

Evolutionary distributed test generation methods for digital circuits



Yu.A.Skobtsov V.Yu.Skobtsov
D.E.Ivanov

Institute of Applied Mathematics & Mechanics
of National Academy of Sciences
of Ukraine

Donetsk National Technical University

Donetsk, Ukraine



Contents

- ❑ Evolutionary computation
- ❑ Classical GA
- ❑ Different realization of parallel GA in test generation
- ❑ «Worker-farmer» model for parallel GA of test generation
- ❑ Distributed fault simulation of digital circuits
- ❑ Data-flow diagram for distributed simulation & test generation
- ❑ «Migration (island)» model for parallel GA of test generation
- ❑ Experimental results
- ❑ ASMID-E – the simulation & test generation system
- ❑ Conclusion



Evolutionary computation

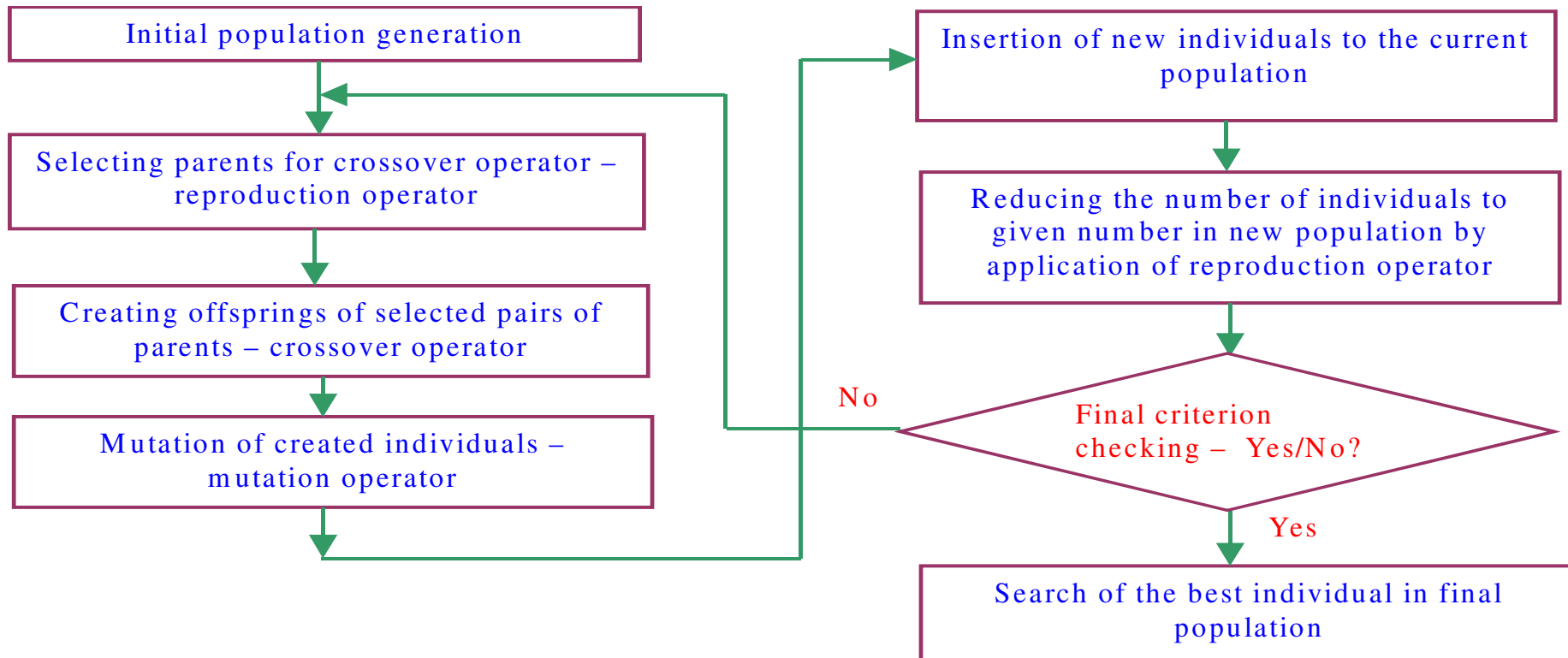
Basic paradigms:

- ❖ Genetic algorithms
- ❖ Genetic programming
- ❖ Evolutionary strategies

All basic paradigms can be applied in ATPG problem solving



A flowchart of classical GA

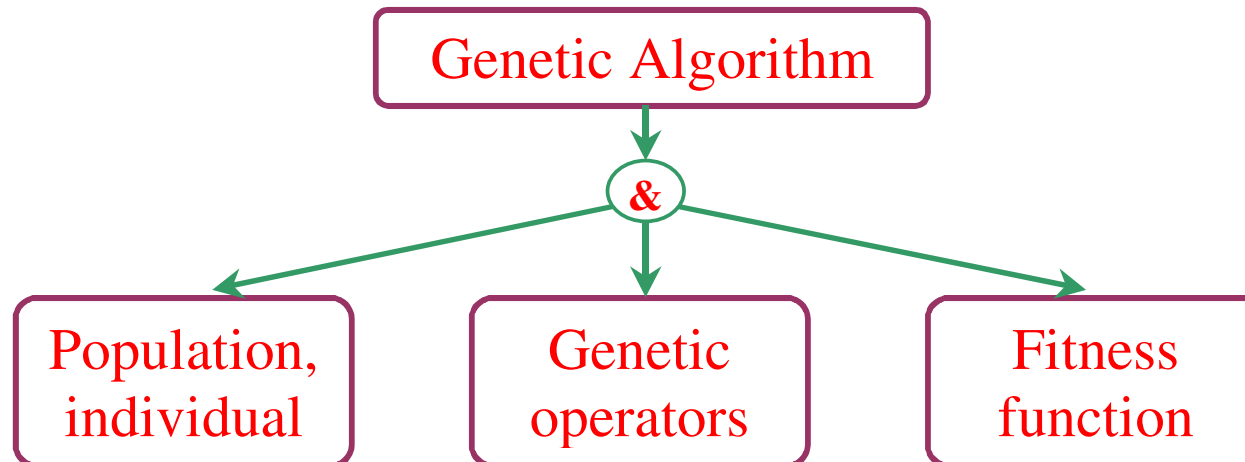




Evolutionary test generation

For solving any problem with genetic algorithm we must define:

- 1) individual and population;
- 2) genetic operators;
- 3) fitness function.





