TOP-DOWN FUNCTIONAL DECOMPOSITION IN DIGITAL PERCEPTRONS

Gabriela Adler
Jacques J. Vidal

March 1987
CSD-870014

TOP-DOWN FUNCTIONAL DECOMPOSITION IN DIGITAL PERCEPTRONS:

Gabriela Adler & Jacques J. Vidal

Computer Science Department
University of California, Los Angeles

July 86

TABLE OF CONTENTS

1. INTRODUCTION

Boolean networks studied at UCLA, also known as "UCLA perceptron systems" [Vers86].

The main motivation behind the research is to design perceptron systems that not only possess attractive functional features, but offer some advantages from hardware (and especially VLSI) viewpoint as well.
[Moor85, Vida85].

Previous work at UCLA has shown that a "flat-weave" network of 2-input boolean nodes presented distinct advantages for implementation. Each node is a DPLM, functionally complete that it can be "set" to any of the 16 possible boolean functions of its two inputs. (fig. 1.a.).
The use of DPLMs set the UCLA networks apart from other adjustable logic networks proposed by other authors [Aleks83] [Arms79]. Aleksander, for example, used a RAM (Random Access Memory) as a building block for his system. A RAM has n address lines and 2^n bits of storage, and therefore is a multiple-input complete node. Armstrong, on the other hand, uses 2-input digital nodes that can only implement four out of sixteen possible functions - namely, the so-called "non-constant increasing functions". Thus, these nodes are incomplete.

As stated before, the ULM is a functionally complete boolean node. Aside from the two inputs, it has a 4-bit register, which stores the responses of the node to each of the four possible binary input patterns. Naturally, n-input ULMs can be used with a 2^n -bit control register. is necessary. Such a node, however, would result in a coarser-grain distributed memory/processing network, an indesirable feature from the point of views of implementation ease, speed and testability.

Given the structure of the node the most striking feature of the network as a whole is its regularity. It is a homogeneous network, i.e., all nodes are physically and functionally identical and the network has a regular, highly interconnected topology. Furthermore the width of the network varies linearly with depth, rather than logarithmically, as with binary trees. An example of a 3-input DPLM network is given in fig. 1.b. This particular network is also lgically complete. (It has been shown (Sal.86), that there exist a complete flat-weave network of this type for any number of inputs.)

Another important characteristic of DPLM networks is a multi-layered, hierarchical architecture. The advantages of this type of structure for vision systems are clearly stated in [Uhr80]. As Uhr pointed out, the serial building of successive layers of "transforms" allows the system to take successively more global looks at a given scene, and to develop continually more abstract representations of the scene. Thus, layering is essential for variable resolution, allowing the system to zoom in and out, looking for pertinent levels of detail.

If depth and width are interpreted as respectively a serial and a parallel dimension of the network, it can be seen to combines the space efficiency of a purely serial systems with the time-efficiency of parallel systems. This is a major difference between UCLM and single-layer nets, which are strictly parallel, such as the system proposed by Aleksander[Aleks70].

From a functional viewpoint, the flat-weave UCLM network is a general-purpose system, since it is capable in principle of realizing any and all boolean functions of its inputs. Furthermore, it is redundant, i.e., each global function may have multiple logical implementations on the same physical network. On the average, a function can be implemented on the three-input network of figure 1-b in 2^{16} different ways [Vers86], a feature that gives such networks the potential for fault-recovery.

The multi-layered flat-weave structure makes the network hard to analyze. When a change occurs at the output , it is not a trivial matter to determine which nodes in which layers are responsible for the change. This is widely known as the "credit assignment problem", and it is the central problem one has to deal to "program" the network.

The fan-out interconnections constitute another source of difficulties. As seen in fig. 1.b, the outputs of certain nodes are inputs to more than one node above them in the hierarchy. Again, this feature makes the analysis of the network difficult, since functional changes occurring in fan-out nodes have a more global effect over the network.

In the balance of this paper, we will deal specifically with the issues of layeredness and fan-out in the context of goal decomposition algorithms applied to the flat-weave and other DPLM networks.

DECOMPOSITION IN DIGITAL PERCEPTRON SYSTEMS

This paper deals with programming strategies for digital perceptron systems in general, using the flat-weave DPLM network to illustrate the procedures .Given a desired global Boolean function to attain, what function must the constituent elements perform in order for the entire network to realize the global function? Or, in simple terms, it is the programming problem : "given a function,implement it on the network".

Note that the decomposition problem referred to above is an *analysis* problem, as opposed to the classic decomposition of boolean functions [Frie75], which is a *synthesis* type of problem. Here, we start with a fixed combinational architecture which must be then functionally configured to handle a particular boolean function. By contrast, n the classical case, one constructs a special architecture that implements a given boolean function.

