The Hopfield-style Network as a Maximal-Clique Graph Machine

Arun Jagota
(jagota@cs.buffalo.edu)
Technical Report 90-25, September 1990
Dept Of Computer Science,
State University Of New York At Buffalo

Abstract

The Hopfield-style network, a *variant* of the popular Hopfield neural network, has earlier been shown to have fixed points (stable states) that correspond 1-1 with the maximal cliques of the underlying graph. The network sequentially transforms an initial state (set of vertices) to a final state (maximal clique) via certain greedy operations. It has also been noted that this network can be used to store *simple, undirected* graphs. In the following paper, we exploit these properties to view the Hopfield-style Network as a *Maximal Clique* Graph Machine. We show that certain problems can be reduced to finding Maximal Cliques on graphs in such a way that the network computations lead to the desired solutions. The theory of NP-Completeness shows us one such problem, SAT, that can be reduced to the Clique problem. In this paper, we show how this reduction allows us to answer certain questions about a CNF formula, via network computations on the corresponding maximal cliques. We also present a novel transformation of finite regular languages to the Clique problem and discuss which (language) questions can be answered and how. Our main general result is that we have expanded the problem-solving ability of the Hopfield-style network without detracting from its simplicity and while preserving its feasibility of hardware implementation.

1 Introduction

The Hopfield network [7] is a well known neural network (or connectionist) model of computation. It is a fully connected network of simple processing units, p_i, with numerically weighted symmetric connections, w_{ij}, between units p_i, p_j. Processing units have states (either discrete in $\{0,1\}$, or continuous in $[0,1]$) depending on whether the discrete or the continuous version of the network is being considered. Each processing unit performs simple and identical computations which generally involve summing (weighted) inputs to the unit, applying an internal transfer function, and changing state if necessary. We will see the precise form of such computations in our variant model. The power of the Hopfield network lies in the connections between units and the strengths (weights) of these connections. An energy function was defined by Hopfield on the states of the network (values of all the units). The energy function, E, in its simplest form is

$$E = -\frac{1}{2}\sum_i \sum_j w_{ij}\ p_j\ p_i \quad - \quad \sum_i w_i\ p_i \tag{1}$$

where we use p_i also to denote the current state (value) of the i_{th} unit and w_i to denote its bias. Hopfield [10] used the fact that the E in (1) is a Lyapunov function (bounded from below) to show that, from any starting state, the network would always converge to some energy minimum upon applying a sequence of asynchronous local state updates (that locally reduced energy).

Since the original formulation, many people have studied various issues related to the Hopfield network, Two key ones are - using continuous units and performing continuous updates [8] and using global parameters such as temperature to guide the state updates [6]. There has also been some work on theoretical computational complexity issues, viewing the network as a computational model [14]. [14] also cites many references to earlier work on these issues.

Because of its ability to converge to local energy minima, the Hopfield network, in principle, has nice properties for content-addressable memories (CAM). In practice, however, the cross-correlation matrix learning rule used in this network limits the storage capacity of the network quite severely. Informally, the capacity of the network is the

number of memories that can be stored in it without *losing* any stored memories or developing *spurious* ones. Many people have tried to formulate alternate learning rules to improve the capacity of the network, but have managed only incremental improvements. Recently though, Shun-Ichi Amari [1], has shown that the capacity of an auto-correlation memory (like the Hopfield network) can be up to $\frac{n^2}{\log n^2}$ if the stored patterns are extremely sparse, where sparseness means for all stored patterns $\bar{x} \in \{0,1\}^n \lim_{n\to\infty} \frac{\bar{x}^T \bar{x}}{n} = 0$.

The Hopfield-style network that we have developed [10], has a different learning and a different update rule which remedies one of the capacity problems. *Any* collection of memories can be stored in the network without losing any of them. No statistical assumptions about the memories that can be stored are needed, nor do the memories have to be sparse. Storing any collection (upto 2^n) of memories requires only $O(n)$ units and hence the network has exponential capacity (Thm 5,6 [10]). This result may have important implications for information storage and retrieval.

We have also found that this network has other attractive properties for content-addressable memories (discussed in [10]) not shared by the Hopfield network. We have applied this network to some non-trivial applications [11, 12]. We have also conducted other kinds of detailed experiments [9, 13]. An important one is storing very large memory bases [13], for which we found that the network has some interesting information processing capabilities.

Outline

In this paper, we will assume knowledge of graph theory, especially the terminology. We will use standard notation as found in [2]. We will develop the view of this network as a general model of computation, without sacrificing the view that it is an extremely simple and physically realisable device with practical applications. The basic idea will be to use the fact [10] that by choosing an appropriate setting for a global network parameter, the stable states of the Hopfield-style network can be made to become the maximal cliques of the underlying graph. The network sequentially transforms an initial state (set of vertices) to a final state (maximal clique) via certain greedy operations. Moreover, any (simple) graph G can be stored in the network so that the graph underlying the network becomes isomorphic to G. Given a problem that can be transformed to a graph whose maximal cliques contain the desired information about the problem, we will first do this transformation and then store the resulting graph in the network. We will then show that certain questions about the original problem can be answered by reaching the appropriate network stable states (maximal cliques of this graph). This is basically why we view the network as a *Maximal Clique* Graph Machine.

This paper is organised as follows. Section 2 formally describes the Hopfield-style network in a restricted form called the *two-state-weights* form (as opposed to *graded-weights* form) and reviews its important properties. There are two main reasons that this form is employed. Firstly, we wish to draw attention to the fact that it is *sufficient* to use only two-valued weights (analogous to 1 bit precision) for the problems in this paper. This could have important implications for analog vlsi implementations. Secondly, certain properties, which we will elaborate upon later, hold only for the *two-state-weights* form. These properties are important for the problems we study here, although they were not required for content-addressable memories (CAM). We remark, informally, that the *two-state-weights* network can still be simulated in the general *graded-weights* network by repeated training (see learning rule in [10] and Section 2 here), which essentially saturates the $+ve$ weights so that they become *two-state*. Section 3 studies the problem of finding large maximal cliques and large maximal f-uniformly-dense vertex sets (a term that will be defined later) in arbitrary graphs, via network computations. Section 4 extends this to problems on Independent Sets and Vertex Covers. Section 5 presents the general way in which problems are mapped onto the network. It also describes an extension to the network. Section 6 uses the well known transformation from SAT to Clique to show how certain questions about CNF formulae can be answered via network computations. Section 7 shows how certain questions about finite regular languages can be answered via network computations. It presents a novel transformation from regular expressions (representing finite regular languages) to graphs. It then shows properties of this transformation that have significance for the network's question answering capability. Section 8 concludes by highlighting and summarising the important results and discussing implications.

2 Model

We now describe the network in its *two-state-weights* form.

The Hopfield-style network is a complete simple undirected weighted graph, with the units as vertices and connections as weighted edges. Let $V = p_1, ...p_n$ denote the set of vertices (units) and w_{ij}, the weight between p_i and p_j. The units in our network have binary states ($\{0,1\}$) only. The initial state of the network is

$$w_{ij}(0) = \rho \quad \forall i,j \quad i \neq j$$

where $\rho < 0$.

Learning Rule

Given any set of vertices $S \subset V$, S can be stored in the network at time t as follows. For all $i \neq j$,

$$w_{ij}(t+1) \;:=\; \begin{cases} 1 & \text{if } p_i,\ p_j \ \in S \\ w_{ij}(t) & \text{otherwise} \end{cases} \tag{2}$$

Notice that any sequence of stores maintains the invariant that for all i and j, $i \neq j$, $w_{ij} \in \{\rho, 1\}$.

Energy descent rule

The update computations are done synchronously. At any time t_k, let p_i denote the state of the unit p_i. Then at time t_{k+1}, for each unit p_i, the quantities n_i, V_i and ΔE_i are computed simultaneously (in parallel), as follows

$$n_i \;=\; \sum_j w_{ij} * p_j \;+\; w_0 \tag{3}$$

$$V_i = \begin{cases} 1 & \text{if } n_i > 0 \text{ and } p_i = 0 \\ -1 & \text{if } n_i < 0 \text{ and } p_i = 1 \\ 0 & \text{otherwise} \end{cases}$$

$$\Delta E_i \;=\; -V_i * n_i \tag{4}$$

where n_i is the net input to unit p_i, V_i is possible direction p_i can switch, ΔE_i is global energy change a switch of unit p_i would cause. w_0 is a new (constant) term in Eqn (3), required for the *two-state* weights form but was not in the *graded-weights* form. It is a very small positive number which represents the bias for p_i to switch ON in the event that $\sum_j w_{ij} p_j \;=\; 0$. We will see that this one extra w_0 term is useful when the network stable states are f-uniformly-dense vertex sets.

Let $p_{i_{min}}$ be the (any if not unique) unit with minimum ΔE. If $\Delta E_{i_{min}} < 0$, then, at time $t_{k\,+\,1}$, $p_{i_{min}}$ is the only unit switched as

$$p_{i_{min}} = \begin{cases} 1 & \text{if } V_{i_{min}} = 1 \\ 0 & \text{o/w} \end{cases}$$

The above update rule sequentially and synchronously performs steepest descent in the energy landscape defined by the E in (1), switching exactly one unit in each cycle in which energy can be lowered.