Parent selection for offspring reproduction $P^t \xrightarrow{P_s(a_i^t)} \tilde{P}$

1) roulette wheel selection
$$P_s(a_i^t) = \frac{f(a_i^t)}{\sum_{j=1}^N f(a_j^t)}$$

2) linear rank-based selection
$$P_s(a_i^t) = \frac{1}{N} \left(\alpha - (\alpha - \beta) \frac{i-1}{N-1} \right)$$

where $1 \leq \alpha \leq 2$ is chosen randomly, $\beta = 2 - \alpha$

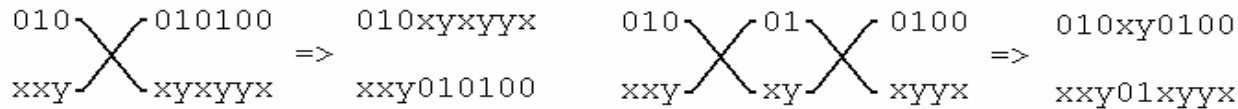
3) uniform rank-based selection
$$P_s(a_i^t) = \begin{cases} \frac{1}{\mu}, & 1 \leq i \leq \mu \quad \mu \leq N \\ 0, & \mu < i \leq N \end{cases}$$

4) tournament selection – $2 \leq m < N$ individuals are selected randomly and the best one is selected as parent. The process is repeated until intermediate population is not formed.

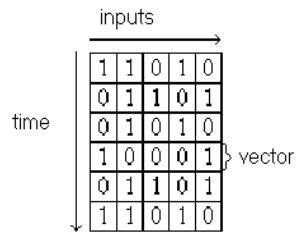


Individuals encoding & crossover schemas for:

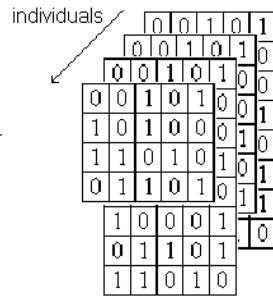
a) combinational circuits test generation:



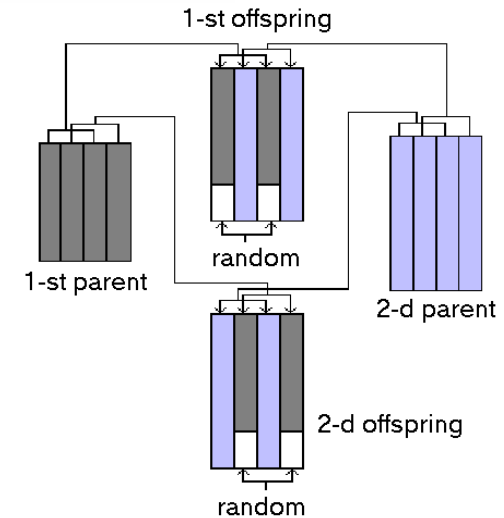
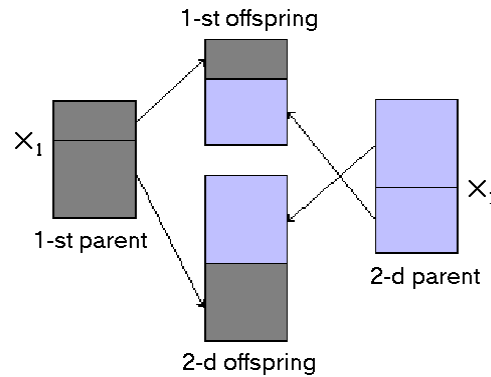
b) sequential circuits test generation:



a) individual



b) population



Mutation schemas:

- a) deleting one randomly located input pattern
- b) adding one input pattern to random position
- c) random bit inversion in test sequence



Intermediate population reduction:

- 1) *Pure reinsertion* – produce as many offspring as parents and replace all parents by the offspring
- 2) *Elitist reinsertion* – produce less offspring than parents and replace the worst parents according to fitness function values
- 3) *Uniform reinsertion* – produce less offspring than parents and replace parents uniformly at random
- 4) *Proportional fitness-based reinsertion* – produce more offspring than needed for reinsertion and reinsert only the best offspring
- 5) *Selective reinsertion* – produce as many offspring as parents, place parents and offspring in reproduction group together, rank individuals in reproduction group and reinsert best N individuals



Fitness functions in ASMID-E system:

$f_1(v, f)$ - the weighted number of the lines with different values in fault and fault-free circuits or number of changes of signals on the outputs of logical gates ;

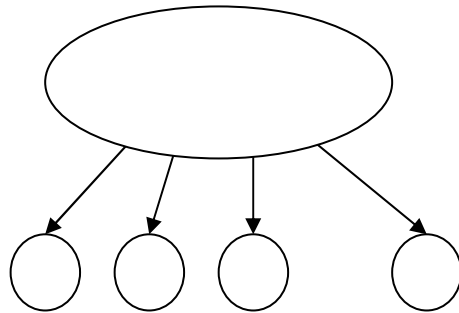
$f_2(v, f)$ - the weighted number of flip-flops with different values in fault and fault-free circuits or number of changes of signals on the outputs of flip-flops;

$h(v, f) = c_1 f_1(v, f) + c_2 f_2(v, f)$ - fitness function for test input vector v