In the following subsections, we will define the notion of decomposition, after which two possible decomposition strategies will be discussed (*bottom-up* and *top-down*). The latter is a new contribution to the problem and the object of this report.

2.1. Network Decomposition

The decomposition problem for a perceptron structure can be defined as follows:

Given: A perceptron network and a target boolean function ("goal function");

Find: An assignment of functions for each node in the network, such that the overall network implements the given function.

More formally, if we denote by l the number of nodes in the perceptron system, by $F(x_1,...,x_n)$ the boolean function implemented by the system, and by $Ø = \{ f_1 ,..., f_p \}$ the functional set of each such node (assuming all nodes identical), then our definition becomes (see fig. 5.1)[Vers86]:

Given: A perceptron structure P and a goal function $G(x_1,...., x_n)$;

Find: Node functions $\{g_j\}_{1\leq j\leq l}$, such that:

1) $g_j \varepsilon \phi$, for any j, where $1\leq j\leq l$ and

2) $F(x_1,...., x_n) = G(x_1,...., x_n)$.

The set $\beta = \{ (j,g_j) \mid 1\leq j\leq l, g_j \varepsilon \phi \}$ is called a functional *assignment*.

In the algorithms to be presented, for the sake of simplicity, we will make a few assumptions regarding the network we'll be dealing with. Namely:

1. All nodes are identical;
2. Each node has two inputs;
3. Each node is functionally complete (i.e., it can implement all 16 boolean functions of its inputs).

The above represent just a set of simplifying assumptions, and not prerequisites for the decomposition algorithms tdiscussed in this paper.

In the remainder of this paper, we will discuss two decomposition strategies for perceptron systems, respectively a bottom-up and a top-down method.

2.2. Bottom-Up Decomposition

The bottom-up decomposition approach consists of assigning responsibilities to each node of the network in a sequential fashion, starting with the bottom layer first. Essentially, the only requirement for such a strategy is that the child nodes are assigned functions *before* their parent node. The order in which the nodes can be processed is not unique - however, which order is used is not an issue, as long as it complies with the above rule. Thus, for the example of flat-weave network shown in fig. 1.b, two possible orderings are (4,5,2,6,3,1) and (4,5,6,2,3,1). The latter is a "layer-by-layer" scheme, and is purely sequential; the former, on the other hand, is more parallel in nature since, for example, after an assignment has been made for nodes 4 and 5, both 6 and 2 could be worked on independently. Therefore, the order in which the nodes are considered may have an impact on the overall efficiency of the algorithm.

A very general and effective bottom-up decomposition method was proposed by Rik Verstraete in [Vers86]. Based on *decomposition charts* [Frie75], this method can be applied to any perceptron structures and any goal function. For the case when the system is a *disjunctive binary tree*, the method is based on the theory of decomposition of boolean functions detailed in [Curt61, Curt63, High73, Frie75]. This is a single-path, straightforward approach.

Things become more complicated when the system is a *non-disjunctive binary tree*. If the perceptron has no internal fan-out (i.e., the output of each node is connected to only one input of a node above it in the hierarchy), then the non-disjunctive tree can be transformed into a disjunctive tree. This would be achieved by introducing *virtual variables* , as shown in fig. 3. These virtual variables are represented by "don't cares" in the truth table (or "decomposition chart") of the goal function. Since the "don't care" entries may receive dual values (i.e., 0 or 1), any instance of the decomposition process may result in a multitude of local assignments, not all of them leading to a valid solution. We are thus dealing with a trial-and-error, search-oriented strategy.

The presence of fan-out nodes in a network increases the amount of search once again, since each internal fan-out is a source of additional "don't care" entries in the function's truth table. For details on how the bottom-up heuristic was implemented on general perceptron networks, we refer to [Vers86].

To summarize, then, the bottom-up approach to decomposition is an inherently sequential strategy that has been successfuly applied to disjunctive and non-disjunctive binary tree networks, as well as general perceptron networks. Its applicability to a wide range of network structures lies, in fact, in its own sequential nature. Indeed, the method outlined above treats the nodes one at a time, each assignment being a local decision, independent of the structure of the remainder of the network.

Therefore, we are faced with a clear trade-off between the *generality* and the *performance* of this algorithm: on one hand, the locality of processing renders a scheme that is very general, i.e., applicable to any perceptron structure; on the other hand, this limited "look-ahead" approach results in increased backtracking, which has a negative impact on performance. This result is not surprising, however, since non-determinism is known to induce backtracking, thus trading generality for efficiency.

We have thus identified two main sources of inefficiency in the bottom-up strategy, namely, *sequentiality* and *search* . In the next subsection, we will present a decomposition strategy that deals effectively with the former drawback of the bottom-up philosophy; and a special-purpose solution that uses a "look-ahead" heuristic in order to prune down the search space, thus limiting the amount of backtracking necessary.

