Evolving Non-Determinism: An Inventive and Efficient Tool for Optimization and Discovery of Strategies

Technical Report

Hugues Juillé
Computer Science Department
Volen Center for Complex Systems
Brandeis University
Waltham, Massachusetts
<hugues@cs.brandeis.edu>

23 Nov 1994

Abstract

In the field of optimization and machine learning techniques, some very efficient and promising tools like Genetic Algorithms (GAs) and Hill-Climbing have been designed. In this same field, the Evolving Non-Determinism (END) model proposes an inventive way to explore the space of states that, combined with the use of simulated co-evolution, remedies some drawbacks of these previous techniques and even allow this model to outperform them on some difficult problems.
This new model has been applied to the *sorting network problem*, a reference problem that challenged many computer scientists, and an original one-player game named *Solitaire*. For the first problem, the END model has been able to build from scratch some sorting networks as good as the best known for the 16-input problem. It even improved by one comparator a 25 years old result for the 13-input problem! For the Solitaire game, END evolved a strategy comparable to a human designed strategy.

Keywords: evolutionary strategies, sorting networks, optimization techniques.

1 Introduction

The aim of this paper is to describe a model that implements a new optimization technique. This inventive model, called Evolving Non-Determinism (END), is largely inspired from simulated co-evolution, a fundamental principle coming from the field of Artificial Life.
In fact, the END model can be seen as many simple competing organisms that interact locally one with each other and which result in the emergence of a particular species of organisms that corresponds to the solution of the optimization problem.
The END model is extremely different from other well-known techniques: Genetic Algorithms (GAs), Hill-Climbing or Simulated Annealing (SA) in the way the information about the landscape (or the topology) of the space of states is used. This allows the END model to outperform these techniques in the case of problems for which there is only little information about the gradient of the landscape.

A drawback of a technique like GAs is that crossover and mutation operators used to evolve genes may create genes that correspond to an invalid solution. For some problems, this drawback can be such that the population size has to be very important to counterbalance this undesirable property.
For Hill-Climbing and SA, some operators have also to be designed in order to evolve solutions by finding some new solutions in their neighborhood. The design of such operators may be very elaborate for some problems.
Unlike these techniques, the END model doesn't care about solution representation or local neighboring solutions since solutions are generated incrementally and are always valid. That is, a solution is represented as a sequence of "components" each of them depending only on the previous components of the sequence. Encouraging results described in this paper, let us expect that a broader field of applications can be tackled by the END model.

In this paper, the END model is applied on two difficult "real-life" problems. The first one is the follow-up of an established problem since several approaches ([1, 5, 9]) have been used to try to improve some 25 years old results concerning sorting networks [7]. Actually, this problem was also at the origin of an early paper [11] in which GAs were used to try to replicate Hillis's experiment for the 16 input problem and in which some ideas of the END model were presented.
The second problem is a very simple one-player game for which the player tries to find a strategy to get a score as large as possible.

This paper is organized as follows. Principles and parameters of the model are presented in details in Section 2. In Section 3, we define a problem for which some mathematical results are well-known and then we use it to analyze efficiency of the model. Results for the two real-life problems are presented in

Section 4. A summary and possible future research are described in Section 5.

2 Evolving Non-Determinism

2.1 Principles

2.1.1 Description of Organisms

In the END model, selection and co-evolution are fundamental principles. Indeed, the model can be seen as a population of N organisms. These organisms live in a grid world, wrap-around, for which there are as many slots as organisms. Each slot is occupied by an organism and each organism works like a non-deterministic Turing machine.

2.1.2 Representation of the Space of States

We already said in the introduction that the kind of problems we want to solve are such that any solution can be built incrementally. We call such a problem an *incremental problem*.
Now, let us introduce some concepts and notations to define formally why we made this assumption.

Definitions:
Let T be an arbitrary tree and P be the incremental problem at hand.
In the following, σ represents any node of the tree T.
We also define:

$children(\sigma)$: the set of children of σ.
$depth(\sigma)$: the depth of the node represented by σ in T.
$parent(\sigma)$: the parent of σ, and
$ancestor(\sigma, i)$: the i^{th} ancestor of σ. Therefore:

$$ancestor(\sigma, 1) = parent(\sigma)$$
$$ancestor(\sigma, 0) = \sigma$$
$$ancestor(\sigma, depth(\sigma)) = root(T)$$

Then, we say that T is the *tree of solutions* for P if T can be defined as follows:

- If σ is an internal node then σ represents a *partial solution* for P.
- Otherwise, $children(\sigma) = \emptyset$ and σ is a leaf of the tree T. In that case, σ represents a *correct solution* for P.
- For any node σ, if $children(\sigma) \neq \emptyset$ then $children(\sigma)$ is the set of *correct and fair extensions* (or *correct and fair decisions*) of the partial solution represented by σ.

With the following definitions:

- A *correct solution* for the problem P is a solution for P.

- A *partial solution* for the problem P is an ordered set of decisions such that this set doesn't represent a correct solution for P but it can be extended to such a solution.
- A *correct extension* (or *correct decision*), noted $c\_extension(\sigma)$, is an extension (or a decision) that transform the current partial solution σ to either another partial solution or a correct solution.
- A *correct and fair extension* (or *correct and fair decision*), noted $c\_f\_extension(\sigma)$, is a correct extension which is useful (or fair). That means that the new set of correct and fair extensions is different from the one corresponding to σ or any ancestor of σ. That is:

  $$children(\sigma) = \{c\_f\_extension(\sigma)\}$$
  $$\{c\_f\_extension(\sigma)\} \subseteq \{c\_extension(\sigma)\}, \text{ and}$$
  $$\forall i \in [0..(depth(\sigma) - 1)],$$
  $$\{c\_f\_extension(\sigma)\} \neq \{c\_f\_extension(ancestor(\sigma), i)\}.$$

  Correct and fair extensions prevent to have infinite path in the tree T if every solutions for the problem P can be described by a finite number of incremental steps and if we don't consider solutions that are equivalent to another one by simply removing some useless extensions (fairness).

Now, if we call *space of states* the space of solutions for P then the space of states is equivalent to the leaves of the tree T.
In fact, any solution for P can be built incrementally by a sequence of decisions (or extensions) and therefore corresponds to a leaf of the tree T.
Conversely, any leaf of the tree T is, by construction, a solution for the problem.

We can conclude this formal presentation of the space of states by the following important observation.

Observation: The set $children(\sigma)$ depends only on previous nodes on the path from the root to the node σ. That means that any given solution for P can be retrieved in the tree T by incrementally building the path from the root to the leaf corresponding to this solution. This is done by correctly guessing at each node of the path which child in the set of children must be picked.
Therefore, the whole tree T doesn't have to be known to built such a path.

To be more concrete, let us consider the example of the Hamiltonian circuit problem. This decision problem can easily be transformed into an incremental problem. Indeed, we are going to show how the tree of solutions T could be built for any graph.

First, we arbitrarily pick a vertex of the graph G as the initial vertex and associate it to the root of the tree T. Now, for each vertex connected to this initial vertex, we create a child to the root. Each of these child nodes is labelled with its corresponding vertex. At this step, the depth of the tree is 1.
This operation is repeated for each of the terminal nodes of T: we consider the set of the neighbors in G of the vertex associated to the terminal node but not these vertices that already correspond to an ancestor of the terminal node. In that way, we can't create cycles (*ie:* we consider only fair extensions). Then, we create a child for each of these vertices and we link it to the considered terminal node. Again, new nodes are labelled with their corresponding vertex.
This process is repeated recursively until the tree T can't be extended.
Finally, for each node for which the depth equals the number of vertices in the graph, we create another child if there is an edge connecting the corresponding vertex to the initial vertex. Therefore, such edges are created if and only if there is an Hamiltonian circuit in the graph.
In that way, this tree represents exactly all possible paths beginning at the initial vertex and there is one leaf corresponding to each Hamiltonian circuit (if there are some). Moreover, if there are some Hamiltonian circuits, they correspond to the deepest nodes in this tree.