Relationship between graded-weights and two-state weights form

We must ensure that the *only* effect of w_0 is to bias a unit to switch ON when $\sum w_{ij} p_j \;=\; 0$; that is when $\sum_j w_{ij} p_j \;\neq\; 0$, the network behavior is the same with or without w_0 in Eqn (3). This is a necessary requirement to even consider the possibility that the *graded-weights* form (w_0 is absent) dynamic properties also hold in the *two-state weights* form. We only need to show, for $\sum_j w_{ij} p_j \;\neq\; 0$, that the sign of n_i remains the same with or without w_0. This would mean that V_i remains the same and so ΔE_i from (4) would be relatively the same (with w_0 vs without w_0) since $w_0$is a global constant. Rewrite $\sum_j w_{ij} p_j$ as

$$\rho \sum_{j \text{ st } w_{ij}=\rho} p_j \;+\; \sum_{j \text{ st } w_{ij}=1} p_j$$

Replace the $\sum$'s by $a, b \in N$ respectively and express ρ as $\frac{-x}{y}$. We get,

$$\sum_j w_{ij} p_j \;=\; \frac{-x}{y} a \;+\; b$$

Assuming $\sum_j w_{ij} p_j \;\neq\; 0$,

$$\begin{aligned} min(|\textstyle\sum_j w_{ij} p_j|) \;&=\; min(|\tfrac{-x}{y} a \;+\; b|) \\ &=\; min(|\tfrac{-xa \;+\; by}{y}|) \\ &\geq\; \tfrac{1}{y} \end{aligned}$$

Pick $w_0 < \frac{1}{y} = \frac{1}{\text{Denom}(\rho)}$. Then n_i in (3) has the same sign with and without the w_0 term. For the rest of the paper, we will assume that $w_0 < \frac{1}{\text{Denom}(\rho)}$.

We can simulate the learning rule of the *two-state weights* form in the *graded-weights* form by setting $\lambda = 1, \lambda_R = 0$ in the *graded-weights* form learning equations ([10]). Then the only difference between them is that in the *graded-weights* form, the learning rule will make $w_{ii} = 1$ for all non-isolated vertices in G_n while in the *two-state* weights form, w_{ii} does not exist (implicitly $w_{ii} = 0 \ \forall i$). This difference does not effect the energy-descent behavior. Assuming there are no isolated vertices in G_n, then $w_{ii} = 1, \ \forall i$ (in the *graded-weights* form), and $w_{ii} = 0, \ \forall i$ in the *two-state weights* form, which has the same effect on the energy-descent behavior in both the forms. If there is an isolated vertex, say i, then $w_{ii} = 0$ in both forms.

Properties

We will use the following graph theory notation quite frequently.

$d_G(v)$ - denotes the degree of vertex v in graph G.

$\bar{d}_G(v)$ - denotes the co-degree of v in graph G. $\bar{d}_G(v) = |V(G)| - d_G(v) - 1$.

$d_G(v)|V'$ - denotes the number of neighbors of v in $V' \subseteq V(G)$. $d_G(v)|V' = |N_G(v) \cap V'|$.

$\bar{d}_G(v)|V'$ - denotes V' minus the number of neighbors of v in V'. $\bar{d}_G(v) = V' - |N_G(v) \cap V'|$.

Let $G_n = (V, E)$, be the graph underlying the network defined as follows. V, the set of vertices, corresponds to the units in the network. $(p_i, p_j) \in E$ *iff* $w_{ij} = 1$. That is, G_n has edges for and only for the $+ve$ weights ($w_{ij} > 0$) in the network. G_n is a *simple undirected* graph. Let $S_a(t_i)$ denote the set of active units in the network at time t_i.

The following set of Lemmas/Theorems proved in [10] illustrate the properties of the model and address its suitability for CAM applications. They are valid under the assumption that $\rho < -2n - 2$. More accurately, they are valid under the assumption that $\rho < -2I_m$ (also proved in [10]), where I_m denotes the size of the largest input set (if known) that will be supplied to the network and where it is assumed that the size of the maximum clique in G_n is $\leq I_m$. This may have implications for implementations because it may be difficult to implement $\rho < -2n - 2$ for large n (large network), but if we additionally know that $I_m << n$ (for example if we know that G_n will have only small maximal cliques), then implementing $\rho < -2I_m$ can certainly be more feasible.

Lemma 1 *There is a 1-1 correspondence between the stable states of the network and the* maximal cliques *in the underlying graph G_n.*

Fig 1 shows the graph G_n underlying a network whose stable states, by Lemma 1 are

```
{ a c e }   { a b d }  { a b c }
```

the maximal cliques in G_n. Replacing all the off-diagonal 0 entries in the adjacency matrix of G_n by ρ gives us the weight matrix of the network.

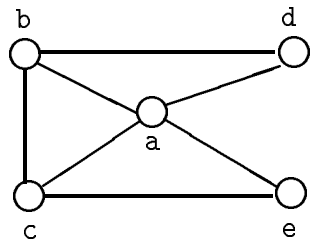


Fig 1 - Gn after training

Theorem 1 *Let the input, $S_a(t_0)$, be presented to the network at time t_0. The network has the following dynamics in the following order.*

1. *The network switches OFF units successively (one by one) in the first $k \geq 0$ steps until the first time $S_a(t_k)$ becomes a clique of G_n. At that time, $|S_a(t_k)| = |S_a(t_0)| - k$.*

2. *The network, then switches ON units successively (one by one) in the next $j \geq 0$ steps until $S_a(t_{k+j})$ is a maximal clique of G_n. At that time, $|S_a(t_{k+j})| = |S_a(t_k)| + j$.*

The network rests at $S_a(t_{k+j})$.

Lemma 2 *If $S_a(t_i)$ is not a clique, then at time t_{i+1}, the unit inhibited by the largest number of units in $S_a(t_i)$ will be switched OFF.*

There are more properties that have implications for CAM but are not relevant to our discussion here. The above Lemmas/Theorems (proved in [10]) and some others not discussed here but in [10] also hold in the *two-state weights* network form. This is because we have seen in the last section that the behavior of the two-state weights form is identical to that of the graded-weights form with $\lambda = 1, \lambda_R = 0$, for which these properties obviously hold. As a side note we add that even in the original proofs of these properties (in [10]), the values of w_{ii} were irrelevant (as long as they were in $[0, 1]$). This shows that many desirable properties of our model are retained even under such a restriction. This restriction, as mentioned earlier, does have the advantage of improving the feasibility of analog implementations of the model (since it employs only low precision (two-valued) weights).

It is quite important (and fortunate) that these properties hold in the *two-state weights* form, because we will need them for the problem solving tasks explored in this paper. We now turn our attention to the first problem.

3 Cliques and f-uniformly-dense vertex sets in graphs

Finding large maximal cliques

The network can find large maximal cliques in graphs, specifically by emulating the greedy algorithm that does so. Given any simple undirected graph $G = (V, E)$, it can be loaded into the network by presenting its edges one by one, as binary sets of vertices during training (application of the learning rule). After all the edges have been presented, G_n, the graph underlying the network is isomorphic to G. The training requires $|E|$ presentations. Since only simple undirected graphs can be loaded into the network and we will be dealing only with these, from now on in this paper we will use the term graph to mean *simple undirected* graph.

Let the initial state of the network be $S_a(t_0)$. By Lemmas 1,2 and Theorem 1, if $\rho < -2I_m$, it can be readily seen that the network emulates the following algorithm. It should be noted that the algorithm makes choices non-deterministically if there are multiple vertices that satisfy the criterion to pick v in the two statements. The non-determinism of the network, when $p_{i_{min}}$ (unit with minimum energy ΔE_i) is not unique, occurs exactly and in the same way as when the algorithm makes choices non-deterministically.

$G_i \leftarrow G[S_a(t_0)]$;

<u>while</u> $V(G_i)$ is not a clique <u>do</u>

Pick $v \in V(G_i)$ such that $d_{G_i}(v) = \delta(G_i)$;
$G_i \leftarrow G_i - \{v\}$;

<u>end</u>;

<u>while</u> $\exists x \in V(G) - V(G_i)$ such that $V(G_i) \cup \{x\}$ is a clique <u>do</u>

Pick $y \in V(G) - V(G_i)$ such that $V(G_i) \cup \{y\}$ is a clique;
$G_i \leftarrow G_i \cup \{y\}$;

<u>end</u>;

Let $S_a(t_0)$, the initial state of the network, be set to $V(G)$, the set of all vertices. The above algorithm then becomes the greedy algorithm for finding a large maximal clique in G. It is well known that after the algorithm terminates, the size of the clique is,

$$\geq \sum_{v \in V(G)} \frac{1}{|V(G)| - d_G(v)}$$

Stable states as f-uniformly-dense vertex sets

The network stable states can also be made f-uniformly-dense vertex sets (cliques are a special case), by setting ρ in $[-2I_m, 0)$ instead of $< -2I_m$. First, we define the concept of an f-uniformly-dense vertex set, a generalisation of the notion of a clique (when $f = \infty$).

Definition 1 *Let $G = (V, E)$ be a graph and $G' = G[V']; V' \subseteq V$ be the V' induced subgraph of G. We say that V' is an f-uniformly-dense vertex set, $0 \leq f \leq \infty$* iff

$$\forall v \in V', \quad \frac{d_{G'}(v)}{\bar{d}_{G'}(v)} \geq f$$

We use uniform-density(V') to denote the maximum f for which V' is an f-uniformly-dense vertex set.