2.3. Top-Down Decomposition

A top-down decomposition procedure is one starting with the top node of the network and working its way downward, until each node has been assigned a "responsibility" (or a function), in such a way that the network as a whole accomplishes the desired goal. Just like in the case of the bottom-up strategy, the order in which these assignments take place is not essential, the only requirement being that the parent of a node is assigned its function before the node itself.

2.3.1. Rationale

There are many differences of detail between the two decomposition strategies (i.e., bottom-up and top-down), which will become apparent later on in this section. There is, however, one major conceptual difference: In the bottom-up approach discussed in 2.2, the global goal is presented to one of the <u>bottom nodes</u> of the network. Thus, nodes that are higher up in the hierarchy have to attain a goal that is dependent upon the assignments of nodes lower in the hierarchy. Hence, the term of *residual goal* . At each decision point, after a *selection* of a boolean function has been made for a node, a *reduction* test is performed to ensure that the assignment just selected is consistent with the global goal of the network. If the answer is positive, then a residual goal is generated that becomes the global goal of the <u>reduced</u> network. These residual goals get gradually "simpler" as the algorithm proceeds through the network in a bottom-up fashion. The net result of this decomposition process is that the top node becomes responsible for only a small portion of the entire "task", while the bottom nodes accomplish most of the work.

The situation is reversed in the top-down approach. There, the global function to be performed by the network is presented to the <u>top node</u>, which then decomposes it into subfunctions and delegates them to the subordinate nodes. Thus, the top node is the only one that "sees" the global goal; it may be, however, unaware of certain changes occurring in the deeper layers of the hierarchy that don't affect it directly. In addition, once a subnetwork has been designated a goal, any subsequent assignments internal to it are irrelevant to the operation of adjacent subnetworks. As a result of these two features [Simo73], known as "loose vertical coupling" and "loose horizontal coupling", the top-down approach shows a high degree of functional independence, and thus appears to be suitable for a parallel implementation of the network control.

Therefore, a major advantage of the top-down principle is that it lends itself to paralell network programming. Another intuitive argument in favor of this approach is that most social structures that are organized hierarchically, operate in a top-down fashion. The goals for an organization are usually only known to the leader of the organization; his/her responsibility is then to decompose and propagate them downwards through the organization, with the goals becoming more focused in the lower levels of the hierarchy. It is this analogy that determined us to investigate the top-down approach, as a natural way in which a hierarchical structure like the perceptron could achieve its self-organization.

In the remainder of this section, we will look at specifically how the top-down strategy can be implemented in digital perceptron systems, starting with simpler DPLM structures (such as disjunctive binary trees), and ending with general DPLM networks, such as the flat-weave structure.

2.3.2. Disjunctive Networks

a. The 4-Input Network

Let us start by considering the disjunctive binary tree of fig. 4.a and let's try to project onto it the function $G(x_1,x_2,x_3,x_4) = x_1\underline{x}_2x_3 + x_1\underline{x}_2x_4 + \underline{x}_1x_2x_3 + \underline{x}_1x_2x_4$. The decomposition chart of this function, with respect to $A_1 = \{x_1,x_2\}$ and $A_2 = \{x_3,x_4\}$, as shown in fig. 4.b, has two distinct types of columns and two distinct types of rows. This implies that the decomposition chart with respect to A_2 and A_1 (fig. 4.c) has two distinct types of columns, as well. Therefore [Fried75, theorem 2.3], there exists a multiple disjunctive decomposition of the function $G(x_1,x_2,x_3,x_4)$ into two functions $g_1(x_1,x_2)$ and $g_2(x_3,x_4)$, i.e., $G(x_1,x_2,x_3,x_4) = G'(g_1(x_1,x_2),g_2(x_3,x_4))$, where:

$g_1(x_1,x_2) = x_1\underline{x}_2 + \underline{x}_1x_2 = x_1$ <exor> x_2 and

$g_2(x_3,x_4) = x_3 + x_4$

If we now denote $z_1 = g_1(x_1,x_2)$ and $z_2 = g_2(x_3,x_4)$, and substitute z_1 and z_2 in G, then we obtain:

$G(x_1,x_2,x_3,x_4) = G'(z_1,z_2) = z_1 \cdot z_2$

Hence, the functions assigned to nodes 2,3, and 1 in fig. 4.a are, respectively, EXOR, OR and AND. They are the result of two *selection* operations and one *substitution* operation (in [Vers86], this operation is referred to as *reduction*). The test expressed by Friedman's aforementioned theorem we will call a *decomposition test* . Selection led to the assignments of EXOR and OR functions to the "child" nodes (2 and 3), while substitution resulted in the assignment of the AND function to the "parent" node, in this case node 1. Since in a disjunctive tree these assignments are unique (up to a negation of the function), the two selection operations can proceed in parallel, independently of each other. If either selection is not possible, then the entire decomposition fails. That is because of the fact that, if a functional decomposition assignment exists for a disjunctive network, then it is unique - hence, no search is necessary.