Of course, the tree T is not built (in the worst case, its size grows exponentially with the number of vertices), but any path from the root to a leaf can be built incrementally without the knowledge of the whole tree.
However, what our model would intend to do would be to find the longest path in this tree in order to decide whether or not there is an Hamiltoninan circuit in the graph.
In fact, we transformed a decision problem into an optimization problem based on the search of a longest path in a tree.

2.1.3 Co-evolution of Organisms

The way our model works is the following: Each organism selects a path from the root to a leaf in the tree of solutions associated to the problem at hand. This path is built incrementally, choosing uniformly randomly a child for each node encountered. Therefore, this path correspond to a correct solution.

Then, each organism can be seen as the member of a species regarding what the first choice is in its solution.
Moreover, a fitness can be associated to each organism. It can be the length of the path associated to its solution if we are looking for a shortest/longest path. It can also be a value attached to the leaf and which can be an evaluation of the solution. Whatever the fitness function is, the model looks for the leaf for which this fitness is optimal (therefore, this leaf may be different than the ones

corresponding to the longest/shortest path in the tree).

According to the value of the fitness function, a selection is performed in the population of organism. This selection is operated as follows:
For each slot, organisms in the neighborhood are considered and the organism with the highest fitness is copied into the slot providing this organism has a better fitness than the one actually occupying the slot. If several organisms have the same fitness in the neighborhood, one of them is picked randomly.
Otherwise, the slot occupant remains unchanged (and has eventually been copied to slots of its neighborhood).

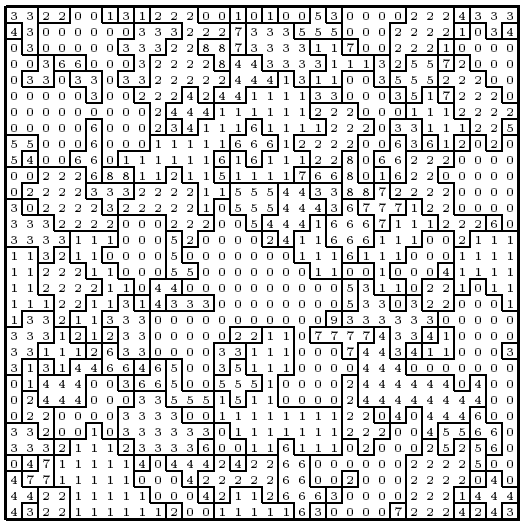
The important thing to notice is that each organism is linked to the solution it represents since its species is represented by the prefix of its solution. Therefore, when an organism is copied to a neighbor slot, its associated solution is also copied.

The idea of this selection is that an organism with a higher value for the fitness probably made a better choice for the first choice of its solution. Therefore, this organism is duplicated and take the place of some worse organisms. The model simulates the co-evolution of competing species.

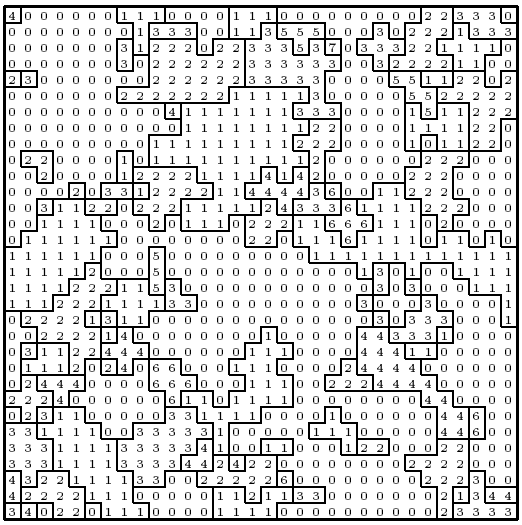
Now that the selection is completed, for the next round, every organism builds another solution. However, each organism keeps the first choice of its previous solution and builds its new solution beginning with this first choice. This process can be seen as a backtrack until depth 1 in the tree of solutions and the incremental building of a new random path starting from this node.
By proceding in that way, organisms don't change the species they belonged to (since it is related to the initial choice). Then, another round of selection is performed.

Figure 1 describes with a simulation how such a population of competing species evolves. This simulation was performed in a 1024 slots world. There are 10 species competing; each species is represented by a number from 0 to 9. The fitness function used for selection is the one described in section 3 for the study of the model performance. Looking at this simulated evolution, it can be seen that as rounds go off, the species represented by '0' takes more and more importance and it almost dominates all the other species after 16 rounds.

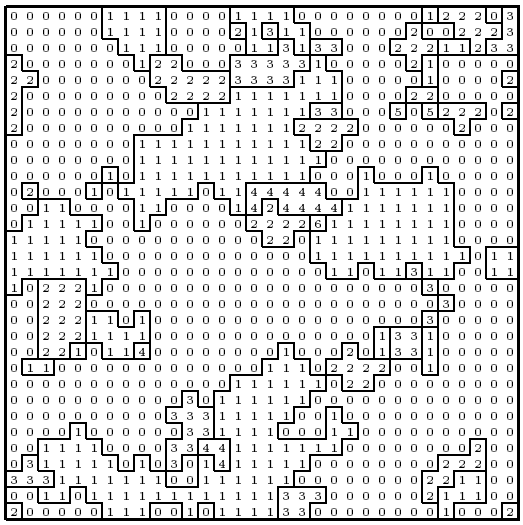
It is easy to understand that, proceding in that way, organisms corresponding to best first choices are theoretically stronger than others during the selection step and therefore duplicate more often. So, their species take more and more importance in the total population.
Therefore, the size of the population and the selection process must be compatible with the size of the problem (regarding the number of species created, for example) to expect this behavior to happen. In other words, the size of the sample must be large enough to allow a valid statistical analysis! Section 3 provides some measures to identify which efficiency may be expected for a given population size.



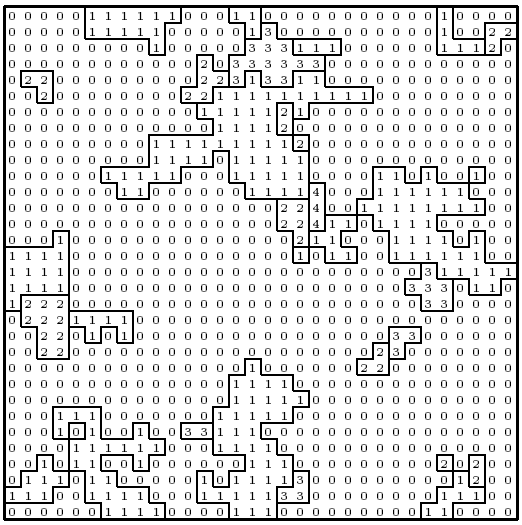
Step 5 - Disorder = 1794



Step 8 - Disorder = 1390



Step 12 - Disorder = 1046



Step 16 - Disorder = 812

Figure 1: *Simulation of the co-evolution of 10 competing species in a 1024 slots world.*

If we let this scenario run indefinitely, we can expect a species to overcome all other species and to be the only one in the population. However, this can take a very long time to happen, especially if there are some different first choices leading to equivalent optimal solutions.

The idea is to decide when to stop this scenario. For that purpose we use a parameter called *commitment degree*. This parameter is used by organisms when they "backtrack" and build a new solution. The commitment degree defines the depth of the backtrack. Initially, its value is one and it is increased according to a strategy.
Moreover, if the value of the commitment degree is increased then the species each organism belongs to is no longer defined by the first choice it made but by the C first choices, where C is the value of the commitment degree.
Therefore, when the commitment degree is increased each new species can be seen as a sub-species or a specialization of previous species.

The strategy to manage the evolution of the commitment degree is of course the key of the good working of the model. It is discussed in the following section.

2.1.4 Strategy to Manage the Commitment Degree

In fact, there are no rules to find the best strategy. For our experiments, we designed two different strategies.