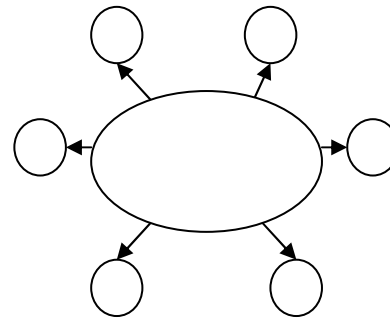
c_1 c_2 - normalization constants

$H(s, f) = \sum_{i=1}^{length} L^i * h(v_i, f)$ - fitness function for test input sequence

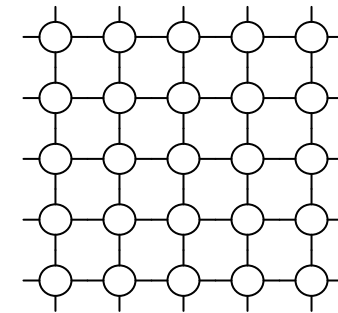
Different realization of parallel GA



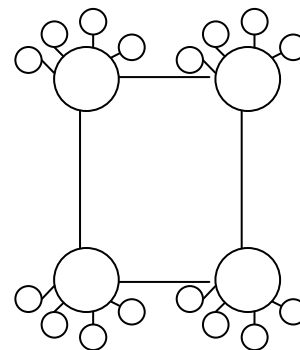
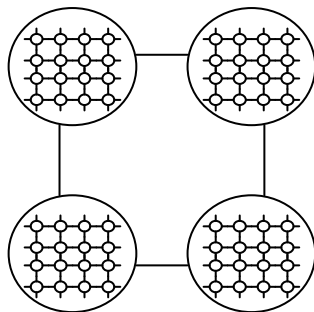
a) worker-farmer



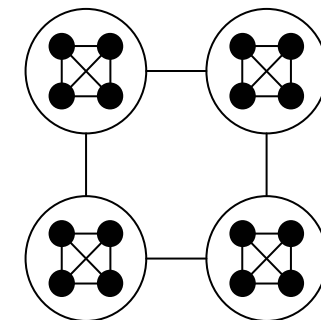
b) model of islands



c) cellular model



d-g) hybrid models





Worker-farmer model is based on classical “simple” GA with calculations which are executed in parallel. It is faster than “simple” GA and usually calculates on different processors fitness-functions values for different individuals (approximately equal complexity).

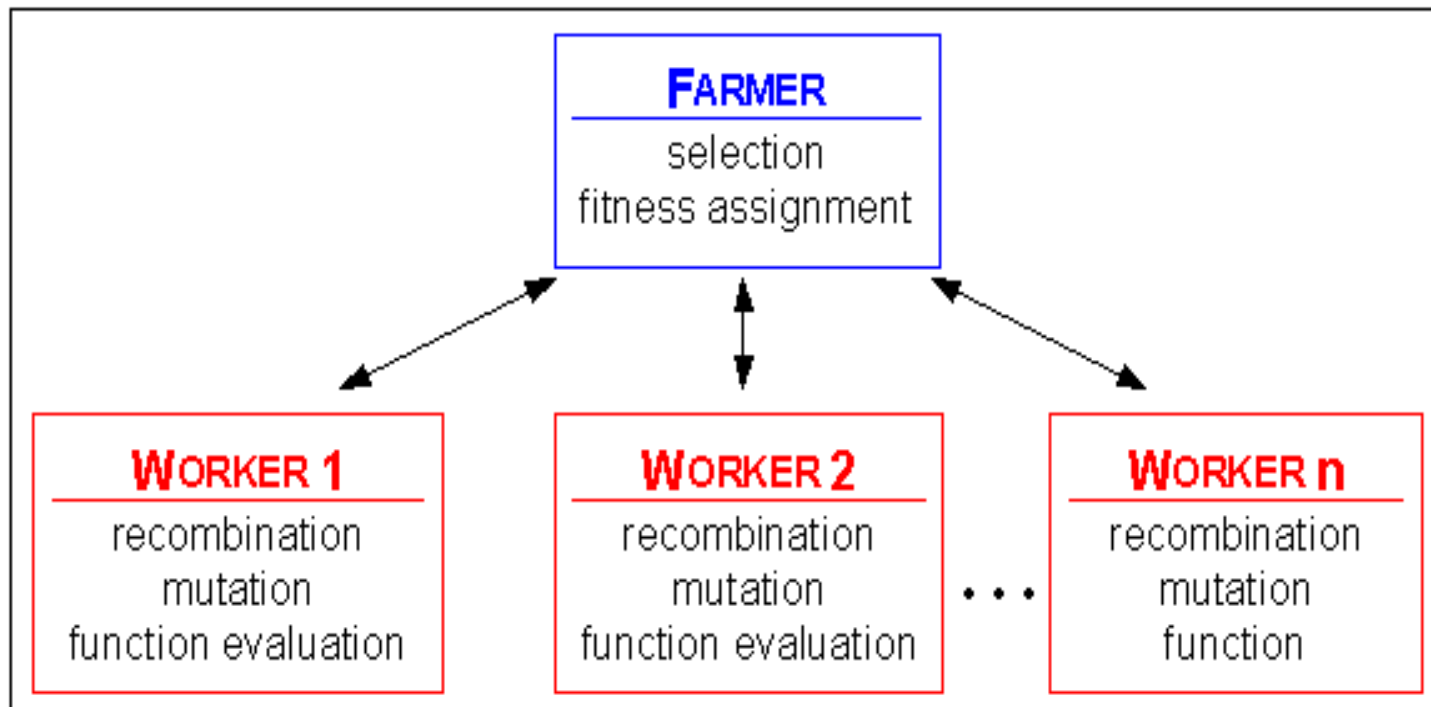
In **Model of islands** the set of subalgorithms are executed in parallel. They are exchanged with some individuals during search process. And each “island” - subpopulation corresponds to its own processor.