b. The n-Input Network

Let us now consider the more general case of a disjunctive binary tree with *n* inputs, where $n>4$ (fig. 5).The same line of reasoning applies. Since the entire network (N) is disjunctive, it follows that the left and right subnetworks (i.e., N_1 and N_2) are mutually disjunctive. As a result, we can treat these subnetworks, for now, as child nodes of the top node of N. So we have reduced the n-input, multi-layered case to the former case of a bi-layered binary tree, with the only difference that each of the bottom "nodes" now have more than two inputs. Using the scheme described above, we can decompose function G into subfunctions $g_1(x_1,\ldots,x_i)$ and $g_2(x_{i+1},\ldots,x_n)$ (where $x_1,\ldots,x_i$ are inputs to N_1, and $x_{i+1},\ldots,x_n$ are inputs to N_2), and by substitution determine the boolean function of the top node. Once it has been determined that g_1 and g_2 exist, and the corresponding assignment has been made to node 1, the same algorithm can be invoked recursively for subnetworks N_1 and N_2. Since no coordination is necessary between N_1 and N_2, the two decompositions can proceed concurrently, until either a complete assignment has been found, or a "dead-end" occurred. A dead-end can happen when, at some node in the network, the decomposition test can't be satisfied, and therefore selection is not possible. The schematics of this recursive algorithm are shown below (S_c is the "cautious selection" function implemented in [Vers86]):

```
procedure DISJ (G, N);
begin if top (N) = leafnode
      then assign (G, top(N))
      else begin $N_1$<-leftchild (top(N),N);
                 $N_2$<-rightchild (top(N),N);
                 $g_1$<-$S_c(\emptyset_1$, G, top($N_1$), $N_1$);
                 $g_2$<-$S_c(\emptyset_2$, G, top($N_2$), $N_2$);
                 if ($g_1 \neq 0$ and $g_2 \neq 0$)
                 then begin ø<-substitute ($\{g_1,g_2\}$, G);
                            assign (ø, top(N));
                            DISJ ($g_1$,$N_1$);
                            DISJ ($g_2$,$N_2$)
                      end
```

```
        end
    end;
```

In conclusion, the algorithm for top-down functional decomposition onto disjunctive networks (DNs) is deterministic and concurrent, and therefore easily implementable on a parallel architecture. The best analogy for it would be that of a global process that, when certain constraints are met, will spawn two or more independent subprocesses. When these subprocesses take over, the "parent" process dies, thus becoming unavailable for generating new processes whenever the current ones turn out to be "sterile" (by that, we mean processes incapable of generating new ones). The only communication links necessary are those between a parent process (node) and its immediate descendants. No communication is needed between sibling processes (nodes), or between nodes that are several layers apart.

Which brings us to the implementation issue. It should be apparent by now, that the algorithm described lends itself to an implementation with *distributed control*. In other words, each node of the perceptron in fig. 5 could have a small controller attached, just powerful enough to undertake the amount of processing contained by each recursive step of our algorithm. These controllers would receive the same goal function from their parent node, run autonomously towards achieving this goal, and report back to the parent node their results. Therefore, most of the processing is local. There are situations, however, when a central controller, although not necessary, would be beneficial. For example, termination signals, such as SUCCESS or FAILURE would be more efficiently handled by the central controller. A "success" signal would inform the central controller that the decomposition phase is over and the network is ready to process data. When receiving a "failure" signal", the central controller would emit an "interrupt" signal to the network, thereby bringing any ongoing decomposition subprocesses to a halt.

In the next subsection, we intend to investigate the same top-down decomposition approach as it applies to non-disjunctive networks (NDNs). Since the decomposition problem has a different degree of complexity for non-disjunctive networks without internal fan-out, versus those with internal fan-out, we will treat these two cases separately.

2.3.3 Simple Non-Disjunctive Networks

Simple non-disjunctive networks (SNDNs) are non-disjunctive networks with fan-out in their inputs, but not in their internal structure. An example of a SNDN is given in fig. 6.