We digress for a moment to consider what this generalisation means in terms of constraint satisfaction. The stable states being maximal cliques ($\rho < -2I_m$) means that in the solution (stable state), constraints must be satisfied in a hard fashion, namely, every *active* unit must be $+ve$-ly correlated with every other *active* unit. Moreover, such a collection of units must be maximal, meaning that no *inactive* unit is $+ve$-ly correlated with all the *active* units. The stable states being f-uniformly-dense vertex sets (ρ in $[-2I_m, 0)$) means that in the solution (stable state), constraints can be satisfied weakly, specifically, every *active* unit need be $+ve$-ly correlated only with a minimum fraction of the other *active* units. This fraction, by rewriting the expression in definition 1, turns out to be $\frac{f}{1+f}$. Also as we shall see, f can be controlled by setting ρ, the global network parameter. Indeed $f = -\rho$.

For the two-state weights form, the energy-descent rule equation (3) can be written in the following form, which will be quite convenient for the proofs that follow. For any unit p_y,

$$n_y(t_{i+1}) = \rho\, \bar{d}(y)|S_a(t_i) + d(y)|S_a(t_i) + w_0 \tag{5}$$

For the *graded-weights* network form, the same energy-descent rule equation (3) for p_y can be written as

$$n_y(t_{i+1}) = \rho\, \bar{d}(y)|S_a(t_i) + \sum_{x \in d(y)|S_a(t_i)} w_{xy} \tag{6}$$

where $w_{xy} \in (0, 1]$.

We now formally present f-uniformly-dense vertex set related network properties. We will present positive results (that a property holds) and negative results (that a property does not hold). If a positive result holds for the *graded-weights* form, it automatically holds for the *two-state* weights form, since the latter is a special case of the former. If a negative result holds for the *two-state* weights form, it automatically holds for the *graded-weights* form, for the same reason. We will also present some positive results that hold only for the *two-state* weights form. After each such result, we will include a remark explaining why it does not hold for the *graded-weights* form. We have examples to support all such remarks but have excluded them from this paper for conciseness, since they belong to *graded-weights* form issues.

Using this new (f-uniformly-dense sets) definition, we first re-state the following Lemma , originally proved in [10].

Lemma 3 *A necessary condition for S to be a network stable state is that*

$$\textit{uniform-density}(S) \geq -\rho \tag{7}$$

Lemma 3 was proven for the *graded-weights* network form and hence also holds in the *two-state weights* form. Lemma 3 does not require that, to be a network stable state, the vertex set satisfy the condition of Lemma 3 maximally. In the grade-weights form, examples of this are easy to construct. This contrasts with Lemma 1, where a network stable state must satisfy the property of being a clique maximally. As the following negative result shows, even in the two-state weights form, a network stable state need not satisfy the condition of Lemma 3 maximally. This automatically implies that the same holds for the graded-weights form.

Negative Result 1 *In the two-state weights form, there can be vertex sets satisfying the condition of Lemma 3 non-maximally but still be network stable states.*

Example : Let the graph in Fig 1b be G_n, the graph underlying the network, and let $\rho = -2.2$. From Fig 1b and Eqn (5), if $S_a(t_i) = \{a, b, e\}$, then $n_c(t_{i+1}), n_d(t_{i+1}), n_f(t_{i+1}) < 0$. So none of these units can switch ON. From Fig 1b uniform-density($\{a, b, e\}$) $\geq 2.2 = -\rho$ and hence by Lemma 4, no unit in $S_a(t_i)$ will switch OFF. Hence {a,b,e} is

a network stable state. From Fig 1b, uniform-density($\{a, b, e, f, c\}$) $\geq 2.2 = -\rho$ so that {a,b,e} satisfies the condition of Lemma 3 but not maximally.

Also by Lemma 1, when $\rho < -2I_m$, a sufficient condition for a vertex set S to be a network stable state is that it be a maximal clique. For arbitrary ρ, this sufficient condition cannot be generalised to a condition in terms of the uniform-density of S. In the *graded-weights* form, its insufficiency can be shown by easy to construct examples. As the following negative result shows, its insufficiency can be shown even in the *two-state weights* form, which also automatically implies its insufficiency in the *graded-weights* form.

Negative Result 2 *In the two-state weights form, there can be vertex sets S that maximally satisfy the condition uniform-density$(S) \geq -\rho$ but are still not network stable states.*

Example : Let the graph in Fig 1c be G_n, the graph underlying the network, and let $\rho = -1$. We note that uniform-density$(\{a, b, c\}) = 1 \geq -\rho$. Moreover, uniform-density$(\{a, b, c, d\}) = \frac{1}{2} < -\rho$, so {a,b,c} is maximal for this property. From Fig 1c $\frac{d(d)|\{a,b,c\}}{\bar{d}(d)|\{a,b,c\}} = 2 \geq -\rho$ and by Eqn (5), if $S_a(t_i) = \{a, b, c\}$, then $n_d(t_{i+1}) > 0$ which means d will switch ON at t_{i+1}. Hence a,b,c is not a network stable state.

The above two negative results show that the strong results of Lemma 1 cannot be extended for arbitrary ρ, something that we had hoped and expected would happen. They also strongly suggest that the results of Thm 1 and Lemma 2 cannot be extended. We find weaker positive results, which are shown below. We also state negative results in cases where intuition indicates that a generalisation of some property of Thm 1 or Lemma 2 should hold, but in reality it does not. All the results below apply to the *two-state weights* form and so the negative results also apply to the *graded-weights* form. Following the positive results, remarks are included to indicate that they do not hold in the *graded-weights form* and brief informal explanations of why they don't. This shows that the restriction to *two-state weights* form does give us additional properties.

Lemma 4 *uniform-density$(S_a(t_i)) \geq -\rho \Rightarrow$ no unit switches OFF at t_{i+1}.*

Proof: Suppose uniform-density$(S_a(t_i)) \geq -\rho$. By definition of uniform-density,

$$\forall x \in S_a(t_i) \qquad \frac{d(x)|S_a(t_i)}{\bar{d}(x)|S_a(t_i)} \geq -\rho$$

From this and Eqn (5),

$$\forall x \in S_a(t_i) \qquad n_x(t_{i+1}) > 0$$

It follows that no unit can switch OFF at t_{i+1}. □

Remark: Lemma 4 does not hold in the *graded-weights* form because, in general, it is possible to choose $w_{xy} \in (0, 1]$ sufficiently small so that even when uniform-density$(S_a(t_i)) \geq -\rho$, by Eqn (6)

$$\exists x \in S_a(t_i) \text{ such that } n_x(t_{i+1}) < 0$$

By Thm1, when $\rho < -2I_m$, if $S_a(t_i)$ is not a clique, then at time t_{i+1} a unit must switch OFF. Counter to intuition, this behavior does not generalise to arbitrary ρ.

Negative Result 3 *In the two-state weights form, uniform-density$(S_a(t_i)) < -\rho$ does not imply that a unit switches OFF at t_{i+1}.*

Example : Let the graph in Fig 1d be G_n, the graph underlying the network, and let $\rho = -2$. We note that uniform-density$(\{a, b, c\}) = 1 < -\rho$. But from Fig 1d and Eqn (5), if $S_a(t_i) = \{a, b, c\}$, then

$$n_a(t_{i+1}) = -1,\ n_c(t_{i+1}) = -1, n_b(t_{i+1}) = 2,\ n_d(t_{i+1}) = 3$$

. Hence d will switch ON at t_{i+1}.

Another result of Thm1, when $\rho < -2I_m$, is that if $S_a(t_i)$ is a clique, but not a maximal one, then at time t_{i+1},

1. A unit must switch ON.
2. $S_a(t_{i+1})$ must still be a clique.

Item 1) generalises to arbitrary ρ, as shown by Lemma 5 below. Counter to intuition, item 2) does not, however, generalise, as shown by Negative Result 4.

Lemma 5 *If uniform-density$(S_a(t_i)) \geq -\rho$ and $\exists x \in V(G_n) - S_a(t_i)$ such that $\frac{d(x)|S_a(t_i)}{\bar{d}(x)|S_a(t_i)} \geq -\rho$ then at time t_{i+1}, some unit $y \in V(G_n) - S_a(t_i)$ such that $\frac{d(y)|S_a(t_i)}{\bar{d}(y)|S_a(t_i)} \geq -\rho$ will switch ON.*

Proof: Since uniform-density$(S_a(t_i) \geq -\rho$, By Lemma 4, no unit in $S_a(t_i)$ can switch OFF. Substituting the particular unit x that satisfies the condition of Lemma 5 into Eqn (5), $n_x(t_{i+1}) > 0$. Hence, if no other unit switches ON, p_x will switch ON, at time t_{i+1}. Suppose some other unit p_y switches ON. This means $n_y(t_{i+1}) > 0$. Noting that w_0 is sufficiently small not to interfere, by Eqn (5)

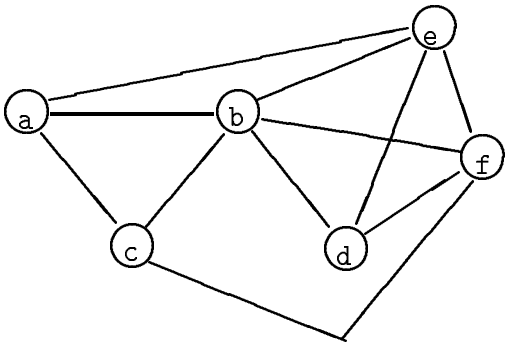
$$\frac{d(y)|S_a(t_i)}{\bar{d}(y)|S_a(t_i)} \geq -\rho$$

□

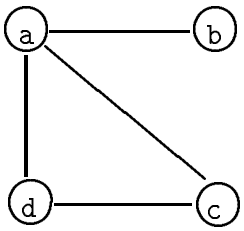
Remark: Lemma 5 does not hold in the *graded-weights* form. To see this, we first recall that Lemma 4 does not hold in the *graded-weights* form. This means that when uniform-density$(S_a(t_i)) \geq -\rho$, which is the pre-condition of Lemma 5, we can't exclude the possibility that a unit $p_x \in S_a(t_i)$ will not switch OFF. Even if Lemma 4 were to hold for the graded-weights form, it is possible to choose, for every $p_y \in V(G_n) - S_a(t_i)$ satisfying $\frac{d(y)|S_a(t_i)}{\bar{d}(y)|S_a(t_i)} \geq -\rho$, the positive weights $w_{xy} \in (0, 1]$, from all $p_x \in S_a(t_i)$, sufficiently small, so that according to Eqn (6), $n_y(t_{i+1}) < 0$, which would mean that no such p_y could switch ON at t_{i+1}.