Cellular model is based on spatially distributed population where evolutionary interactions are possible only between the closest neighbor individuals. Individuals are sited in the nodes of some regular structure - net.

In **Hybrid models** two-level approach is used in parallelization process. In d) model at high level is applied model of islands and at low level - cellular model. In e) at low level is implemented Worker-farmer model. And finally in model g) model of islands is applied at both levels.

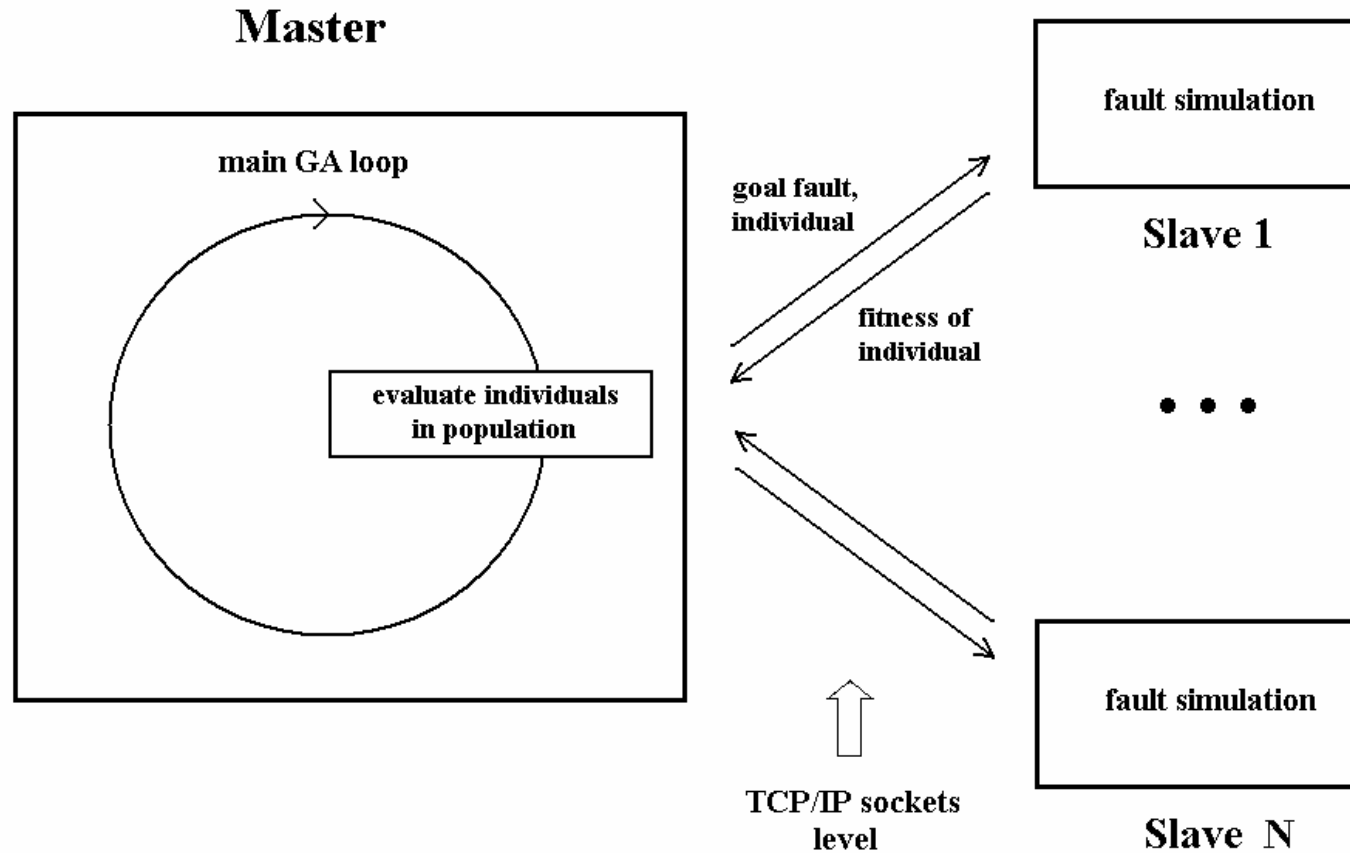


Parallel GA in test generation –
global model “worker/farmer”





Data flow diagram in “master-slave” realization of GA





Master process algorithm for GA generation in “master-slave” model

```
Master_of_GA_test_generation(circuit,list_of_clients)
{
  load_circuit_description_and_send_to_clients();
  make_list_of_faults()
  while( stop_criterion_not_reached() )
  {
    generate_start_population_and_select_goal_fault();
    for( i=0 ; i<MaxPopulations ; i++ ) // main GA loop
    {
      for( j=0 ; j<PopulationSize ; j++ )
      {
        Selection(ParentA,ParentB);
        Do_GA_Operations (ParentA,ParentB,Offspring);
        Add_in_intermediate_population (Offspring);
      }
      for( j=0 ; j<number_of_clients ; j++ )
        evaluate_individuals_in_parallel_on_clients();
      ConstructNewPopulation();
      if( test_sequence_found ) additional_simulation();
    }
  }
}
```