The algorithm for top-down decomposition in SNDNs is an extension of the algorithm DISJ for disjunctive networks. Given a SNDN, it can be transformed into a DN by introducing *virtual* *variables* in the truth table of the global function. We'll refer to the resulting network as the *virtual network* of the SNDN it originated from. This mechanism, of reducing a SNDN to a DN, is identical to the one used for bottom-up decompositions, as outlined in [Vers86]. We'll refer to this mechanism here as *relaxation* , since its purpose is to "relax" a constraint, namely that of several nodes having common inputs.

a. The 3-Input Network

Consider, for example, the 3-input network in fig. 6. By introducing the virtual variable x_2^v, the network becomes disjunctive in four variables. As a result, the size of the decomposition chart associated with the new network will have twice the size of the original truth table. However, since $x_2^v=x_2$, half of the new truth table contains *don't care* entries.

Let us try, for example, to decompose onto the network S the function $G(x_1,x_2,x_3) = x_1x_2 + x_2x_3$, whose truth table (or decomposition map or chart) is shown in fig. 6. In the virtual network S^v, the two subnetworks of the top node contain, respectively, the inputs $\{x_1,x_2\}$ and $\{x_2^v,x_3\}$; therefore, let's consider the decomposition chart with respect to $A_1=\{x_1,x_2\}$ and $A_2=\{x_2^v,x_3\}$ (fig. 6). Since S^v is a disjunctive network, we can use the decomposition test defined in the previous subsection. Note, however, that due to the presence of "don't cares", a successful test may generate more than one solution. In other words, selection is non-deterministic, and as a result, the two operations of selecting $g_1(x_1,x_2)$ and $g_2(x_2^v,x_3)$, are no longer independent of each other: a substitution has to take place between them. In effect, the selection of g_2 operates on a residual truth table (G^r), which is the result of substituting g_1 into the virtual truth table (G^v). The schematics of the algorithm applied to the network S are illustrated in fig. 5.6. Note that a successful decomposition will always result in a "collapsed" truth table (2*2 in a binary tree), which represents the functional assignment for the top node. The general algorithm is listed below:

```
procedure **NONDISJ** (G, N);
begin if top (N) = leafnode
      then assign (G, top(N))
      else begin $N_1$<-leftchild (top(N),N);
                 $N_2$<-rightchild (top(N),N);
                 $G^v$<-virtual (G, top(N), N);
```

```
                $g_1$<-$S_c$($\varnothing_1$, $G^V$, top($N_1$), $N_1$);
                if $g_1 \neq 0$
                then begin $\varnothing$<-substitute ($g_1$, $G^V$);
                           $g_2$<-$S_c$($\varnothing_2$,, top($N_2$), $N_2$);
                           if $g_2 \neq 0$
                           then begin $\mu$<-substitute ($g_2$, $\varnothing$);
                                      assign ($\mu$, top(N));
                                      NONDISJ ($g_1$,$N_1$);
                                      NONDISJ ($g_2$,$N_2$)
                                end
                     end
          end
   end;
```

As demonstrated by the example, the top-down strategy for SNDNs is one based on search, since there may be more than one way to "collapse" the virtual truth table into a 2 column-2 row truth table corresponding to the top node. The following question arises at this point: is the decomposition test derived from Friedman's theorem a necessary and sufficient condition for decomposability? Clearly, the condition expressed by the test is *necessary* : if the decomposition chart of a given function G with respect to $\{A_1,A_2\}$ (or $\{A_2,A_1\}$, for that matter) has more than two mutually incompatible columns or rows, then no multiple disjunctive decompositions of G exist [Frie75, theorem 2.3].

However, as the counterexample in fig. 5.7 demonstrates, the test is *not sufficient* . Let's first make the following observation. The theoretical method used to find a non-disjunctive decomposition only ensures that, if the test is satisfied, then G is decomposable into two functions $g_1(A_1)$ and $g_2(A_2)$; it does not, however, guarantee g_1 and g_2 to be decomposable on the remaining subnetworks. Thus, the test provides a criterion for synthesizing a network that will implement a given function. Our goal, as stated previously, is that of adapting to a rigid architecture, rather than finding an architecture that conforms to given functional constraints. For our purpose then, the test is necessary, but not sufficient. As a result, backtracking has to take place whenever a "dead-end" path is recognized and must be "undone".

To conclude then, in order to increase the efficiency of the top-down algorithm, we need a more powerful test than the current decomposition test. Ideally, such a test should consist of a necessary <u>and</u> sufficient condition, whereby <u>all</u> possible redundant assignments could be generated after successive backtrackings. Such a test, however, is very difficult to construct. In the next subsection, we'll present a method of strengthening our current test, by complementing it with a another, sufficient although not necessary test.

b. An Informed Search Approach

As it has already been pointed out, the test used as a basis for decomposition in the previous section is a local, rather limited one, incurring backtracking whenever the solution being considered turns out to be invalid.

What is needed then, is a more "informed" approach, whereby each selected function would be subjected to a *feasibility* test. While the decomposition test ensures that *a* network that achieves the new function (subgoal) can be found, the feasibility test checks to see if *the* particular network to which this function was designated can, in fact, implement it. Thus, this is a "look-ahead" kind of test, which assumes some degree of knowledge about the structure of the remaining network - hence, the term *informed* heuristic.