The first one is the simpler since the commitment degree is increased every n rounds, where n is fixed. n has to be chosen astutely so that good species have time to grow and to reach a significant size.
The problem with this strategy is that the value of n is difficult to estimate *a-priori*.

The second strategy uses a measure of the state of the model called *disorder measure*. The idea of this measure is to have a way to detect when a species overcomes others to a degree such that we can consider that this species corresponds to the first best choices and such that, after increasing the commitment degree, sub-species of this overcoming species are significantly represented according to the total population size. The value of this measure tends to decrease as some species dominates the others. When this disorder measure reach a given threshold, the commitment degree is increased.
The drawback of this strategy is that it can take a long time for the disorder measure to reach the given threshold if there are some different first choices that are equivalent. This problem doesn't appear with the first strategy. That's why a combination of these two strategies has been used for real problems.

Thus, as the commitment degree increases, organisms commit themselves in

these earliest choices which seem to be the most attractive.

Finally, when the commitment degree can't be increased because it equals the length of the found out solution, the simulation stops.

2.2 Parameters of the model

From the description of the model in the previous section, it appears that a few parameters manage the way the model works. These parameters are the following:

Population size: as this size grows, the number of solutions generated at each round increases. The size of the "sample" is more important and it is more likely that species corresponding to optimal solutions don't disappear because of a too small number of representatives.

Neighborhood used for selection: Unformally, the intuition behind this parameter is the following:
If this neighborhood is large then we can expect that when a good solution is found out, it propagates more easily in the population. Therefore, the convergence to a good solution can be fast.
However, a large neighborhood may forbid the discovery of a better solution since it shrinks the space of explored solutions. It is not the case when a small neighborhood is used. But, the drawback of the later case is that convergence is slower.

Management of the commitment degree evolution: the strategy used to schedule increasements of the commitment degree is very important. Indeed, it must be such that the number of representatives of "interesting" species at the next round is sufficient to expect them to overcome other species with a high probability.

All these parameters will be analyzed experimentally in section 3.

2.3 Comparison to other Optimization Techniques

As we have already said, our model doesn't exploit information about local gradient of the landscape. Indeed, once a solution is built, only first choices of this solution are used for the next round. That means that we don't try to get some better solution in the neighborhood of this solution, using some gradient information as it is the case for Simulated Annealing or Hill-Climbing. This remark doesn't apply to Genetic Algorithms. In fact, the drawback of GAs are the operators used to evolve in the landscape. Indeed, cross-over or mutation may make very difficult the search of an optimal solution if the new genotype doesn't correspond to a valid solution. In the case of some problems, like the

two presented in this paper, this drawback is not negligible and good efficiency can only been reached if a very large population of genes is used.

To understand a little more easily how the END model works, let us make the following analogy:
Children of the root of the tree of solutions can be seen as a partition of the space of states. Each child (or species) corresponding to a particular sub-set of this partition.
Then, the selection process allows species that generate a better solution on average (regarding the fitness) to dominate other species. Such species correspond to the domains of the space of states for which the mean value for the fitness is larger. Therefore, at this stage, details and gradient of the landscape of the space of states are not considered. Roughly, we can compare this when we look at two mountains regions very far away and we expect that the highest peak is located in the mountain for which the average altitude is the higher. That is, we expect the highest peak to be in a region of high mountains!
Then, as the commitment degree increases, each domain is partitioned into smaller sub-domains and, therefore, details of the landscape take more and more importance.

Of course, it is easy to define a landscape such that this strategy doesn't work. For example, define a fitness such that the optimal correspond to a peak located in a region with a very low fitness and for which another region, far from this optimal peak, has a high average value. This strategy will be inclined to find out a local optimum in the region of high average fitness.
As we will see later in this paper, the space of states for the sorting network problem has this kind of landscape.

However, a second property of the END model has to be taken into account. This property is coming from the ability of the model to maintain a certain *degree of diversity*. This degree of diversity is directly related to the strategy used to manage the evolution of the commitment degree and it allows the model to not converge too quickly.

Finally, an analogy may also be done between the END model and a classical Artificial Intelligence heuristic search technique: *Beam Search* [2].
In this technique, a number of nearly optimal alternatives (the beam) are examined in parallel and some heuristic rules are used to focus only on promising alternatives, therefore keeping the size of the beam as small as possible. This technique proceeds like the END model in the sense that the search state space is described by a directed graph in which each node is a state and each edge represents the application of an operator that tranforms a state into a successor state. A solution is a path from an initial state to a goal state.
Therefore, the problem modeling is very similar for the two techniques. However, the main difference is that the END model doesn't use some heuristic rules but

only a fitness function that quantify the degree of quality of a solution. Beam search uses heuristic rules to prune the set of alternatives at each step of the incremental building of a path to a goal state. This pruning is performed by the END model during selection but it is performed in an *auto-adaptive* way: no heuristic is provided to the model.
The END model evolves by itself a strategy of search in the space of states.

3 Experimental Analysis of the model

3.1 The Reference Problem

The problem used for experiments is called the *ramp problem* which is in fact a sorting problem. Hillis first addressed this sorting problem, evolving some organisms he called "Ramps" [8] and he discovered how difficult it can be because of the topology of the space of states. However, the study of this problem allowed him to study with success some inventive features to evolve his population of organisms. Later, he even decided to train his modified version of the genetic algorithm to tackle a more challenging problem: the design of sorting-networks.

Formally, the ramp problem can be stated as follows:
Let S_n be an arbitrary set of n different integers we want to sort. For this purpose, the following fitness function is defined:

$$f_n(\sigma) = \text{number of } \textit{ascents} \text{ in } \sigma,$$

where:

- σ represents an arrangement of S_n, and
- an ascent is a pair of consecutive terms in σ such that the first term is smaller than the second one.

For example, if $S_{10} = \{0,1,2,3,4,5,6,7,8,9\}$:

$$f_{10}(\{0,3,5,7,2,1,4,6,8,9\}) = 7$$
$$f_{10}(\{0,1,2,3,4,5,6,7,8,9\}) = 9$$

So, if we look at the sequence of sorted items in S_n we get a sequence of increasing numbers, *ie*: a ramp.

This problem is obvious but is however very difficult for GAs when the genotype is coded in the trivial way, *ie*: each gene corresponding to one element of S_n. Indeed, for this problem, GAs are liable to provide a sequence of sorted sub-sets instead of a unique sorted set. For example, with the same set S_{10} as before, GAs will more likely find a local optimum solution like:

$$\{1,3,4,5,8,9,0,2,6,7\}$$

That is: 2 sorted sub-sets (*ie*: a double-ramp) for which the value of the fitness function is only 1 unit lesser than the optimal value.

In fact, the number of arrangements for which the function f_n equals a given value is a well-known problem. These numbers are called *Eulerian numbers* [3] and are noted $\left\langle \begin{array}{c} n \\ m \end{array} \right\rangle$. $\left\langle \begin{array}{c} n \\ m \end{array} \right\rangle$ represents the number of permutations of S_n that have m ascents. Their values are provided by the following formula:

$$E_n(m) = \left\langle \begin{array}{c} n \\ m \end{array} \right\rangle = \sum_{k=0}^{m} \left(\begin{array}{c} n+1 \\ k \end{array} \right) (m+1-k)^n (-1)^k$$

where m is the value of the fitness function for which we want to count the number of solutions.
We can check, using the symmetry property: $E_n(m) = E_n(n-(m+1))$, that:
$E_n(n-1) = 1$, *ie:* there is a unique optimal solution, and
$E_n(n-2) = 2^n - (n+1)$
This last formula tells us that the number of local optima for which fitness value is 1 unit lesser than the optimal value grows exponentially with the size of the set S_n.

The tree of solutions T corresponding to this problem is the tree representing every possible permutations. Each path from the root to a leaf corresponds to the incremental building of a set of size n by picking uniformly randomly one item in S_n which has not been picked yet.
Therefore, the degree of a node σ in T equals $(n - depth(\sigma))$, every leaves is of depth n and the total number of leaves is $n!$.
For our experiments, the algorithm used by each organism of the population to generate a solution to the problem works in that way and is presented in figure 2.