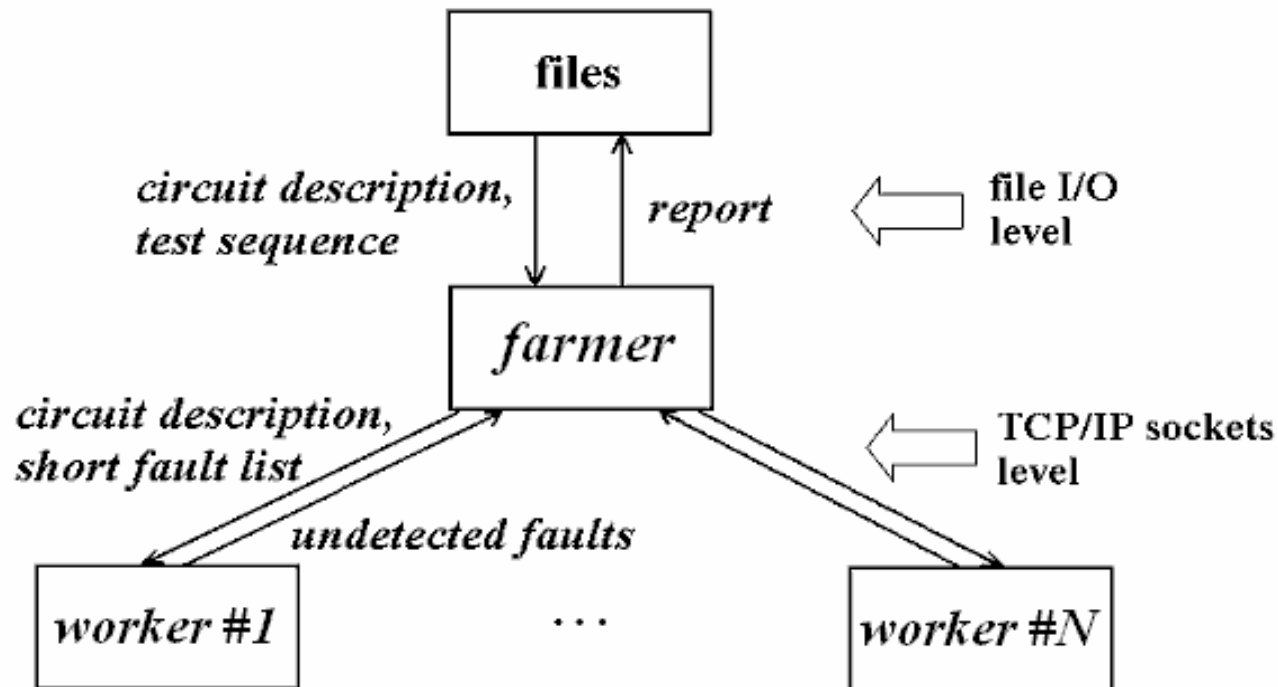


Slave process algorithm for GA generation in “master-slave” model

```
Slave_of_GA_generation()  
{  
    search_of_master_process();  
    if( master_was_found() )  
    {  
        receive_circuit_description();  
        while( stop_signal_not_received )  
        {  
            receive_short_fault_list();  
            receive_individual();  
            parallel_fault_simulation();  
            send_fitness_of_individual();  
        }  
    }  
}
```



Data flow diagram for distributed fault simulation





Farmer process algorithm for distributed simulation

```
simulation(circuit, test)
{
    number_of_workers = search_of_workers();
    if( number_of_workers != 0 )
    {
        input_circuit_description();
        input_test();
        make_full_fault_list();
        partition_the_fault_list(number_of_workers);
        for( i=0 ; i< number_of_workers; i++ )
        {
            send_to_worker_i_circuit_description ();
            send_to_worker_i_part_of_fault_list();
            send_to_worker_i_test_sequence();
        }
        for( i=0 ; i< number_of_workers; i++ )
        {
            receive_list_of_undetected_faults();
        }
        make_report();
    }
}
```



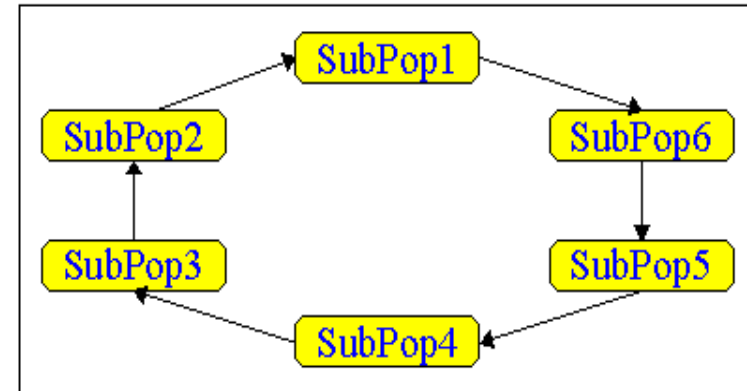
Worker process algorithm for distributed simulation

```
worker_process_fault_simulation()  
{  
    search_of_farmer_process();  
    if( master_was_found )  
    {  
        receive_circuit_description();  
        receive_short_fault_list();  
        parallel_fault_simulation()  
        send_list_of_undetected_faults();  
    }  
}
```

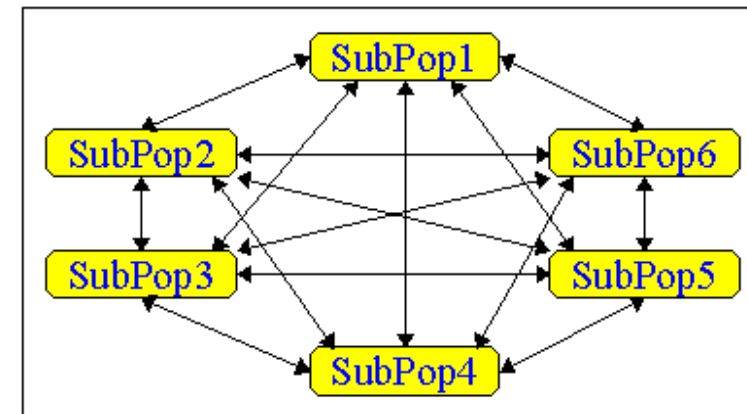
Parallel GA in test generation – *migration (island) model*.