In what will follow, an example of how such a heuristic works will be given. The example is a 3-input complete SNDN, which resulted from a 3-input GPP by replicating its fan-out node (fig. 9.a). The feasibility test is based on a set of rules, derived empirically and then proved to be valid. They are given here without proof:

Let T be the truth table associated with a given global function G. Then:

1. If T has only one type of column (i.e., all columns are identical), then G is decomposable onto exactly one layer (i.e., the bottom layer);

2. If t has two distinct types of columns, then G is decomposable onto at most two (i.e., 1 or 2) layers;

3. If T has three distinct types of columns, then G is decomposable onto at most three (i.e., 2 or 3) layers;

4. If T has four distinct types of columns, then G is decomposable onto exactly two layers.

As an example, the truth table in fig. 9.b would necessitate the entire network to implement it (since it has 3 mutually incompatible columns), while the truth table in fig. 9.c can be implemented on a subnetwork of the top node (having only 2 mutually incompatible columns).

Note that the above set of rules applies to the example SNDN only. Depending on its structure, each network would have a different set of rules as part of the feasibility test. An interesting issue for future research is that of finding a general set of such rules, applicable to any perceptron structure.

In our informed search algorithm, we'll use a slightly modified version of these rules. The revised rules are represented in the diagram of fig. 10. Clearly, they express sufficient (though not necessary) conditions for decomposability; thus, we obtain a *strict* subset of the set of all possible network configurations corresponding to a given global function. In this sense, the informed search algorithm is similar to the adventurous algorithm for bottom-up decomposition [Vers86]: by neglecting some correct assignments, it reduces the size of the search tree. The program for the informed search algorithm is listed below:

```
procedure **INFORMED** (G,N);
begin if top(N) = leafnode
      then assign (G, top(N))
      else begin $N_1$<-leftchild (top(N), N);
                 $N_2$<-rightchild (top(N), N);
                 $G^v$<-virtual (G, top(N), N);
                 $g_1$<-$S_c$ ($Ø_1$, $G^v$, top($N_1$), $N_1$);
                 test_if_feasible ($g_1$, $N_1$, feasible1);
                 if feasible1
                 then begin ø<-substitute ($g_1$, $G^v$);
                            $g_2$<-$S_c$ ($Ø_2$, ø, top($N_2$), $N_2$);
                            test_if_feasible ($g_2$, $N_2$, feasible2);
                            if feasible2
                            then begin $\mu$<-substitute ($g_2$,ø);
                                       assign ($\mu$, top(N));
                                       INFORMED ($g_1$, $N_1$);
                                       INFORMED ($g_2$, $N_2$)
                                 end
                      end
           end
end;
```

c. An "Optimized" Heuristic

Let's again consider the 3-input complete SNDN in fig. 9.a. We can now effectively use the previous rules (1-4) towards an *optimized* heuristic. The optimization not only results in time savings, but also yields a *minimal* solution. A solution is said to be minimal if, given a global function and a perceptron network, the number of nodes that are assigned non-trivial functions via decomposition, is minimal (by "non-trivial" we mean those boolean functions other than LEFT and RIGHT - also known as *transparent* functions). The merit of a minimal configuration is that it makes the network less prone to failure: since fewer nodes are operational to begin with, the probability of node failure is smaller (if by "failure" we mean the degradation of a node into a "transparent" one).

How do we accomplish a minimal configuration? The idea is to "push" the global goal function as far into a corner of the network as possible. One way to accomplish this, is to simply reverse the order between the feasibility test and the selection process in the informed search algorithm outlined above. As usual, the algorithm will start at the top node of the network, with a given goal function G. At the begining of each decomposition step, the function's "place" in the network is established by using rules 1-4, on which the feasibility test is based. For example, a function whose truth table has two mutually incompatible sets of columns will be "routed" directly to nodes 2 and 3, which will then simultaneously attempt the decomposition. Nodes 2 and 3 may, or may not be able to both assume the new function (however, at least one of them will). If they do, then we get duplicate implementations of the same function, within the same network - hence, *redundancy*. The algorithm is recursive, so that each new offspring of a function is "pushed" as far downward as possible, thus taking full advantage of the redundancy of the network by realizing multiple internal "versions" of a given function. The optimized algorithm is outlined below:

```
procedure OPTIMIZED (G, N);
begin if top (N) = leafnode
      then assign (G, top (N))
      else begin feasible<-true;
                 $N_1$<-N;
                 while feasible do
                 begin N<-$N_1$;
                       $N_1$<-leftchild ( top (N), N);
                       test_if_feasible (G, $N_1$, feasible);
                 end;
                 $N_1$<-rightchild (top (N), N);
                 $G^v$<-virtual (G, top(N), N);
                 $g_1$<-$S_c$ ($Ø_1$, $G^v$, top($N_1$), $N_1$);
                 ø<-substitute ($g_1$, $G^v$);
                 $g_2$<-$S_c$ ($Ø_2$, ø, top($N_2$), $N_2$);
```

```
                $\mu$<-substitute ($g_2$, ø);
                assign ($\mu$, top(N));
                OPTIMIZED ($g_1$, $N_1$);
                OPTIMIZED ($g_2$, $N_2$)
            end
        end;
```