The initial value of the commitment degree is 0. So, the list `PARTIAL_SOLUTION` is empty at the beginning. When the value of the commitment degree increases, the list `PARTIAL_SOLUTION` keeps the elements corresponding to the first choices of the organism. The selection process ensures that organisms who made the best first choices are more likely to survive and spread over the population.

Experiments were performed on a Maspar MP-2 parallel computer. The configuration of our model is composed of $4K$ processors elements (PEs). In the maximal configuration a MP-2 system has $16K$ PEs. Each PE has a 1.8 Mips processor, forty 32-bit registers and 64 kilobytes of RAM. The PEs are arranged in a two-dimensional matrix.
This architecture is particularly well-adapted for our model since it is designed as a two-dimensional (grid) world. Each PE simulates one organism if we want to study a $4K$ population, but it can also simulate more organisms if we want a larger population.

Next sections present results of our experiments for different values of the

```
Begin with a list PARTIAL_SOLUTION of n items
/* where n = value of the commitment degree */

DO BEGIN
    Compute the set NON_USED_ITEMS of items not in
        PARTIAL_SOLUTION

    IF (not_empty(NON_USED_ITEMS))
        Pick one ITEM uniformly randomly in
            NON_USED_ITEMS
        PARTIAL_SOLUTION = append(PARTIAL_SOLUTION, {ITEM})
    END_IF
UNTIL (empty(NON_USED_ITEMS))

/* A list containing every elements of the initial set has */
/*  been generated.                                        */
```

Figure 2: END algorithm for each organism for the ramp problem.

model's parameters.

3.2 Evolution of Efficiency and Time Complexity for Different Schedule Strategies

In this section, we are interested to study the efficiency of the model for each of the two strategies we defined to manage the commitment degree.
We called the first one the *fixed schedule strategy*. In that case, the commitment degree is regularly increased after a fixed number of rounds.
The second one is called the *disorder measure strategy*. The disorder measure is a way to determine when the species distribution in the population is such that some species overcome the others. The less disorder is allowed, the more some species overcome the others and therefore, the less diversity is allowed (remember figure 1).

The aim of this section is to compare those two strategies regarding the efficiency of the model, that is the success rate to find out the best solution. Figure 3 presents the results for two different problem sizes. We completed our experiments using a 4096 slots world. For each parameter value, 50 runs have been performed. For the disorder measure strategy, the number of rounds has been determined by averaging the total number of rounds for the successful runs. Indeed, since the evolution of the commitment degree is managed by a threshold for this strategy, the total number of rounds can be different for one

run compared to another one.

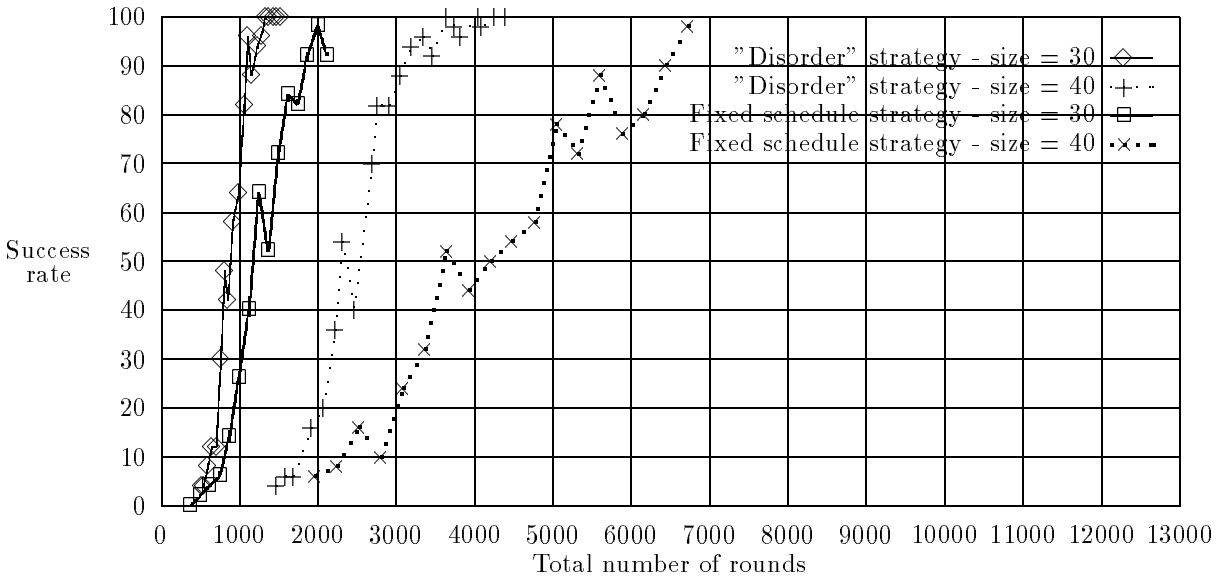


Figure 3: *Success rate versus the number of rounds for the disorder measure and the fixed schedule strategies.*

The conclusion of this experiment is that the disorder measure strategy allows the same efficiency with a smaller number of rounds, compared to the fixed schedule strategy. This is easy to understand since the number of rounds required for some species to overcome the others decreases as the difference of fitness increases between the species. Indeed, for the ramp problem, as the commitment degree increases, the differentiation is more and more important between species which took the first best choices and the others.
However, the fixed schedule strategy may be interesting in the case of problems for which there are several different first choices that lead to the optimal solution. In that case, it may take a very large number of rounds for some species to overcome the others. But, using the fixed schedule strategy the increasement of the commitment degree can be forced. A marriage of those two strategies allows a very good efficiency for the END model.

3.3 Evolution of Efficiency and Time Complexity for Different Selection Neighborhood

The aim of this section is to understand the effect of the size of the neighborhood considered during selection on the efficiency of the END model.
The size of the neighborhood is defined by its radius. The Manhattan distance is used to determine if an organism is part of a neighborhood.
For the experiments, a 4096 slots world and a problem size of 20 items were used. The radius was in the range: [1..8]. Then, using the fixed schedule strategy so that the computation time was the same for each value of the neighborhood radius, the curve of the success rate was plotted. Such a courb was plotted for a few values of the fixed number of rounds for the commitment degree increasement. For each point of this graphics, 50 runs have been performed. Figure 4 presents results of these experiments.

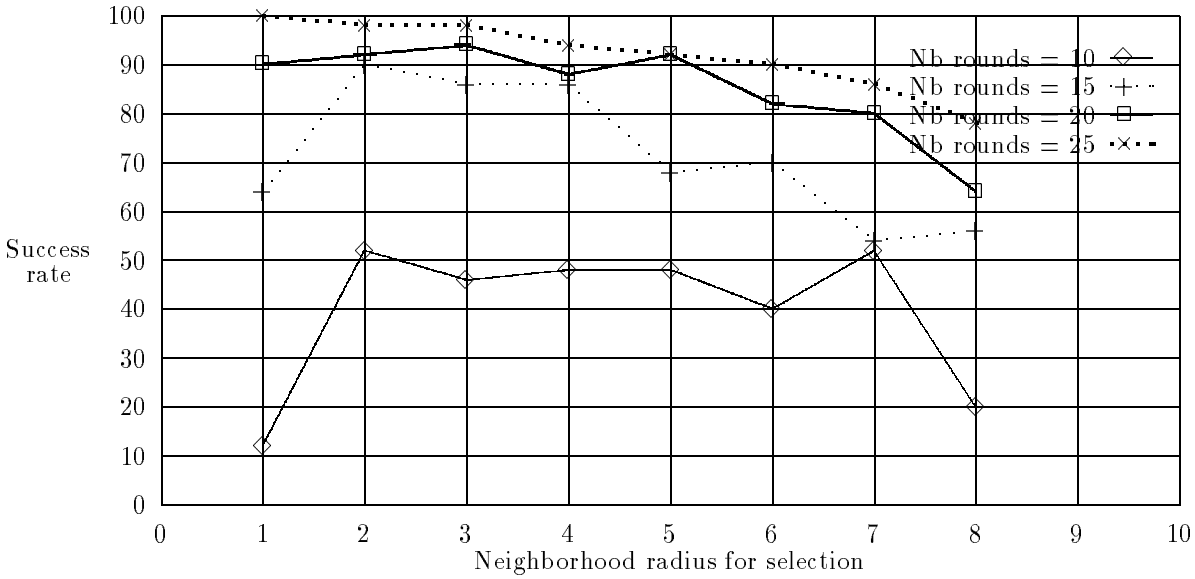


Figure 4: *Success rate versus neighborhood radius for selection, for different values of the number of rounds used for the fixed schedule strategy.*

The analysis of these results tells us that a large radius allows the model to find an optimal solution more efficiently when the total number of rounds is low. That is, an optimal solution is found out faster.
On the contrary, as the total number of rounds increases, a small neighborhood allows the model to find an optimal solution more efficiently because the convergence is slow. For large values of the radius, the convergence is too fast and

the optimal solution cannot be found at each run.
Therefore, experiments support the intuition presented in section 2.2.