Negative Result 4 *If uniform-density$(S_a(t_i)) \geq -\rho$ and at time t_{i+1} some unit switches ON, then it does not imply that uniform-density$(S_a(t_{i+1})) \geq -\rho$.*

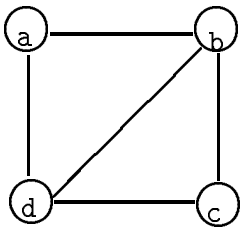
Example : Again, let the graph in Fig 1c be G_n, the graph underlying the network, and let $\rho = -1$. With $S_a(t_i) = \{a, b, c\}$, uniform-density$(S_a(t_i)) \geq -\rho$ and $\frac{d(d)|S_a(t_i)}{\bar{d}(d)|S_a(t_i)} = 2 \geq -\rho$ so that $n_d(t_{i+1}) > 0$. d switches ON at t_{i+1} and so $S_a(t_{i+1}) = \{a, b, c, d\}$ But then $\frac{d(b)|S_a(t_{i+1})}{\bar{d}(b)|S_a(t_{i+1})} = \frac{1}{2}$ which means uniform-density$(S_a(t_{i+1})) < -\rho$.



(b)



(c)



(d)

Fig 1 b,c,d - Examples for negative results

By Lemma 2, when $\rho < -2I_m$, if $S_a(t_i)$ is not a clique, then at time t_{i+1}, a unit (in $S_a(t_i)$) inhibited by the largest number of active units will be switched OFF. For $\rho < -2I_m$, this *is* the switch OFF property responsible for the network's greedy behavior and exploited to emulate the clique finding greedy algorithm. Unfortunately, by Negative Result 3, we have already seen that it does not generalise to arbitrary ρ, since uniform-density$(S_a(t_i)) < -\rho$ does not even guarantee that a unit will switch OFF at time t_{i+1}. We can still generalise it partly, so that if a unit does switch OFF, then it will be a one inhibited by the largest number of active units, as shown by Lemma 6.

Lemma 6 *If a unit switches OFF at time t_{i+1}, then*

1. *uniform-density$(S_a(t_i)) < -\rho$*
2. *This unit is inhibited by the largest number of units in $S_a(t_i)$.*

Proof:
Part 1 - Let $p_s \in S_a(t_i)$ be the unit that switches OFF. $n_s(t_{i+1}) < 0$ and from Eqn (5)

$$\frac{d(s)|S_a(t_i)}{\bar{d}(s)|S_a(t_i)} < -\rho$$

which means uniform-density$(S_a(t_i)) < -\rho$.
Part 2 - Let $p_l \in S_a(t_i)$ such that $d(p_j)|S_a(t_i) = \delta(G_n[S_a(t_i)])$. In other words, p_l is an active unit inhibited by the largest number of active units. Suppose p_s is inhibited by fewer active units than p_l, that is, $\bar{d}(p_s)|S_a(t_i) < \bar{d}(p_l)|S_a(t_i)$. This means

$$d(p_s)|S_a(t_i) > d(p_l)|S_a(t_i)$$

Also, since $\rho < 0$,

$$\rho\ \bar{d}(p_l)|S_a(t_i) < \rho\ \bar{d}(p_s)|S_a(t_i)$$

From Eqn (5), $n_l(t_{i+1}) < n_s(t_{i+1})$ and since $n_s(t_{i+1}) < 0$, p_l will switch OFF at t_{i+1}, which contradicts assumption. Hence p_s must be inhibited by at least the same number of units as p_l is by $\Box$.
Remark: Neither Part I nor Part II of Lemma 6 hold in the *graded-weights* form. Part I does not hold because it is possible to choose the positive weights amongst units in $S_a(t_i)$ sufficiently small so that a unit in $S_a(t_i)$ can be switched OFF even if uniform-density$(S_a(t_i)) \geq -\rho$. Part II does not hold because it is possible to choose the positive weights from other units in $S_a(t_i)$ to p_l relatively small while choosing positive weights from other units in $S_a(t_i)$ to p_s relatively large, so that even though p_l is inhibited by more units in $S_a(t_i)$ than p_s, p_s receives more inhibition, that is $n_s(t_{i+1}) < n_l(t_{i+1})$.

Testing a vertex set for clique, f-uniformly-dense property

The network can be used to *test* if $S \subseteq V$ is a clique, based on the decision that the network makes in the first cycle with S as input. Assume $\rho < -2I_m \leq -2n - 2$. Let $S_a(t_0) \leftarrow S$.

1. If no unit switches at t_1, by Lemma 1, S is a maximal clique.
2. If a unit switches ON at t_1, by Thm1, S is a clique but not maximal. The V flag for the unit that switches indicates the direction of switch (ON or OFF).
3. If a unit switches OFF at t_1, by Thm 1, S is not a clique.

The network can also be (sometimes) used to *test*, for $S \subseteq V$, whether uniform-density$(S) \geq -\rho$ or not. Set $\rho = -f$. Let $S_a(t_0) \leftarrow S$.

1. If no unit switches at t_1, by Lemma 3, uniform-density$(S) \geq -\rho = f$
2. If a unit switches OFF at t_1, by Lemma 6 uniform-density$(S) < -\rho$.

We see that a unit switching ON conveys no information about the uniform-density of S.

4 Independent Sets, Vertex Covers in graphs

Independent Set problems

We first define the concept of an f-uniformly-sparse set of vertices, a generalisation of the notion of Independent Set (when $f = 0$).

Definition 2 *Let $G = (V, E)$ be a simple undirected graph and $G' = G[V']; V' \subseteq V$ be some induced subgraph of G. We say that V' is an f-uniformly-sparse set,$0 \leq f \leq 1$* iff

$$\forall v \in V', \quad \frac{d_{G'}(v)}{\bar{d}_{G'}(v)} \leq f$$

We use uniform-sparsity(V') to denote the minimum f for which V' is f-uniformly-sparse.

Problems related to the Independent set (IS) can be mapped straightforwardly. Given G, load G^c, its complement, onto the network. The following IS problems on G can be formulated as clique problems on G^c and answered by the network.

1. Find a *large* maximal independent set. The network emulates the greeding algorithm for doing so. It is easy to show that the independent set I found by the algorithm meets the following bound

$$\begin{aligned} |I| &\geq \sum_{v \in V(G)} \frac{1}{1 + d(v)} \\ &\geq \frac{|V|}{1 + \bar{d}} \end{aligned}$$

   where $\bar{d}$ is the average degree in G.

2. Find a set $V' \subseteq V$ such that uniform-sparsity(V') $\leq \frac{1}{-\rho}$. Any stable state that the network reaches is guaranteed to satisfy this condition, because by Lemma 3, it will find V' in G^c such that uniform-density(V') $\geq -\rho$. However, there is no guarantee that V' will satisfy this condition maximally and since the network does not emulate the greedy algorithm, there is no guarantee that V' will even be a large set.

3. Test if $S \subseteq V$, is a maximal independent set, a non-maximal independent set or not an independent set (see subsection on Testing a set for clique).

4. If no unit switches with $S \subseteq V$ as $S_a(t_0)$, uniform-sparsity(S) $\leq \frac{1}{-\rho}$. This follows from Lemma 3.

5. If a unit switches OFF with $S \subseteq V$ as $S_a(t_0)$, uniform-sparsity(S) $> \frac{1}{-\rho}$. This follows from Lemma 6.

Vertex Cover Problems

We first define the concept of an f-edge-cover set of vertices, a generalisation of the notion of Vertex Cover (when $f = 0$).

Definition 3 *Let $G = (V, E)$ be a simple undirected graph and $G' = G[V']; V' \subseteq V$ be some induced subgraph of G. We say that V' is an f-edge-cover set,$0 \leq f \leq 1$ if the number of edges of G not covered by V' is $\leq f(|V - V'| - 1)(|V - V'|)$.*

We use edge-cover(V') to denote the minimum f for which V' is an f-edge-cover set.