d. The n-Input Network

The decomposition strategy outlined in 2.3.3 (a) can be applied recursively to a generic n-input (n>3) SNDN. The idea is the same as in the case of an n-input disjunctive network. Namely, within each recursive step, we can view the respective network as a 3-node network, in which the bottom nodes are represented by the descendant subnetworks of the top node (fig. 11).

The algorithm can be implemented either sequentially, or in parallel, by using decentralized control. The relaxation, selection and substitution procedures would be executed by each node controller sequentially; however, once a node has generated subgoals for its successor networks, the top nodes of these subnetworks can work concurrently towards achieving them. When backtracking occurs, only the selection and substitution steps need to be undone, since there is only one way to "relax" a given non-disjunctive network into a disjunctive one. In order to execute either the informed or the optimized heuristic, the controllers must have extra processing power to be able to perform the feasibility test.

Will the top-down approach discussed so far work on a more complex, highly interconnected network? We'll attempt to answer this question in the following subsection.

5.3.4. Complex Non-Disjunctive Networks

So far we have seen how the top-down methodology works on disjunctive and non-disjunctive networks without internal fan-out (SNDNs). We started by constructing an algorithm for the disjunctive case, and then used it as a "building block" for the simple non-disjunctive case. In the process of doing that, we made use of relaxation, a general technique whereby a constraint is temporarily suspended to facilitate processing; once a solution set of the '"relaxed" (or virtual) problem is found, the constraint is reconsidered in order to determine the subset of solutions to the initial (actual) problem. In our case, the constraint consisted of fan-out inputs, and to relax this constraint we had to introduce virtual variables. We say that we have *replicated* the fan-out inputs.

The networks considered in this subsection have internal fan-out, and may or may not have input fan-out. An example of a complex non-disjunctive network (CNDN)-or a *graph network* -is the flat-weave network discussed previously (fig. 1.b), which has both internal and external fan-out. For this class of networks, we can use the same approach as for SNDNs. Namely, since any CNDN can be "relaxed" into a SNDN by replicating the fan-out nodes, we can then apply any of the algorithms for SNDNs to the relaxed network. Thus, in this case the constraint consists of *fan-out nodes* , and to relax it we are introducing *virtual nodes* (fig. 9.a).

So far, then, the strategy for CNDNs closely parallels that of SNDNs. There is, however, one major difference: having to "collapse" the virtual network back onto the real physical structure. In other words, both the fan-out node (#5 in GPP's case), and its clone (#5^V) must have the same functional assignment.

In summary, the strategy for top-down decomposition in a CNDN consists of the following three steps:

1. *Relaxation* of the initial CNDN into a SNDN;
2. *Decomposition* of the global function onto the SNDN;
3. *Satisfaction* of the internal fan-out constraint.

The algorithm is, again, recursive with each recursive phase being comprised of the above three processing steps. In what follows, we'll illustrate the algorithm as it applies to a 3-input GPP, with emphasis on the constraint satisfaction process.

a. The 3-Input Network

Let us again consider the 3-input network of fig. 1.b. This "tightly-knit" structure does not readily lend itself to analysis, for the following reason: when trying to decompose a function onto it in a top-down fashion, functional assignments impressed from the top node downward are countered by an upward flow of consequential changes. In other words, the fan-out node (#5) introduces additional functional interdependencies between nodes 2 and 3, that cannot be predicted at the time these nodes get resolved. The two decomposition subproblems cannot work independently from one another, because they would typically try to assign different functions to their common nodes (in this case, node 5). A solution to one subproblem constrains the solution to the other, and thus a large amount of coordination is necessary.

Therefore, a first step towards decomposition would be to relax the fan-out constraint, by replicating node 5; the result would be a SNDN like the one in fig. 9.a. We can now apply any of the algorithms discussed in 2.3.3 to the relaxed (virtual) network T^V. However, none of these algorithms guarantees a solution with identical assignments to nodes 5 and 5^V. Hence, we now have to reconsider this constraint, as follows:

Let us denote by f_5 and f_5^V, respectively, the assignments of nodes 5 and 5^V. There are two possibilities:

1. $f_5 = f_5^V$.