3.4 Evolution of Complexity to Find the Optimal Solution

In this section, we are interested to study, for different problem size, the evolution of parameters value that allow the model to find out the optimal solution. We completed our experiments with a world size of 4096 slots and a neighborhood size for selection reduced to the 4 closest neighbors (radius = 1). This value has been chosen to allow the maximum freedom to explore the space of states (*ie:* to maintain diversity and to prevent premature convergence).

Problems for a size of the input set ranging from 20 to 60 elements have been considered. For each instance of the problem, we ran the model with different values for the threshold parameter and for each value of the threshold, we completed 50 runs. Then, the number of runs for which the model converged to the optimal solution was counted.
Results are presented in figure 5.

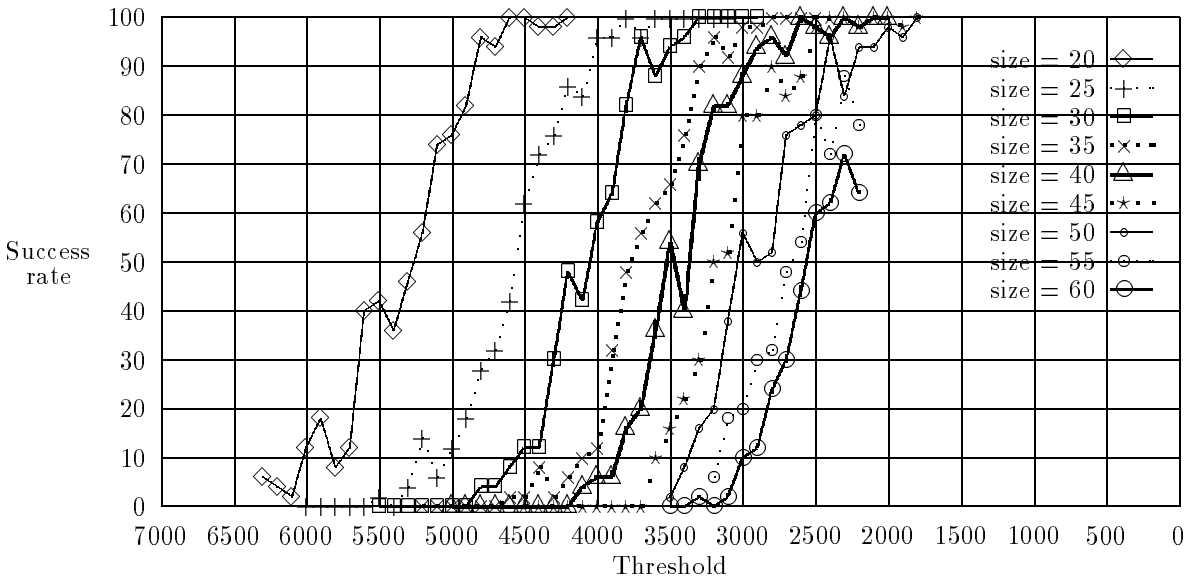


Figure 5: *Success rate versus threshold value.*

Now, an interesting measure to study would be the growth of the total number of rounds as the size of the problem increases to reach a given success rate. In that purpose, with the same data set, we measured for which value of the threshold a given success rate is reached and we looked at the average of the

corresponding number of rounds that were necessar for successful runs to find out the optimal solution. Results corresponding to this measure are presented in figure 6.
The very interesting result is that the courb is not linear but is logarithmic. This result is very surprising since the size of the space of states and the number of local optima grows exponentially with the problem size.
In fact, since the random generation of solutions takes linear time, this courb means that for a given probability $(1-\epsilon)$ and **for problem sizes within range for a given population size**, the END model is able to find the optimal solution in polynomial time.

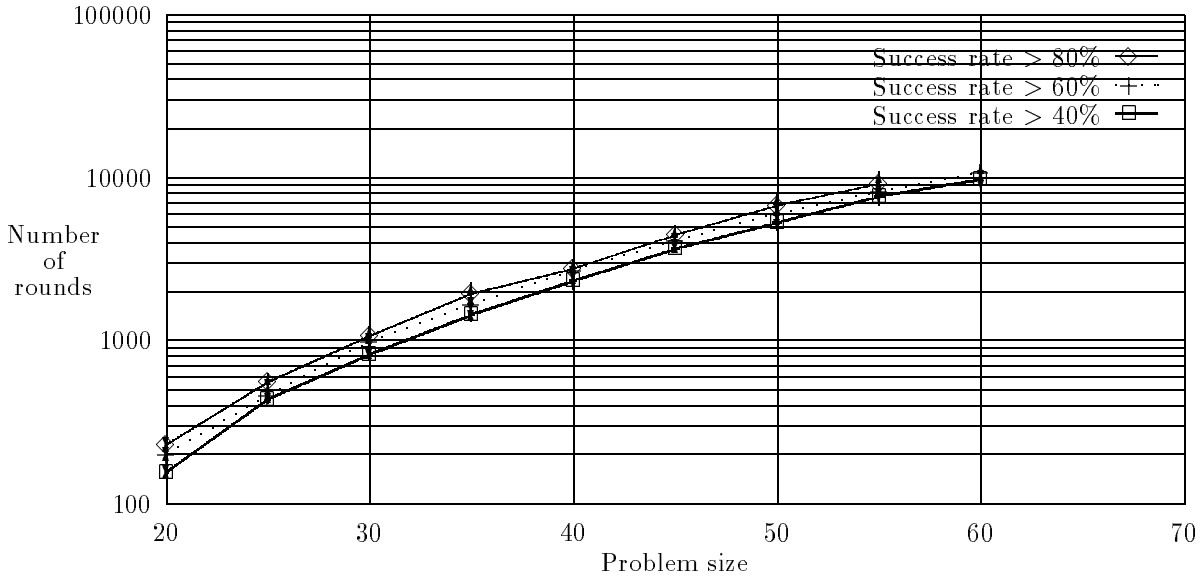


Figure 6: *Number of rounds to reach a given success rate.*

However, there is no miracle in that result. Indeed, it can be shown that the expected (or average) value of the fitness function for a randomly generated solution is:

$$\overline{f_{n,i}} = \frac{n}{2} - \frac{i-1}{n-1}$$

where:

- i is the first choice for the solution, and
- n is the size of the input set.

Then, if we consider the size of the sample that is required to determine with a low probability of error which first choice is the best one, it can be proven that this size increases at most as a polynomial of degree 4. Experimentally,

we determined that the number of rounds required to reach a given success rate grows a little slower than a 4^{th} degree polynomial. Therefore, this is as good or slightly better than a naive statistical approach.
In fact, the END model has two important advantages compared to such an approach:

- It doesn't need an *a priori* knowledge of the topology of the space of states.