The migration structures of individuals between subpopulations:

1) in a ring topology

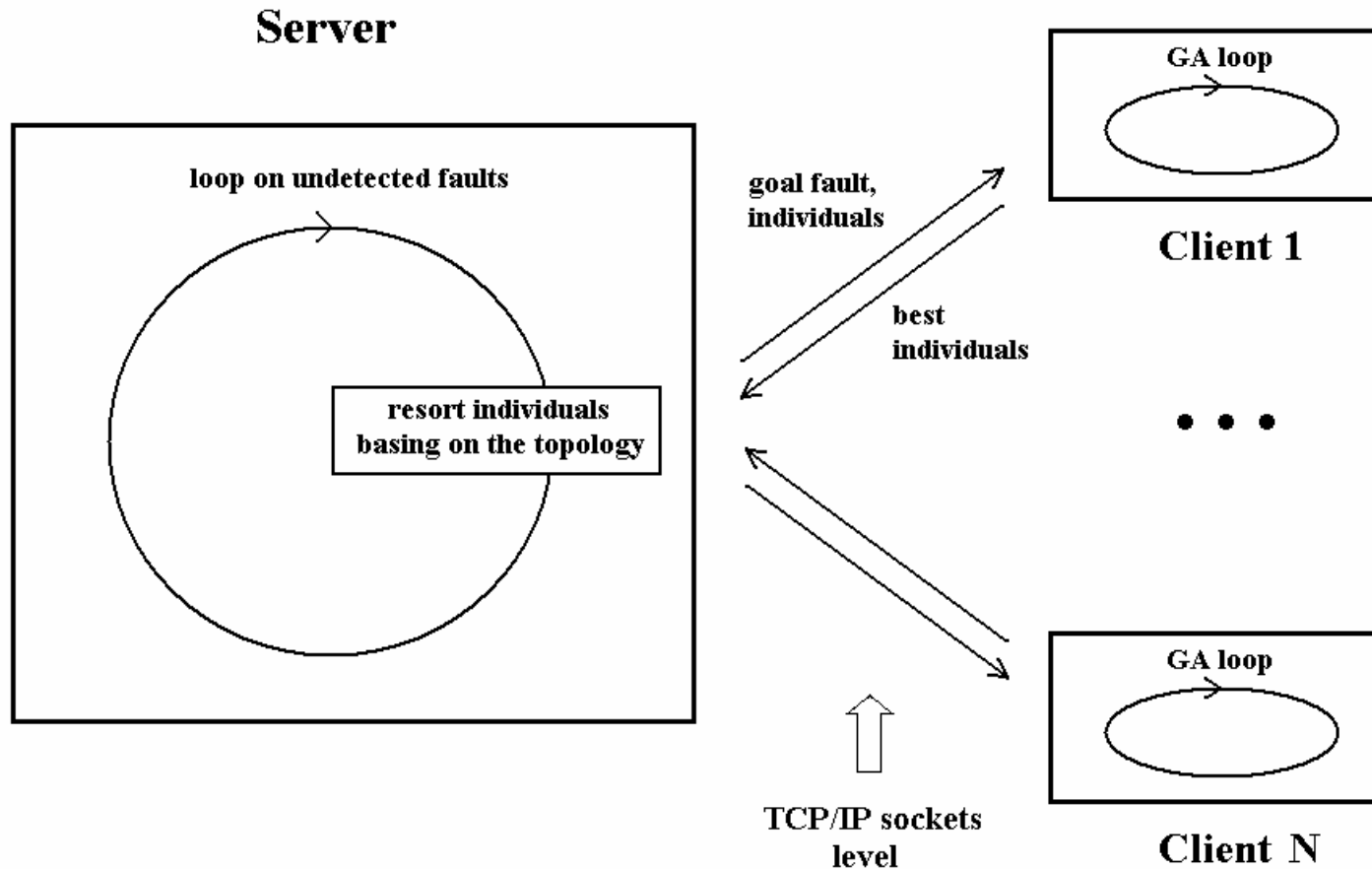


2) between all subpopulations
(complete net topology - unrestricted)





Data flow diagram in “islands” realization of GA





Server process algorithm for GA generation in “islands” model

```
Server_of_GA_test_generation(circuit,list_of_clients)
{
  load_circuit_description_and_send_to_clients();
  make_list_of_faults()
  generate_start_population();
  for( i=0 ; i<number_of_clients ; i++) send_GA-parameters_to_client_i();
  while( stop_criterion_not_reached() )
  {
    select_goal_fault();
    for( i=0 ;i<number_of_clients ) sent_goal_fault_for_client_i();
    while( goal_fault_not_detected() )
    {
      for( i=0 ;i<number_of_clients ) receive_N_individuals_from_client_i();
      resort_individuals_basing_on_topology();
      for( i=0 ;i<number_of_clients ) send_new_N_individuals_to_client_i();
    }
    add_found_sequence_to_TEST();
  }
  save_test_in_file();
}
```