In this case, the decomposition already satisfies the constraint, and therefore it is a valid solution.

2. $f_5 \neq f_5^V$.

This type of decomposition does not satisfy the constraint, therefore the algorithm has to backtrack for an alternate solution, as follows:

a. Assign f_5 to node 5^V;

b. Apply decomposition to the right subnetwork of node 1;

c. If decomposition was successful, then a solution has been found; EXIT.

d. If decomposition failed, then repeat steps (a)-(c) for function f_5^V and node 5;

e. If decomposition failed again, then backtrack to the second layer and generate new subgoals for nodes 5 and 5^V;

f. Go to step (a).

We can speed up this last phase of each decomposition subproblem by taking advantage of parallelism. That is, f_5 and f_5^V can be generated, substituted in the sibling subnetwork, and then subjected to the decomposition test, all in a concurrent manner. The average time needed for satisfying the constraint would thus be reduced by a factor of two.

b. The n-Input Network

For an n-input network like the 5-input graph network of fig. 12 which has multiple fan-out nodes, the approach becomes orders of magnitude harder.

Each node inside the highlighted area of the network, being a fan-out node, is a contention element. As we advance with the decomposition process towards the bottom of the network, the coordination scheme between these nodes becomes increasingly complex. The reason being, that the deeper a layer is within the network, the more fan-out nodes it has. As a result, any change in any of these nodes would have a "ripple" effect throughout the entire layer, causing the sibling nodes to adjust accordingly. Additional research is needed in this direction.

3. CONCLUSION

This paper has explored an important issue related to the adaptation and control of perceptron systems, namely functional decomposition in boolean networks. A new approach has been proposed. The general algorithm presented, proceeds in a top-down fashion and is complementary to the bottom-up approach implemented by Rik Verstraete [Vers86]. As was the case with the bottom-up approach, the fan-out connections (external and internal) were identified to be the main source of difficulty in developing an efficient strategy. As a result, the decomposition algorithm was very straightforward on disjunctive networks but became more complicated when applied to non-disjunctive networks with external fan-out. Search ibecomes necessary because of the "don't care" entries in the virtual decomposition chart. Finally, approach was extremelly inefficient in the case of non-disjunctive networks with both external and internal fan-out, due to the large amount to coordination required by the presence of overlapping nodes.

Thus, because fan-out connections introduce redundancy in the network, the more such connections a structure will have, the more search will be required by the decomposition algorithm. Another "side-effect" of fan-out connections, which again has a negative impact on the performance of decomposition algorithms, is the constraints they bring along with them. As has been shown, each fan-out node represents a contention point for the nodes directly above it, and this conflict must always be resolved before the decomposition process can continue. Thus, the more "contention" (or overlapping) nodes exist in a perceptron system, the more sequential will the top-down decomposition process become, due to the interdependencies they introduce. We demonstrated this point by showing in section 2.3.4 how the decomposition approach proposed for SNDNs can be applied within reasonable performance limits, to a CNDN with only one overlapping node; however, when trying to apply this same approach to a general CNDN (i.e., with multiple overlapping nodes), its inefficiency became immediately apparent.

What are, then, the characteristics and potential advantages of the top-down strategy?

Firstl, the algorithms proposed here have exploited the advantage of decomposing a function onto a network with "large" nodes - meaning nodes with a large number of inputs. It has been argued [Vers86], that whenever the node or subnetwork being assigned responsibility is only a little smaller in size than the whole network, the problem is simpler than when the node is much smaller. We have taken advantage of this feature by recasting the problem onto a 3-node network equivalent to the initial network (subnetwork), at each step of the decomposition process.

Second, the algorithms presented in section 2.3 mirror, to some degree, the parallel structure inherent to perceptron systems. Sequential processing is still present ifor non-disjunctive networks, but it is limited to resolving local conflicts, while the decomposition as a whole remains largely parallel. As a direct result of parallelism, distributed control becomes possible; limited communication would have to take place between adjacent local controllers, or between the controllers and the environment (or a central controller), to resolve assignment conflicts or to transmit termination signals, respectively.

Lastly, a partial solution has been offered to the issue of coordinating local assignments with global requirements imposed by the structure of the network. The informed search algorithm introduces a more focused search strategy, by incorporating this global knowledge into a compact set of rules, used at each node to decide which residual goals are implementable on the remaining network. The optimized heuristic carries this idea even further, by deciding, based on these rules, which part of the network should implement the global goal, as well as any intermediary subgoals generated during the decomposition process. Although the availability of global information at the node level would apparently speed up the search, as a result of reduced backtracking, it is also true that the amount of incurred local processing would increase accordingly. It is not clear at this time, whether the advantage of a more focused search outweights the increase in local processing.