- A strategy is evolved by the model itself to find out an optimum, by eliminating "unpromising" species very soon and by maintaining diversity to keep attractive species. Therefore, an auto-adaptive behaviour emerges while the model is searching for the optimum. This also allow a large improvement for the total work.

4 Results for Two Difficult Real-Life Problems

4.1 Sorting Networks

4.1.1 Presentation

An *oblivious comparison-based algorithm* is such that the sequence of comparisons performed is the same for all inputs of any given size. This kind of algorithm received much attention since they allow an implementation as circuits: comparison-swap can be hard-wired. Such an oblivious comparison-based algorithm for sorting n values is called an n-input *sorting network* (a survey of sorting networks research is in [7]).
There is a convenient graphical representation of sorting networks as the one in figure 7 which is a 10-input sorting network. Each horizontal line represents an input of the sorting network and each connection between two lines represents a *comparator* which compares the two elements and exchange them if the one on the upper line is larger than the one on the lower line. The input of the sorting network is on the left of the representation. Elements at the output are sorted and the larger element is on the bottom line.

Performance of a sorting network can be measured in two different ways:

1. Its *depth* which is defined as the number of parallel steps that it takes to sort any input, given that in one step disjoint comparison-swap operations can take place simultaneously. Current upper and lower bounds are provided in [9], in which the depth for 9- and 10-input sorting networks is proved to be optimal using an algorithm executed on a supercomputer.
   Table 1 presents these current bounds on depth for sorting networks for $n \leq 16$.
2. Its *length*, that is the number of comparison-swap used. Optimal sorting networks for $n \leq 8$ are known exactly and are presented in [7] along with the most efficient sorting networks to date for $9 \leq n \leq 16$. Table 2 presents

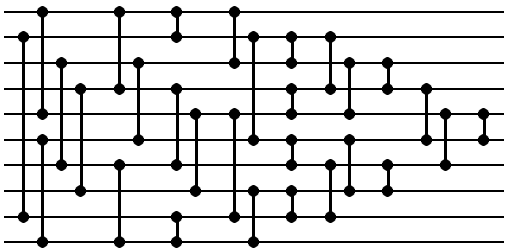


Figure 7: *A 10-input sorting network using 29 comparators and 9 parallel steps* [7].

inputs	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Upper	0	1	3	3	5	5	6	6	7	7	8	8	9	9	9	9
Lower	0	1	3	3	5	5	6	6	7	7	7	7	7	7	7	7

Table 1: *Current upper and lower bounds on the depth of n-input sorting networks.*

these results.
The 16-input sorting network has been the most challenging one. Knuth [7] recounts its history as follows. First, in 1962, Bose and Nelson discovered a method for constructing sorting networks that used 65 comparisons and conjectured that it was best possible. Two years later, R. W. Floyd and D. E. Knuth, and independently K. E. Batcher, found a new way to approach the problem and designed a sorting networks using 63 comparisons. Then, a 62-comparator sorting network was found by G. Shapiro in 1969, soon to be followed by M W. Green's 60 comparator network.

inputs	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
comparators	0	1	3	5	9	12	16	19	25	29	35	39	46	51	56	60

Table 2: *Best upper bounds currently known for length of sorting networks.*

4.1.2 Previous work

Most of the previous work to find good sorting networks focussed on the 16-input problem.

The first person who used optimization technics to design sorting networks is W. Daniel Hillis. In [5], he used GAs and then co-evolution to find a 61-comparator, only one more sorting exchange than the construction of Green.

However, Hillis considerably reduced the size of the search space since he initialized genes with the first 32 comparators of Green's network. Indeed, since the pattern of the last 28 comparators of Green's construction is not very intuitive, one can think that a better solution exists with the same initial 32 comparators.

In a previous paper, overviewing the END model [11], a 60 comparator sorting network was found out with the same initial conditions as Hillis, that is as good as the best known. Two attempts were also done on the 15-input problem and the model was able to find two 56 comparators sorting networks, again as good as the best known, with no initial conditions.

Ryan [10] applied a Genetic Programming approach to the problem of 9-input sorting networks. Kim Kinnear also used GP, but in the area of adaptive sorting algorithms, to find an algorithm to sort n elements in $O(n^2)$ time [6].

Ian Parberry ([9]) worked on the optimal depth problem and found out optimal values for 9 and 10-input sorting networks using an efficient exhaustive search.

4.1.3 Non-Deterministic Sorting Network Algorithm

The aim of this non-deterministic algorithm is to generate incrementally and randomly some valid sorting networks. Each organism runs this algorithm (see figure8), making some random choices until a valid sorting network is found. Moreover, this algorithm can generate only valid and fair sorting networks. That is, valid sorting networks with no useless comparators.

Valid sorting networks are built using the *zero-one principle*, which is a special case of Bouricius's theorem.

Zero-one principle: If a network with n input lines sorts all 2^n sequences of 0's and 1's into nondecreasing order, it will sort any arbitrary sequence of n numbers into nondecreasing order.

Therefore, we only consider all 2^n possible inputs instead of the $n!$ permutations of n distinct numbers to incrementally build a sorting network.

The fitness of a sorting network is represented by its length. However, for the selection process, ties are broken using the depth of the sorting networks. In that way, we also generate some sorting networks with a few number of parallel steps.
Then the selection process propagates the best solutions.

```
Begin with an empty or a partial network

DO BEGIN
        Compute the set of significant comparators

        IF (set of significant comparators IS NOT empty)
            Pick one of these comparators randomly and add
            it to the existing partial network
        END_IF
UNTIL (set of significant comparators is empty)

/* A valid sorting network has been generated */
```

Figure 8: END algorithm run by each organism.

4.1.4 Results

Results for the 16-input problem initialized with the first 32 comparators of Green's sorting network will not been presented. The last version of the model is able to evolve a sorting network as good as the best known with a success rate of almost 100% within 10 to 15 minutes.
Only results of evolved sorting networks with no initialization are described.

In order to increase the probability of finding good sorting networks, we used a $64K$ population size, each processor of the Maspar simulating 16 organisms. Therefore, because of the high degree of parallelization of this model, computation time provided in this section would be divided by 4 with the maximal configuration of this parallel computer ($16K$ processors) and by 16 with an hypothetical $64K$ processors Maspar.

Before presenting results, let us come back to a previous observation.
As it has been said in section 2.3, the landscape of the space of states is such that optimal solutions don't correspond to the species which perform well for lower value of the commitment degree. That means that if we let the model evolve until there are only a few species remaining then the probability that the best solution be found out is very low. This behaviour can be oberved in figure 9 which shows the evolution of the success rate as the value of the threshold decreases. First the success rate increases but, once the diversity degree is forced to be lower by decreasing the threshold value (the disorder measure strategy is used), the success rate also decreases.

This explains why one needs to maintain diversity to be able to find out an optimal solution.
We have been interested to tackle two different sorting network problems:

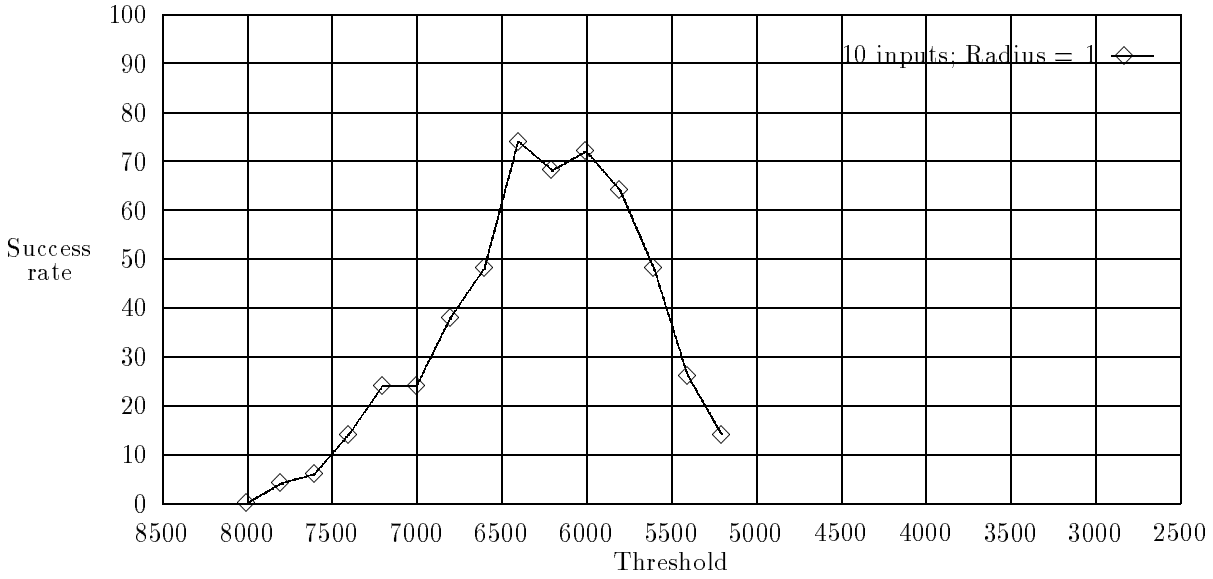


Figure 9: *Success rate for the 10-input sorting network problem versus the threshold value for the disorder measure strategy.*

1. The 13-input problem because of the large gap between the best known result and the 12-input one.
2. The 16-input problem because it is the most challenging one.

For the 13-input sorting network problem, we ran the END model 6 times with different values for the selection neighborhood radius (between 1 and 5) and the threshold. Each run was completed in about 8 hours. For 2 runs, we got a sorting network better than the best known. That is, the END model discovered two sorting networks using only 45 comparators, one comparator less than the best current known. Moreover, those two sorting networks use 10 parallel steps which is very good since to get smaller delay time one often has to add one or two extra comparator modules ([7]) and the best known delay for the 13-input problem is 9.
Figure 10 presents those 13-input sorting networks.

For the 16-input sorting network problem, we ran the END model 3 times. Each run was completed in a period of 48 to 72 hours. However, only 12 to 18 hours would be required with the maximal configuration for the Maspar (*ie:* using $16K$ processors). For 2 runs, a 60 comparator sorting network was found out, each of them using 10 parallel steps! This is as good as the best human-built sorting network designed by M. W. Green. Those sorting networks were entirely designed from scratch by the END model (*ie:* there was no initial comparators

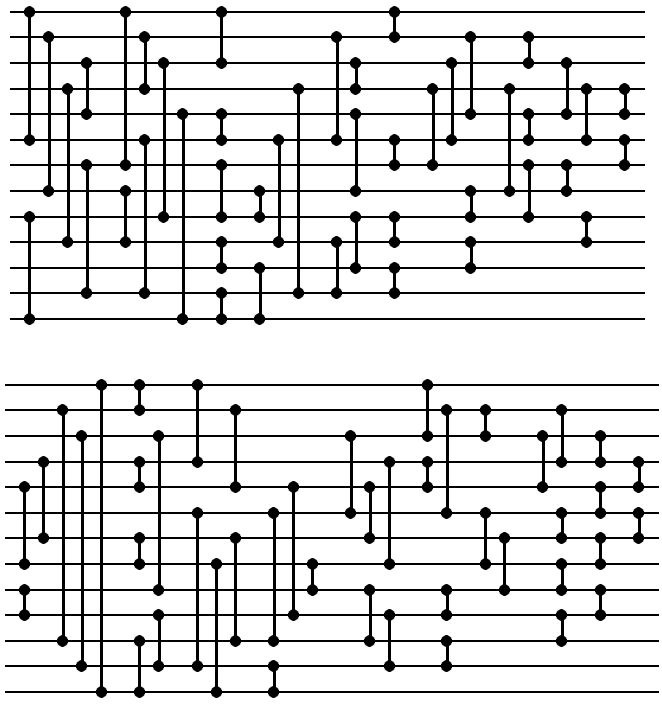


Figure 10: *Two 13-input 45-comparators sorting networks using 10 parallel steps.*

as it was the case the previous times this problem was tackled using computers). Figure 11 presents these two 16-input sorting networks.

As a remark, it is interesting to notice that the 4 last parallel steps of those two sorting networks are identical (but are however different from Green's construction).

4.2 A One-Player Game: Solitaire

4.2.1 Presentation of the game

This second problem is an original one and no one has published about it. However, it is an interesting one since it shows how difficult the modelling of a problem can be for other classical optimization techniques.

To play this game, you only need a piece of paper with a grid layout and a pencil. First, the player draws a fixed initial pattern composed of crosses, like the left picture in figure 12. The rule is that a cross can be added at

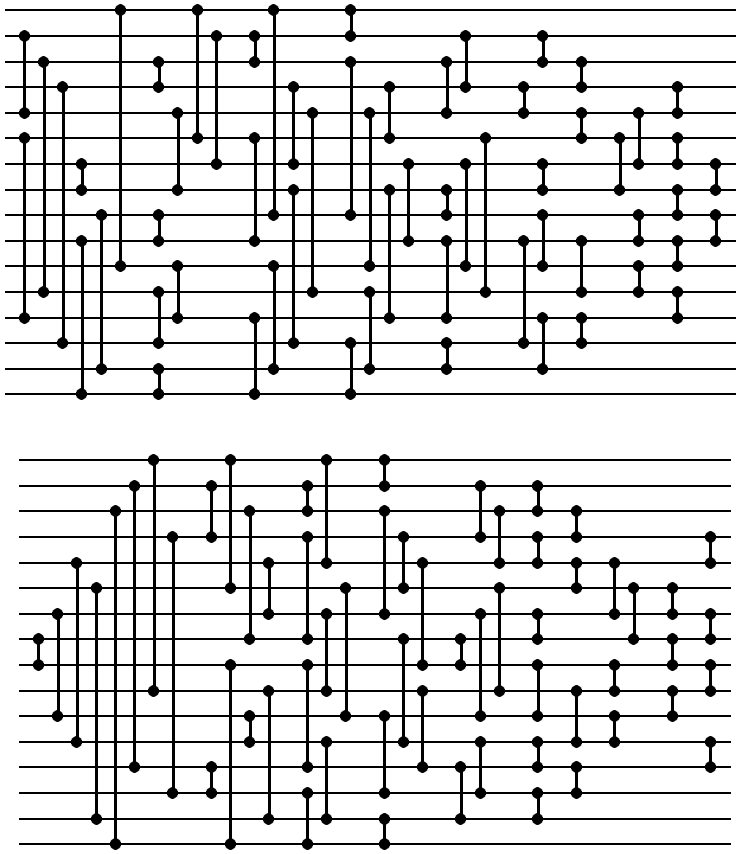


Figure 11: *Two 16-input 60-comparators sorting networks using 10 parallel steps.*

an intersection of the grid layout if and only if it allows the drawing of a line composed of 5 crosses that do not overlap another line. This line may however be the continuation of another line or may cross another line. That is, the new line can share at most one common cross with another line. This new line can be drawn vertically, horizontally or diagonally.
The right picture in figure 12 shows a possible configuration of this game after a few moves. Crosses of the initial pattern are circled to be identified more easily. Now, the goal of this game is simply to draw as many lines as possible!
Therefore, it is clearly an optimization problem.

If this game is played by hand, one can see that a good strategy is difficult to elaborate. After a few plays, a score of 70-80 lines is relatively common. However, to reach 90 lines is less obvious and a score greater than 100 lines is exceptional.

One can think that, maybe, there is a pattern of moves such that, it would be possible to repeat it infinitely. Alas, it can be proven that the number of moves is finite. That is, there is an optimal configuration for this game with a maximal number of lines. However, no tight upper bound has been established for this maximal number.