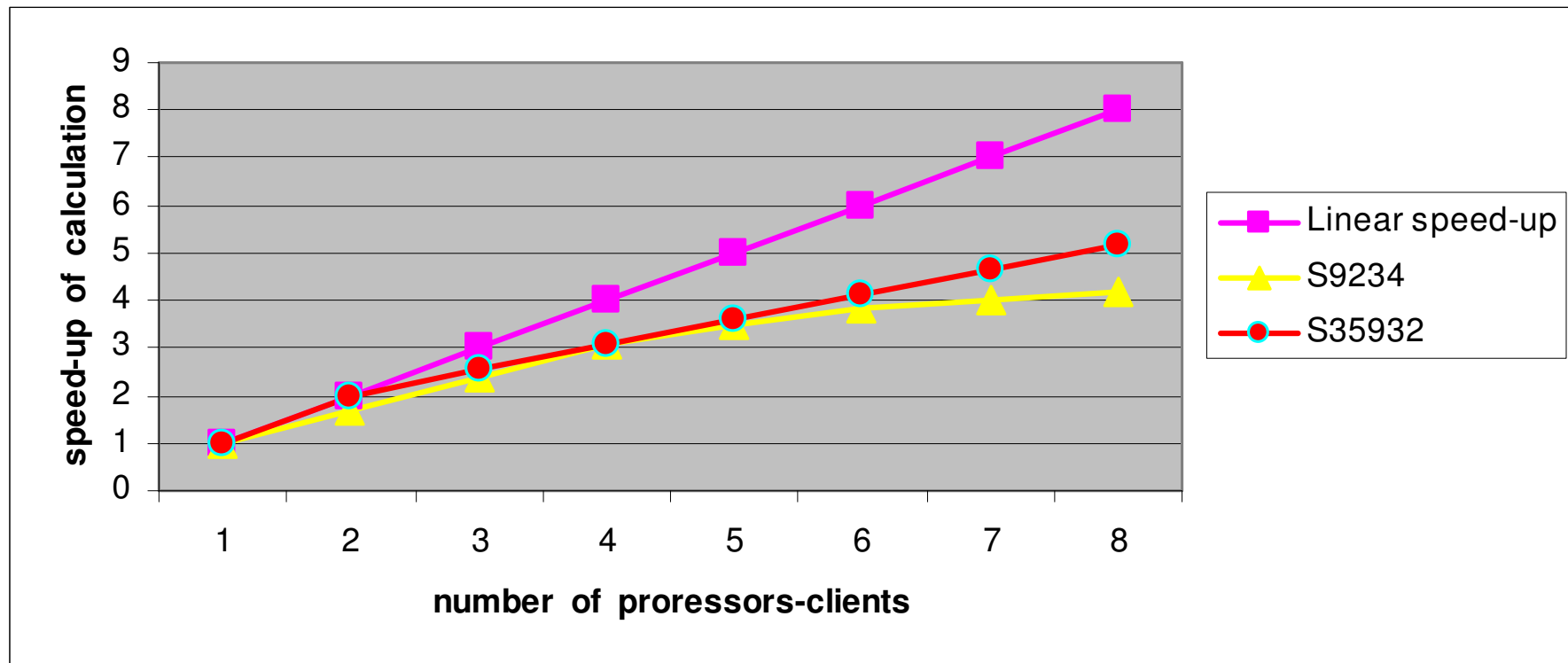


Client process algorithm for GA generation in “islands” model

```
Client_of_GA_test_generation()
{
    search_of_server();
    if( server_found() )
    {
        receive_circuit_description();
        receive_GA_parameters();
        while( stop_criterion_not_reached() )
        {
            receive_start_individuals_from_server();
            for( i=0 ; i<number_of_local_generation )
            {
                Make_GA_Loop();
            }
            send_M_best_individuals_to_server();
        }
    }
}
```

