10}	15	6	4.3	20	5.9	1970	2041	77	3.80
T_{11}	15	6	4.3	25	5.1	2012	2076	80	5.11
T_{12}	20	7	4.0	30	4.4	2530	2609	94	7.06
T_{13}	20	7	4.0	30	4.6	2324	2365	70	4.29
T_{14}	20	7	4.7	30	4.4	2524	2637	117	8.30
T_{15}	20	10	4.0	30	4.7	3301	3410	106	9.78
T_{16}	20	9	4.1	30	5.0	3264	3410	145	13.90
T_{17}	20	10	4.2	40	4.2	3333	3452	134	15.59
T_{18}	20	9	4.0	30	5.7	3985	4133	151	21.82
T_{19}	20	9	4.5	40	4.5	3768	3901	140	18.30
T_{20}	20	9	4.5	45	4.0	3614	3717	159	16.46

4.5 Termination Criterion and POP Size

The termination criterion is satisfied when we converge to a solution. In this work, convergence is indicated when the best-so-far string does not change its Z value for 20 consecutive generations. We experimented with different values of POP size , and it was found that POP of value 20 yields the desired best solution quality.

5 Experimental Results

We have applied our derivation method to distributed database transactions that may use and simultaneously update several databases over a network. For example, let us suppose that a bank has x accounts databases $Racc_1$, ... and $Racc_x$, located at some of its branches. Also, let us suppose that a given transaction should find the new entries in each accounts database and update y relational customer databases $Rcus_1$, ... and $Rcus_y$ that are also located at some of the bank's branches. We have modeled 20 transactions as PNR transitions (T_1 to T_{20}), with various number of registers on various allocations. Table 1 shows the data of these transactions. In the table, #R is the number of registers, #SB is the number of registers whose values are changed, #US is the average number of registers used in a statement, #PE is the number of PE's, and #AP is the average number of PE's which have the same register.

For each transaction of Table 1, we have generated a corresponding ILP problem and measured the execution time taken by the GA for solving it. The experiments have been done using a PC (with Pentium-II 266 MHz). #VR is the number of problem variables, #CT is the number of constraints, #MS is the minimum number of messages exchanged, and TM is the execution time of the GA (in seconds).

For all transactions of Table 1, even for those that involved large numbers of variables #VR and constraints #CT, the GA was able to converge to a solution in a fairly reasonable time.

6 Conclusion and Future Work

Based on our previous work on protocol synthesis[4], we have defined and formalized the message exchange optimization problem, in order to reduce the cost of communications between different derived distributed specifications (or PE's). The optimization problem is formulated using the 0-1 integer linear programming model, and solved using an adapted genetic algorithm. This algorithm is shown to find a solution for the optimization problem in a very reasonable time.

Our future work is to develop a distributed environment including our methods for protocol derivation, and formulation plus optimization of the message exchange problem. WWW servers with Java servlets would be a good implementation environment for such a purpose.

References

[1] K. Saleh, Synthesis of Communication Protocols: an Annotated Bibliography, *ACM SIGCOMM Comp. Comm. Review*, 26(5), 1996, pp. 40–59.

[2] C. Kant, T. Higashino and G.v. Bochmann, Deriving protocol specifications from service specifications written in LOTOS, *Distributed Computing*, 10(1), 1996, pp. 29–47.

[3] T. Higashino, K. Okano, H. Imajo and K. Taniguchi, Deriving Protocol Specifications from Service Specifications in Extended FSM Models, *Proc. ICDCS-13*, 1993, pp. 141–148.

[4] H. Yamaguchi, K. Okano, T. Higashino and K. Taniguchi, Synthesis of Protocol Entities' Specifications from Service Specifications in a Petri Net Model with Registers, *Proc. ICDCS-15*, 1995, pp. 510–517.

[5] A. Al-Dallal and K. Saleh, "Protocol Synthesis Using the Petri Net Model," *Proc. PDCS'97*, 1997.

[6] J.E. Baker, Adaptive Selective Methods for Genetic Algorithms, *Proc. Int. Conf. on Genetic Algorithms*, 1985, pp. 101–111.

[7] F. Easton and N. Mansour, A Distributed Genetic Algorithm for Employee Staffing and Scheduling Problems, *Proc. Int. Conf. on Genetic Algorithms*, 1993, pp. 360–367.

[8] D.E. Goldberg, *Genetic Algorithms in Search, Optimization and Machine Learning* (Addison-Wesley, 1989).

[9] J.J. Grefenstette, Optimization of Control Parameters for Genetic Algorithms, *IEEE Trans. Systems, Man, and Cybernetics*, 16(1), 1986, pp. 122–128.

[10] J.T. Richardson, M.R. Palmer and G. Liepins and M. Hilliard, Some Guidelines for Genetic Algorithms with Penalty Functions, *Proc. Int. Conf. on Genetic Algorithms*, 1989, pp. 191–197.