Problems related to the Vertex Cover (VC) can also be mapped straightforwardly. Given G, load G^c, it's complement, into the network. The following VC problems on G can be formulated as IS problems on G, which in turn can be formulated as clique problems on G^c and answered by the network.

1. Find a small minimal vertex cover of G. First find I, a large IS as earlier. Then $V - I$ is a small minimal vertex cover. Again, the network is emulating the greedy algorithm for doing so.

2. Find a vertex set V' such that edge-cover(V') $\leq \frac{1}{1 - \rho}$. Again, this is done by finding an f-uniformly-sparse vertex set I as before. Now $I = V - V'$ and since uniform-sparsity($V - V'$) $\leq \frac{1}{-\rho}$ and $\rho < 0$,

$$\begin{array}{lll} \forall v \in V - V' & & \\ \frac{d(v)}{\bar{d}(v)} & \leq & \frac{1}{-\rho} \\ \Leftrightarrow \frac{d(v)}{|V - V'| - 1} & \leq & \frac{1}{1 - \rho} \end{array}$$

Hence the number of edges in $G^c[V-V']$ is $\leq \frac{1}{1-\rho}(|V-V'|-1)(|V-V'|)$ and hence edge-cover$(V') \leq \frac{1}{1-\rho}$. Again, we cannot guarantee that V' will be minimal, for this property, nor that it will be small, due to our inability to emulate the greedy algorithm.

3. Test if $S \subseteq V$ is a minimal vertex cover, a non-minimal vertex cover or not a vertex cover. Again, this can be done by testing if $V - S$ is a maximal independent set, a non-maximal independent set or not an independent set respectively.

4. If no unit switches with $S \subseteq V$ as $S_a(t_0)$, then edge-cover$(V-S) \leq \frac{1}{1-\rho}$. This follows from Lemma 3.

5. If a unit switches OFF with $S \subseteq V$ as $S_a(t_0)$, then uniform-sparsity$(V-S) > \frac{1}{1-\rho}$. This follows from Lemma 6.

5 General Problem Solving Framework

Now that we have seen how the Clique, Independent Set and Vertex Cover problems can be mapped onto the network, we sketch the general method of solving problems with the Hopfield-style network. Withing our scope, a problem A is an object (or property) along with that collection of questions that can be asked about it which are answerable via the Hopfield-style Network computations. For example, graphs and Independent Set related questions on them that are answerable by our network form the Independent Set Problem, for us. More specifically, the object in A must be transformable to a graph so that some question about A that we want answered can be represented as a set of vertices in the transformed graph. The answer to this question must be contained in the state transition trajectory that the network will follow, from this set of vertices as the network initial state. The state transition trajectory is governed by the network's energy-descent rule and known dynamic properties. Answers to most questions will be contained in some property of the final (stable) state (usually maximal clique) that the network reaches from the question as the initial state. This property might be the set of vertices in this stable state, or even just its size. This is why we choose to call this network a *Maximal Clique* machine. In general though, some questions, like testing whether a vertex set is a clique, might be answerable by some part of the trajectory itself, in this case from the response of the network in the first cycle.

We have classified the information processing capabilities of the network into the following categories along two dimensions, the computations it can perform, and what information can be extracted from the stable state it reaches. These categories serve to identify the different types of questions that the network can answer, independent of specific problems. We have already seen how some of these computations are used to answer questions about the Clique, Independent Set and Vertex Cover problems. The other information processing capabilities, like **size-decision**, require an extension to the model that we will see soon. We will then see how they can be used to answer more questions. We use the term network *stable state* only in describing these information processing capabilities knowing fully well that these stable states can be controlled to be the maximal cliques in G_n or they could be controlled to be the maximal f-uniformly-dense vertex sets ($f = -\rho$) in G_n depending on the questions we wish to be answered, by choosing ρ appropriately. The information processing capabilities described below are the same, regardless of which of these two is the case. The stable states being f-uniformly-dense vertex sets allows answering generalised versions of questions when the stable states are maximal cliques. For example, we might ask **Is $V' \subset V(G)$ of a graph G a vertex set of size $> k$ and whose uniform-density is $\geq$ 2?**. This is a **state tester** computation (see below) using **stable state size-threshold** information (see below) on the stable states which are chosen to be f-uniformly-dense vertex sets ($f = 2$) by choosing $\rho = -2$.

Computations

large stable state computation Find a large stable state of the network. We've seen how to do this earlier (Cliques, f-uniformly-dense sets).

transducer computation Transform some network initial state (vertex set) to a final (stable) state , according to the network energy-descent rule.

state tester computation Test if the initial state is a maximal clique (or f-uniformly-dense vertex set) of G_n, a non-maximal clique (or f-uniformly-dense vertex set) of G_n or not a clique (f-uniformly-dense vertex set) of G_n. As we have seen earlier, this decision can be made from just the network behavior in the first cycle.

Stable State Information

stable state size-threshold Is the size of this stable state (the network has reached) $\geq$ *theta*?

stable state size Output the size of this stable state (the network has reached).

stable state set Output the vertices in this stable state (the network has reached).

We would like to digress a bit and remark that CAM applications of the network, like those we have previously studied ([11, 12], [13]) can also be viewed within this framework. Normally, CAM applications utilise **transducer computations** with **stable state set** information for memory retrieval. But the additional computation and information categories delineated above indicate what additional questions about stored CAM memories can be answered by the network. The reason these additional computations are also meaningful for CAM applications is that after training on CAM memories, the graph underlying the network is just some graph. On this graph all these computations are defined as for any other graph. One problem, however, is that CAM storage can produce *spurious* memories which means that a computation on the underlying graph cannot, in general, be interpreted as an answer to some question about the stored memories. For example, if we ask the network to retrieve a *large* stored memory it could indeed retrieve a large maximal clique in G_n, via a **large stable state computation** (and **stable state set** information), but this clique may represent a spurious rather than a stored memory. Nevertheless, in situations where it can be shown that spurious memories don't form, we can answer a variety of questions on the stored memories, via the network computations. In addition to **transducer computations**, the following are probably useful computations for CAM applications; **large stable state computations** (retrieving some large stored memory) and **state tester computations** (testing if an input is a stored memory). The work in [13] is based on **state tester** computations.

Model extension for *size* and *size-threshold* stable state information

Let q_A be a question (boolean:yes/no answer) answerable only via a **size-threshold** (threshold externally settable) on the stable state the network reaches, from it as input. In other words, answering q_A entails checking if, from the question as input, the size of the stable state the network reaches exceeds a certain threshold. We have implicitly assumed that the clique sizes corresponding to the two answers are linearly separable, namely that a threshold θ exists so that the answer is 'yes' if the final state clique size is on one side of θ, and 'no' otherwise.

Alternatively, let q_A be a question whose answer lies in the **size** of the stable state the network reaches, from it as input. We wish to extend the network to allow extraction of both kinds of information, as and when needed. We build a perceptron denoted by P, whose input represents the state of the network, as follows. The perceptron accepts n inputs (the outputs of the n units of the network) and has n fixed weights (of value 1). The perceptron computes n_P, a linear function of the input followed by T_P, a hard-threshold on the value of n_P as shown below. The threshold parameter θ can be externally set.

$$n_P(t_{k+1}) \;=\; \sum_j p_j(t_k) \;-\theta \;=\; |S_a(t_k)| \;-\; \theta$$

$$T_P(t_{k+1}) = \begin{cases} 1 & n_P(t_{k+1}) \geq 0 \\ 0 & \text{o/w} \end{cases}$$

T_P can also be interpreted as a θ-out-of-n threshold logic gate. But the perceptron view is more general since it is a two unit device, n_P allowing us access to the size of the stable state (by setting $\theta = 0$), and T_P allowing access to the binary thresholded decision, where the threshold can be externally set.

6 CNF Problems

We will show how the network can answer some questions about Boolean Formulae in CNF (conjunctive normal form). We will use the well-known transformation from SAT (a decision problem on CNF formulae) to Clique. The transformation from SAT to vertex cover is described in [3]. The transformation from SAT to IS is quite similar. Readers familiar with this transformation might choose to view the SAT to clique transformation as a SAT to IS transformation followed by taking the graph complement. Here, we present the transformation from SAT to Clique explicitly, using the same notation and style as in [3].