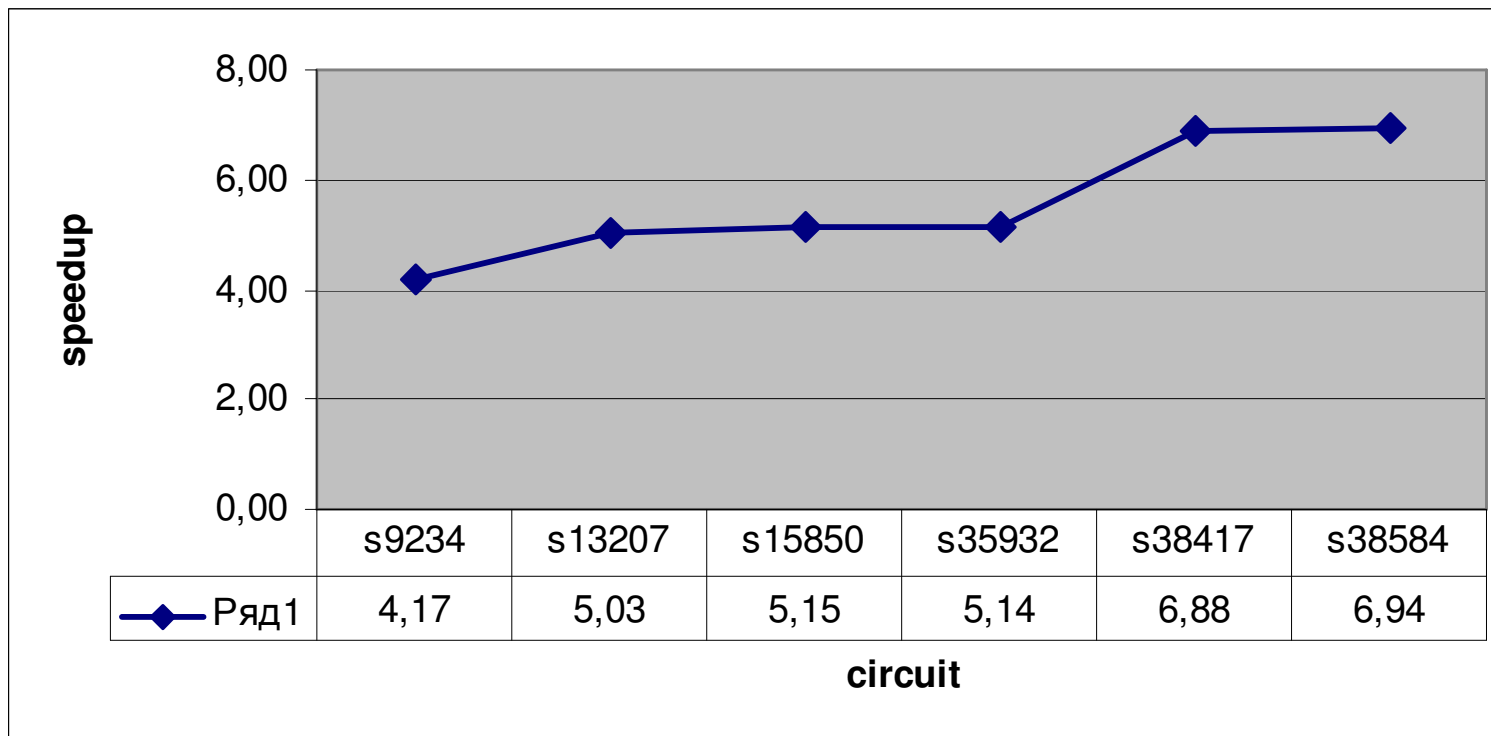


Speed-up of calculation for S9234 and S35932 circuits





Speed-up for large ISCAS-89 circuits with 8 processors realization



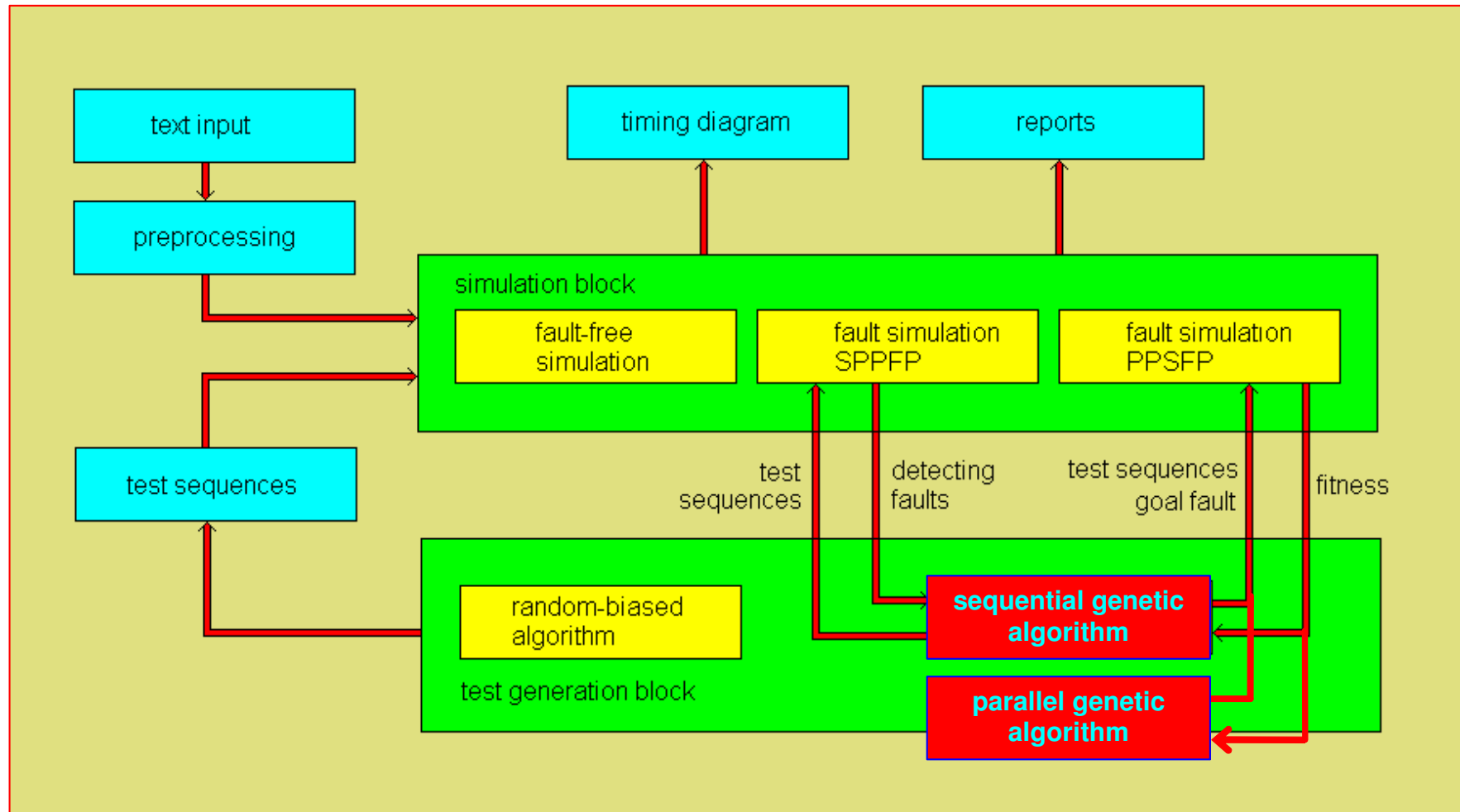


Experimental results for “island model”-based distributed GA-based test generation

Circuit from ISCAS89 benchmark	Speeding-up relatively to one processor	Fault coverage increase
S1196	1.59	+0.8%
S1238	1.8	+0/6%
S1423	1.1	+12.8%
S1488	6.1	+7.1%
S5378	5.16	+1.3%
S35932	5.35	+1.6%



ASMID-E functional flowchart





Conclusion

Parallel genetic algorithms could be applied to test generation

Obtained results confirm the effectiveness of test generation and fault simulation algorithms parallelization. “Farmer-worker” model in comparison with “island model” essentially easier in software implementation. But “island model” raises fault coverage of generated tests especially for large circuits. Therefore parallelization based on the “island model” has a reason only in case when generated tests have unsatisfactory fault coverage for “farmer-worker” model.



You can connect with the authors by e-mails:

•Yuriy A. Skobtsov:

skobtsov@kita.dgtu.ua

•Vadim Yu. Skobtsov:

skobtsov@iamm.ac.donetsk.ua

•Dmitry E. Ivanov:

ivanov@iamm.ac.donetsk.ua