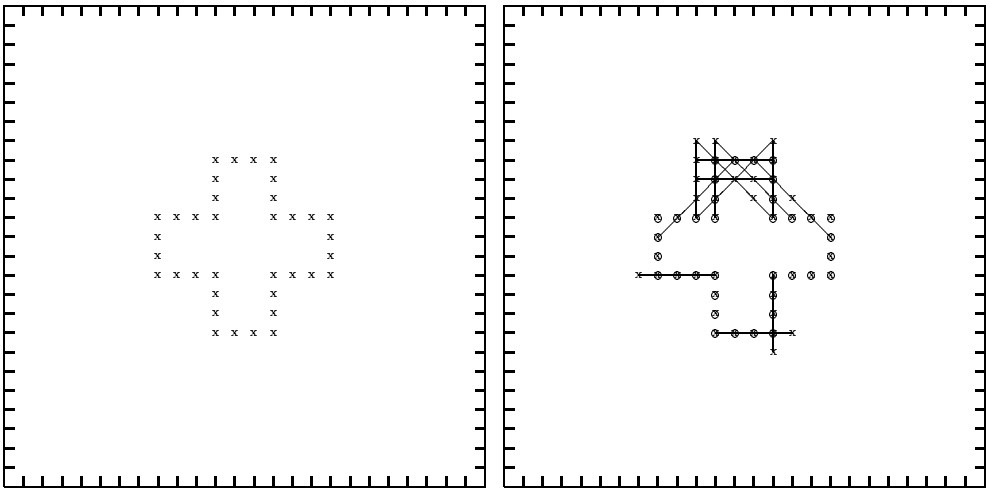


Figure 12: *The initial configuration and a possible configuration after 13 moves for the Solitaire game. For a clearer picture, the grid layout is not drawn but is represented by the rules on the border.*

This game is interesting because it is a typical example of a problem which seems impossible to code using other optimization technique like GAs or Hill-climbing. That shows that the END model is a tool to address new complicated problems.

4.2.2 Results

The most recent result is a 110 lines game configuration. This result has been found out with a population size of 1024 and required two weeks of computation on a workstation.

By curiosity, we were interested to know the configuration of the game after the first 40 moves (this configuration is shown in figure 13). Then, we were very astonished to observe that most of the moves were located in the same area of the game board. It is the same as our best own strategy for hand-playing! Therefore, the END model evolved a strategy by itself which is comparable to an "experienced" human player!

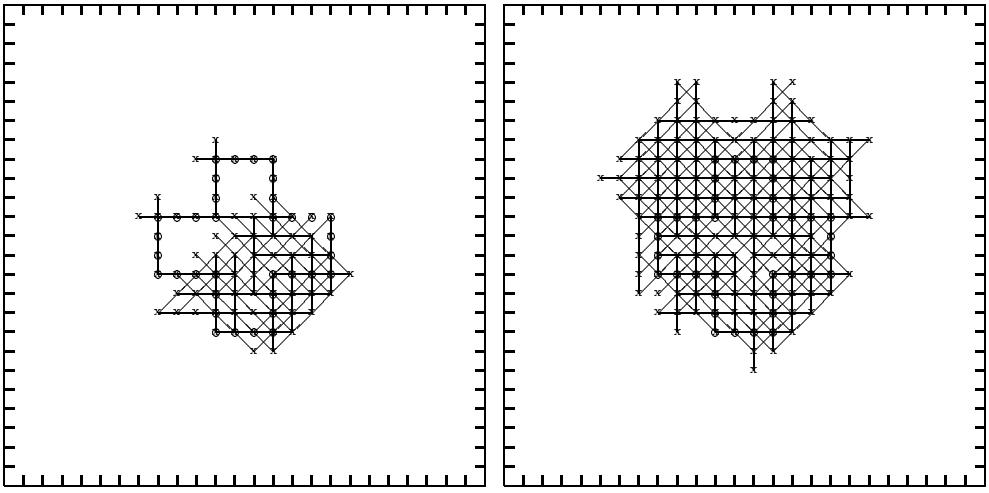


Figure 13: *The best configuration found out for the Solitaire game, using 110 lines (on the right); and the configuration for this best play after 40 moves (on the left).*

5 Conclusion

This paper presented an inventive and very promising technique. By using a new approach for the search in the state space and a particular way to construct incrementally solutions, this model can outperform actual techniques. However, it should be noticed that when the topology of the search space for a given problem is well-known and its gradient is appropriate for Hill-Climbing or SA, these techniques are more efficient than the END model regarding execution time. Indeed, the approach of the END model is a statistical one which progressively

takes into account details of the landscape; no gradient information is used.

The aim of this paper was to present in details the parameters of the model in order to understand their importance in the efficiency. This is the reason why we focussed only on some elementary operators for selection and solution generating. The model described in this paper can be easily enhanced with some new features like:

- Allowing the use of some heuristics for solution generating to reduce the number of potential extensions at each node of the tree of solutions.
- Managing a local memory for each organism that would memorize its "past" and would allow learning.
- Each organisms looks for a local optimum before selection rounds. When possible, this technique allows a faster convergence.

The END model is also highly parallelizable and it is easy to imageine a parallel machine, using a 2-D mesh architecture, with a very large number of processors, each one simulating an organism. Since the complexity of the problems the model is able to handle increases with the size of the population, such an implementation would be fantastic!

Finally, the very encouraging results let us think about applying this model on even more challenging problems like:

- multi-player games,
- problem solving,
- mechanical discovery of heuristics or theorems,

for which each organism would be an elementary and naive game player or problem solver. Then, we could expect the emergence of some "high-level strategies"...

6 Acknowledgments

I would like to thank Marty Cohn, Patrick Tufts, Ira Gessel, Gary Drescher, Jacques Cohen and Jordan Pollack for their advice and discussions. The idea to use the ramp problem to study the END model efficiency comes from Gary. Patrick helped me a lot by exchanging ideas and references with me. I would like also to thank Roger Gallier who challenged me one day with the Solitaire problem. I owe many sleepless nights to him and this problem. Thanks also to the NSF whose grant allowed the Brandeis Computer Science to buy a Maspar machine. Without this machine, all this work wouldn't have been possible. Finally, I want to thank my wife, Anne, for the moral support she provided me while I was working on this project and her constant curiosity.

References

[1] Richard Belew and Thomas Kammeyer, *"Evolving Aesthetic Sorting Networks using Developmental Grammars"*. In *Proceedings of the Fifth International Conference of Genetic Algorithms.*

[2] Roberto Bisiani, *"Beam Search"*. In *Encyclopedia of Artificial Intelligence*, Vol. 2, Second Edition, John Wiley & Sons, 1992.

[3] Ronald L. Graham, Donald E. Knuth and Oren Patashnik, *Concrete Mathematics, a Foundation for Computer Science.* Second edition, Addisson-Wesley, 1989.

[4] Milton W. Green, *"Some Improvements in Nonadaptive Sorting Algorithms"*. Stanford Research Institute. Menlo Park, California.

[5] W. Daniel Hillis, *"Co-Evolving Parasites Improve Simulated Evolution as an Optimization Procedure"*. In *Artificial Life II*, Langton, et al, Eds. Addison Wesley, 1992, pp. 313-324.

[6] Kim Kinnear, *"Generality and Difficulty in GP: Evolving a Sort"*. In *Proceedings of the Fifth International Conference on Genetic Algorithms*, S. Forrest, Morgan Kaufmann Publishers, 1993.

[7] Donald E. Knuth, *The Art of Computer Programming: Volume 3 - Sorting and Searching.* Addison Wesley, 1973.

[8] Steven Levy, *Artificial Life: the Quest for a New Creation.*Pantheon Books, 1992.

[9] Ian Parberry, *"A Computer-Assisted Optimal Depth Lower Bound for Nine-Input Sorting Networks"*. In *Mathematical Systems Theory*, No 24, 1991, pp. 101-116.

[10] Conor Ryan, *"Pygmies and Civil Servants"*. In *Advances in Genetic Programming*, Kim Kinnear, Ed. MIT Press, 1994.

[11] Patrick Tufts and Hugues Juille, *"Evolving Non-Deterministic Algorithms for Efficient Sorting Networks"*. Unpublished.