SAT to Clique transformation

Let $U = \{u_1, u_2, \ldots u_n\}$ be a set of variables and $C = \{c_1, c_2, \ldots c_m\}$ be a set of clauses making up an arbitrary *CNF* formula. We must construct a graph $G = (V, E)$ and a positive integer $K \leq |V|$ such that G has a clique of size $\geq K$ if and only if C is satisfiable. The construction will be made up of several components. For each variable $u_i \in U$, there is a truth setting component $T_i = (V_i, E_i)$ with $V_i = \{u_i, \bar{u}_i\}$ and $E_i = \phi$, that is, two isolated vertices. Note that any clique can contain at most one of $u_i, \bar{u}_i$ since they are not adjacent. We take the *join* of all the truth-setting components to form T, a $T_{2,n}$ graph, that is, a *complete* n-partite graph, each partition containing two vertices. Each partition represents a particular T_i. For each clause $c_j \in C$, there is a satisfaction testing component $S_j = (V'_j, E'_j)$.

$$\begin{aligned} V'_j &= \{ a_1[j], a_2[j], \ldots a_k[j]\} \\ E'_j &= \phi \end{aligned}$$

, where $a_i[j]$ represents the i^{th} literal in clause c_j. Note that any clique can contain at most one vertex from V'_j since S_j is empty. We take the *join* of all the satisfaction testing components to form S, a *complete* m-partite graph, each partition containing $|c_i|$ vertices, for the corresponding S_i. The only part of the construction that depends on which literals occur in which clauses is represented by the absence of certain *communication* edges, edges between T and S. The absence of a particular communication edge $\{t, s\}$ where $t \in V(T), s \in V(S)$ indicates that if the literal associated with vertex t is ON, then the vertex s, denoting its complement in a particular clause, cannot be ON. We first take the *join* of all the satisfaction testing components with all the truth-setting components, denoted by

$$T \bowtie S$$

and meaning that every vertex in T is adjacent to every vertex in S. For each clause $c_j \in C$, let the k literals in c_j be denoted by $x_{1_j}, x_{2_j}, \ldots x_{k_j}$. Then the communication edges for S_j that should be absent from $T \bowtie S$ are given by

$$E''_j = \{\{a_1[j], \bar{x}_{1_j}\}, \{a_2[j], \bar{x}_{2_j}\}, \ldots, \{a_k[j], \bar{x}_{k_j}\}\}$$

We subtract these edges from $T \bowtie S$, for all j. The construction of our instance of clique is completed by setting $K = n + m$ and

$$G = (T \bowtie S) - (\cup_{j=1}^{m} E''_j)$$

Fig 2 shows an example of the graph obtained when $U = \{u_1, u_2, u_3, u_4\}$ and $C = \{\{u_1, \bar{u}_3, \bar{u}_4\}, \{\bar{u}_1, u_2, \bar{u}_4\}\}$. This example is taken from [3]. For convenience of illustration, edges between T and S are not shown. Every vertex in S is adjacent to every vertex in T, *except* for those pairs connected by **dotted** lines.

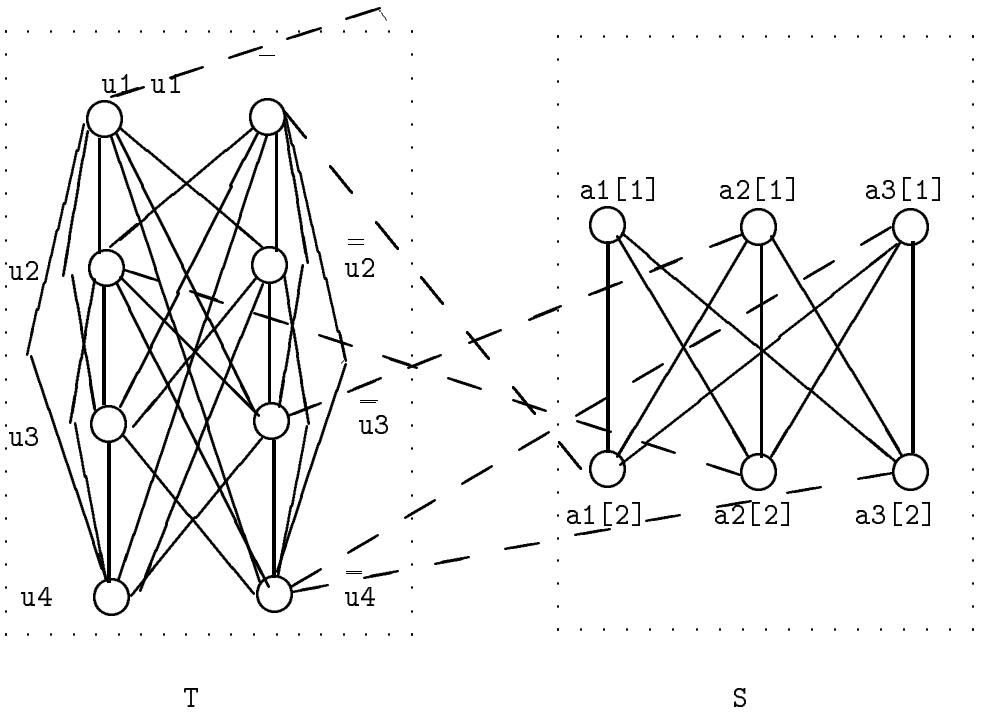


Fig 2 - Clique instance from SAT instance U = {u1,u2,u3,u4}

$\bar{C}$ = {{u1,$\bar{u3}$,$\bar{u4}$},{u1,$\bar{u2}$,u4}} $\bar{K}$ = n + m = 6

It is well known that C is satisfiable if and only if G has a clique of size K. The proof is similar to the one for vertex cover in [3].

Mapping CNF to the network

Given a *CNF* Boolean formula C, the clique graph G is constructed as above and stored in the network with $\rho < -2I_m$ ($I_m = n + m$). We now see what questions about C we can answer from this graph stored in our network. Consider the question of testing if a particular assignment of truth-values is a satisfying one. The following Lemma is crucial to showing that the network can answer such a question. Specifically, Lemma 7.1, 7.2 show that to test if a truth-value assignment is a satisfying one, it is sufficient to check the size of of *any* maximal clique which is a superset of that truth-value assignment. Although this may be quite obvious, Lemma 7.3 explicitly shows that given *any* maximal clique of size $n + m$, the n vertices from the truth-setting components in this clique form a satisfying assignment.

Lemma 7 *Let ST be the set of vertices in G (from the truth-setting components) that correspond to a truth-value assignment to C. Then*

1. IF ST satisfies C THEN
   the size of every maximal clique MC such that ST is a subset of MC
   is n + m

2. IF ST does not satisfy C THEN
   the size of every maximal clique MC such that ST is a subset of MC
   is less than n + m

3. IF (MC is a maximal clique of size n + m) THEN
   the intersection of MC and T is a satisfying assignment of C

Proof:

1. Let ST be a truth-value assignment that satisfies C and let MC be a maximal clique such that $ST \subset MC$. Clearly, $|MC| \leq n + m$. Suppose $|MC| < n + m$. MC contains at most one vertex from each S_i (since $E(G[S_i]) = \phi$). Since $|ST| = n$ and there are m clauses, this means that $\exists S_i$ such that $MC \cap S_i = \phi$.
   Claim: $\exists v \in S_i$ adjacent to all vertices in ST.
   Suppose not. Then $\forall v \in S_i$, $\exists s \in ST$ such that v is not adjacent to s But this means c_i cannot be satisfied by ST which contradicts ST is a satisfying assignment $\Box$.
   Choose the v that satisfies the claim. Since $S_i \cap MC = \phi$, v is also adjacent to all vertices in $S \cap MC$. It follows that v is adjacent to all the vertices in MC. But this contradicts the assumption that MC is a maximal clique.

2. Suppose MC is a maximal clique such that $ST \subset MC$. Suppose ST does not satisfy C. $\exists c_i$ such that ST does not satisfy c_i.
   $$\forall v \in S_i, \exists s \in ST \text{ such that } v \text{ is not adjacent to } s$$
   But then $MC \cap S_i = \phi$ which means $|MC| < n + m$.

3. Suppose MC is a clique of size $n + m$. MC contains m vertices from S and n vertices from T. $MC \cap T$ represents a truth-value assignment to C. Suppose that $MC \cap T$ is not a satisfying assignment. From 2) above, $|MC \cap T| < n + m$ which contradicts assumption. Hence $MC \cap T$ *is* a satisfying assignment. $\Box$.

We will now see that the network can always decide if a truth-value assignment satisfies C or not.

Corollary 1 *Suppose G is stored in the network. Let the network initial state be the set of vertices corresponding to a truth-value assignment X of C. Then X is a satisfying assignment if and only if the network rests at a stable state whose size is $n + m$.*

Proof: Since X is a legal assignment, $G[S_a(t_0)]$ is complete. By Thm 1 the network will never switch OFF a unit. Hence $S_a(t_0) \subseteq S_a(t_f)$ where $S_a(t_f)$ is the network final state and by Lemma 1 it is a maximal clique. By Lemma 7

$$|S_a(t_f)| = n + m \quad iff \quad X \text{ satisfies } C$$

$\Box$.

The final test (indicated by Corollary 1) after the network has reached a stable state can be done by a **size-threshold** decision (setting $\theta = n + m$) on this stable state.

Corollary 2 *The number of clauses satisfied by X, a truth-value assignment, is always at least $|S_a(t_f)| - n$.*

Proof: Clearly when X is a satisfying assignment, plugging $|S_a(t_f)| = n + m$ (by Lemma 7) into the above formula gives m, which is indeed the total number of clauses. Suppose X is not a satisfying assignment. In the proof of corollary 1, we see that $S_a(t_0) \subseteq S_a(t_f)$, where $S_a(t_0)$ represents X. Since X is a truth-value assignment, $|S_a(t_0)| = n$. In the proof of Lemma 7, we see that no clique contains more than one vertex from a particular S_i. Putting these together, $S_a(t_f)$ contains $|S_a(t_f)| - |S_a(t_0)| = |S_a(t_f)| - n$ vertices from distinct S_i. S_i represents the clause c_i and from our transformation, if $S_a(t_f)$ contains a vertex from S_i, then X $(= S_a(t_0))$ satisfies c_i. It follows that X satisfies $|S_a(t_f)| - n$ clauses $\Box$.

By Cor. 2, we can find how many clauses a truth-value assignment satisfies, from **size** information of the network stable state. More specifically, n_P contains the desired value if we set $\theta = n$. We can also answer the question **Does this particular assignment satisfy at least k clauses?** by a **size-threshold** decision (setting $\theta = n + k$) on the network stable state.

Corollary 3 *The network tests if X satisfies C (or the number, k, of clauses that X satisfies) in $k \leq m$ steps.*

Proof: $|S_a(t_0)| = n$ and $G^c_\phi[S_a(t_0)]$ is complete. Since the largest clique in G has size $n + m$, $|S_a(t_f)| \leq n + m$. By Thm 1, network reaches stable state in $\leq m$ steps. From this final state, the yes/no answer is computed by our extensions in $O(1)$ time $\Box$.

Consider the question of testing whether a truth-value assignment satisfies at least one clause. This can be answered by a **state tester** computation (also follows from Corr. 2) as follows. As before, set the initial state of the network to the truth-value assignment. In the first cycle, the network can only switch a unit ON, if at all. If it does, then the answer is **yes** else **no**.

7 Finite Regular Languages Problem

Let α denote a *-free regular expression. It follows that the language $L_\alpha \subset \Sigma^*$ which it denotes is finite. We will show that a graph G_α can be constructed so that there is a unique maximal clique in G_α for every string $s \in L_\alpha$. This will allow us, as we shall see, to answer certain questions about L_α. The reader might find it useful to look ahead to the Fig 3 examples and familiarise themselves with the transformation before coming back to read the following paragraphs.

We will see that G_α, obtained from our transformation, is a $(k+1)$-partite graph, where k is the length of the largest string in L_α. The $(k+1)^{th}$ partition will be a special partition called V_s , containing vertices that will encode string lengths in the maximal cliques. We use $s(v)$ to denote the string length that $v \in V_s$ encodes. The name of a non-V_s vertex is an ordered pair (a,b) where $a \in \Sigma$ is the symbol it represents and p is the position in the corresponding string that a occurs in. p also represents the partition number in G_α that this vertex lies in. We let $p_{G_\alpha}(v)$ denote the partition number of a vertex in G_α as well as the partition component value in its name.

Our goal is to have the unique maximal clique in G_α corresponding to a string $s \in L_\alpha$ be such that it contains one vertex from each of the first $|s|$ partitions and a vertex representing $|s|$ from the V_s partition. A maximal clique with such a structure will be called a maximal *string-clique*, because it can be directly interpreted as a string. We require it to be maximal because what we really want is a correspondence between strings in L_α and network stable states, which are maximal (cliques or f-uniformly-dense sets) only. Moreover, non-maximal *string-cliques* would correspond only to left substrings of strings in L_α. Consider strings $s', s \in L_\alpha$ and s is a left substring of s' (that is $s' = ss$", $|s"| \geq 1$). Without the V_s vertex denoting $|s|$ in the *string-clique* of s, it would be a strict subset of the *string-clique* of s' and hence unable to represent s. With the V_s vertex representing $|s|$, the *string-clique* of s is guaranteed to stay maximal. In general, we must prevent the possibility of the *string-clique* of s becoming a strict subset of any spurious clique (not necessarily a *string-clique*) that might develop during the transformation. We will term this as the *non-containment* condition to denote both this and the earlier case.

Let us see what facts and conditions are sufficient to guarantee the *non-containment* condition.

Fact 1 *No maximal clique in G_α can contain more than one vertex from V_s.*
This will follow directly from the transformation definition below. No where in the definition (base case, $\cup^p$ definition or the algorithmic definition of ".") will an edge between V_s vertices exist or get added).

In conjunction with Fact 1, the following conditions are collectively sufficient (as will be shown in the proof of Thm 2) to guarantee the *non-containment* condition. We term them *proper-size* and *partition-preserving* respectively and define them below.

Definition 4 *G_α is a* proper-size *graph if for every vertex $v \in V_s$, v has the property of non-adjacency to every vertex in partitions numbered greater than $s(v)$.*
In other words, $p(u) \leq s(v) \quad \forall u \in N(v)$

Definition 5 *G_α is a* partition-preserving *graph if all non-V_s vertices with the same partition component value in their name also lie in the same partition. In other words, no two vertices with the same partition component value are adjacent.*

Transformation

We now give an inductive, algorithmic definition of the transformation from α to G_α.

1. If $\alpha \in \Sigma$, then $G_\alpha = (\{(\alpha,1),1\},\{\{(\alpha,1),1\}\})$. where '1' is the vertex named '1' in $V_s(G_\alpha)$.
2. $G_{\alpha\cup\beta} = G_\alpha \cup^p G_\beta$. where '$\cup^p$' is the graph union operator with the further stipulation that the vertices must be kept in the same partitions after the union. Specifically, if G_α, G_β are i-partite, j-partite respectively, and without loss of generality $i \leq j$, then

$$\begin{aligned} V_l(G_{\alpha \cup \beta}) &= V_l(G_\alpha) \cup V_l(G_\beta);\ 1 \leq l \leq i \\ &= V_l(G_\beta);\ i < l \leq j \end{aligned}$$

   where $V_l(G)$ is the set of vertices in the l^{th} partition of G. Also,

$$E(G_{\alpha\cup\beta}) = E(G_\alpha) \cup E(G_\beta)$$

   We note that our vertex naming convention, which explicitly includes partition numbers, will automatically meet the above requirements if we place the vertices in the partition corresponding to their partition number, after union.

3. $G_{\alpha.\beta}$ is defined by algorithmic construction from G_α and G_β. It is similar to but also quite different than the conventional join of G_α and G_β.
   Algo $G_{\alpha.\beta}$

   (a) $G_{\alpha\ .\ \beta} \leftarrow \phi$

   (b) for $v \in V_s(G_\alpha)$ do

      i. $G_{\alpha v} \leftarrow G[N_{G_\alpha}(v)]$

      ii. $G_\beta\text{copy} \leftarrow$ G_β shifted to the right to start from partition $s(v) + 1$ (partition component value of non V_s vertices adjusted) and the size vertices (V_s) revalued
      Specifically:
      $\forall u \in V(G_\beta) - V_s(G_\beta) : p_{G_\beta\text{copy}}(u) \leftarrow p_{G_\beta}(u) + s(v)$
      $\forall u \in V_s(G_\beta) : s_{G_\beta\text{copy}}(u) \leftarrow s_{G_\beta}(u) + s(v)$

      iii. $G_{\alpha\beta v} \leftarrow G_{\alpha v}\ X\ G_\beta\text{copy}$

      iv. $G_{\alpha\ .\ \beta} \leftarrow G_{\alpha\ .\ \beta} \cup^p G_{\alpha\beta v}$

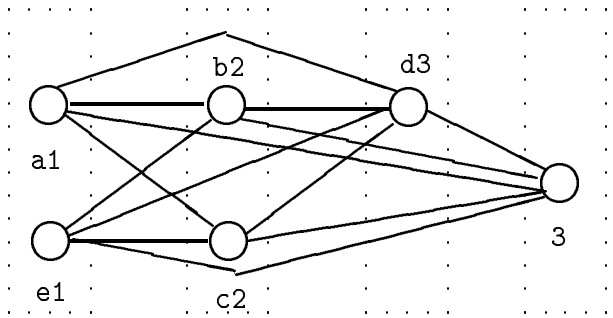
   (c) end for

where X is the complete two graph join.

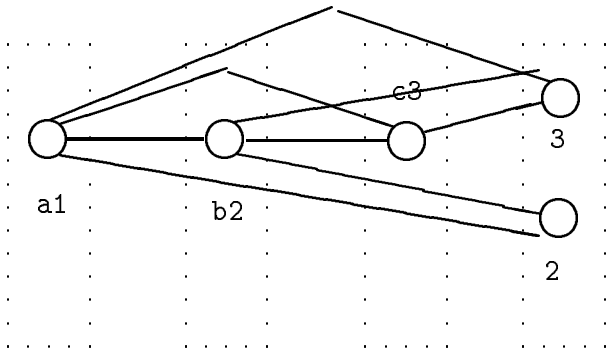
Examples

The following examples show the graphs, obtained by the above transformation, for the corresponding regular expressions.

(a.b U a.c U e.b U e.c).d



a.b U a.b.c



(a.b U c).d

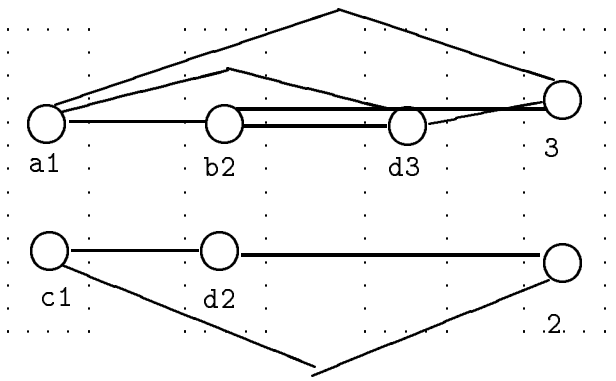


Fig 3 - Regular expressions and their transformation graphs

We digress a bit to note that this transformation is somewhat inefficient, in that extra copies of G_β are being generated in constructing $G_{\alpha.\beta}$. Our main goal was to keep the correspondence between strings and maximal cliques direct (via string-cliques), which would not be possible with a more efficient construction of $G_{\alpha.\beta}$, such as by taking

the conventional join of G_α and G_β . Another reason is that our transformation is independent of α, for the same L_α . For example, G_α is the same whether $\alpha = (a.b \cup c).d$ or $\alpha = a.b.d \cup c.d$. By a conventional join, G_α would be different in the two cases.

It would be nice if our transformation were to guarantee a 1-1 simple correspondence between strings in L_α and maximal cliques in G_α. This would allow us to use the network as a finite regular language acceptor (to test if a string belongs to the language) via the network **state tester** computations. We show that for languages that have *fully-factored* regular expressions, such a 1-1 correspondence exists.

Definition 6 *A* **fully-factored** *regular expression is a collection of clauses separated by the '.' operator, each clause is a union of symbols in* Σ.

`Example :  (a U b U c).(d U e).(a U d)`

Lemma 8 *If* α *is a* **fully-factored** *regular expression for a language* L_α*, then there is a 1-1 correspondence between the maximal cliques in* G_α *and strings in* L_α.

Proof: We will informally sketch the proof. Let $|\alpha|$ denote the number of clauses (also the length of every string) in α. G_α is a $(|\alpha|+1)$-partite graph, the first $|\alpha|$ partitions are complete (completely-connected), each partition corresponding to a clause in α. The $(|\alpha|+1)^{th}$ partition is a V_s partition containing only one vertex v ($s(v) = |\alpha|$) adjacent to all vertices in G_α. We let α_i denote the i^{th} clause in α and $|\alpha_i|$ denote the number of alphabet symbols (in Σ) in α_i. It is easy to see that $\Pi_{i=1}^{|\alpha|}|\alpha_i|$ denotes the number of strings in L_α as well as the number of maximal cliques (all of size $|\alpha| + 1$ and all are *string-cliques*) in G_α, which establishes the 1-1 correspondence $\Box$.
Fig 3a shows G_α for the example (a.b U a.c U e.b U e.c).d which is the same G_α for the fully-factored expression (a U e).(b U c).d for the same language.

Unfortunately, for arbitrary finite regular languages to possess such a 1-1 correspondence is an unrealistic expectation. A simple counter-example shows otherwise. Let $\alpha = c.a.r \cup c.o.n \cup f.o.r$. It is easy to check that $\{c_1, o_2, r_3, 3\}$ is a maximal clique of G_α, which does not correspond to any string of L_α.

One hope is that, in general, there is a unique maximal *string-clique* in G_α for every string in L_α. For then, we can at least use the network as a partial rejector in the following way. If the set of vertices denoting a string is not a maximal clique of G_α, then it is not in L_α. If it is a maximal clique, then we can say nothing about membership, since it might be a spurious one. The test for maximal clique can be done via the network **state tester** computations. To formalise our hope, we say that G_α, obtained by transformation from α, is a *string-preserving* graph if there is a unique maximal *string-clique* in G_α for every string $s \in L_\alpha$. We now state and prove the following theorem.

Theorem 2 G_α*, obtained by transformation from* α *(a regular expression for a finite regular language* L_α*), is a* string-preserving *graph.*

Proof: By induction on the structure of α.
Basis: $\alpha \in \Sigma$.
$G_\alpha = (\{(\alpha,1),1\},\{\{(\alpha,1),1\}\})$. G_α contains only one maximal clique $\{(\alpha,1),1\}$ which is a maximal *string-clique* for the only string α in L_α. Hence G_α is *string-preserving*. G_α is also, trivially, *proper-size* and *partition-preserving*.
Induction Hypothesis: Let G_α (G_β) be graphs representing regular expressions α (β) of languages L_α (L_β). Let G_α, G_β be *string-preserving, proper-size* and *partition-preserving.*
We have to show that

1. $G_{\alpha \cup \beta}$ is *string-preserving, proper-size* and *partition-preserving.*
2. $G_{\alpha . \beta}$ is *string-preserving, proper-size* and *partition-preserving.*

We have added the additional requirements of *proper-size* and *partition-preserving* in the induction hypothesis because we will need them in the induction step. Specifically, we will first prove that $G_{\alpha \cup \beta}(G_{\alpha . \beta})$ is *proper-size* and *partition-preserving* and use this to show that $G_{\alpha \cup \beta}(G_{\alpha . \beta})$ is *string-preserving.*

1. We first show that $G_{\alpha \cup \beta}$ is *proper-size*. Suppose not. $\exists v \in V_s(G_{\alpha \cup \beta})$ such that v is adjacent to some vertex u for which $p_{G_{\alpha \cup \beta}}(u) > s(v)$. Since $G_{\alpha \cup \beta}$ is a union of G_α, G_β which preserves partitions, (u,v) is an edge either in G_α or G_β. Assume that (u,v) is an edge in G_α. Hence $p_{G_\alpha}(u) > s(v)$. But this contradicts I.H. that G_α is proper-size.
   Hence $G_{\alpha \cup \beta}$ is *proper-size.*

   We now show that $G_{\alpha \cup \beta}$ is *partition-preserving*. Suppose not. Then $G_{\alpha \cup \beta}$ has some pair u, v of vertices in the same partition i that are adjacent. Again, since $G_{\alpha \cup \beta}$ is a union of G_α , G_β which preserves partitions,

(u, v) is an edge either in G_α or G_β in the same partition. But this contradicts I.H. that G_α and G_β are *partition-preserving.*
Hence $G_{\alpha \cup \beta}$ is *partition-preserving.*

We now show that $G_{\alpha \cup \beta}$ is *string-preserving.* Consider $s \in L_{\alpha \cup \beta}$. Then $s \in L_\alpha$ or $s \in L_\beta$ or $s \in$ both. By I.H., either G_α or G_β or both contain a unique maximal string-clique C associated with s. Since $G_{\alpha \cup \beta}$ is a union of G_α , G_β , C is a string-clique in $G_{\alpha \cup \beta}$. We have to show that it is maximal because only then would it correspond to s. Suppose $C' \supset C$ is a larger clique in $G_{\alpha \cup \beta}$. C' contains the vertex $v \in V_s$ such that $s(v) = |C| - 1$.

(a) By Remark 1, C' cannot contain a second vertex from the V_s partition.

(b) Since $G_{\alpha \cup \beta}$ is *partition-preserving*, C' cannot contain a second vertex from partitions 1 to $|C| - 1$.

(c) Since $G_{\alpha \cup \beta}$ is *proper-size*, C' cannot contain any vertex from partitions numbered greater than $|C| - 1$.

Hence $C' = C$. Hence C is maximal string-clique in $G_{\alpha \cup \beta}$ corresponding to $s \in L_{\alpha \cup \beta}$. Hence $G_{\alpha \cup \beta}$ is *string-preserving.*

2. We first show that $G_{\alpha.\beta}$ is *proper-size.* Suppose not. $\exists v \in V_s(G_{\alpha.\beta})$ such that v is adjacent to some vertex u for which

$$p_{G_{\alpha.\beta}}(u) > s(v) \tag{8}$$

Let lv be value of loop variable in the algorithm iteration in which edge (u, v) is added to $G_{\alpha\beta lv}$ in algorithm step b-iii.

(a) The edge came from $G_\beta$copy . But then the $y \in V_s(G_\beta)$ such that

$$s(y) = s(v) - s(lv) \tag{9}$$

(that is after reversing the effect caused by shifting) is adjacent to u and

$$p_{G_\beta}(u) = p_{G_{\alpha.\beta}}(u) - s(lv) \tag{10}$$

From (8),(9),(10), $p_{G_\beta}(u) > s(y)$ which contradicts I.H. that G_β is *proper-size.*

(b) The edge is added in algorithm step b-iii, when constructing the join of $G_{\alpha lv}$ and $G_\beta$copy . But this means u is in $G_{\alpha lv}$ and v in $G_\beta$copy . From the algorithm, $p_{G_\alpha}(u) = p_{G_{\alpha.\beta}}(u)$ and hence by our assumption,

$$p_{G_\alpha}(u) > s(v) \tag{11}$$

Now since $v \in G_\beta$copy and $G_\beta$copy has at least one partition, excluding V_s , by algorithm step b-ii

$$s(v) > s(lv) \tag{12}$$

From (11),(12), $p_{G_\alpha}(u) > s(lv)$ where $u \in V(G_\alpha) - V_s(G_\alpha)$ and $lv \in V_s(G_\alpha)$. Also, since $u \in V(G_{\alpha lv})$, u is adjacent to lv. But this contradicts I.H. that G_α is *proper-size.*

Hence $G_{\alpha.\beta}$ is *proper-size.*

We now show that $G_{\alpha.\beta}$ is *partition-preserving.* Suppose not. $\exists u, v \in$ same partition in $G_{\alpha.\beta}$ such that they are adjacent. Let lv be value of loop variable in the algorithm iteration in which edge (u, v) is added to $G_{\alpha\beta lv}$ in algorithm step b-iii.

(a) The edge came from $G_{\alpha lv}$. Since $E(G_{\alpha lv}) \subset E(G_\alpha)$, this would contradict I.H. that G_α is *partition-preserving.*

(b) The edge came from $G_\beta$copy . Since $G_\beta$copy is simply a shifted version of G_β (partitions shifted), this edge is in G_β , which contradicts I.H. that G_β is partition-preserving.

(c) The edge is added in algorithm step b-iii, when constructing the join of $G_{\alpha lv}$ and $G_\beta$copy . But this is not possible since all vertices in $G_\beta$copy have larger partition number than any vertex in $G_{\alpha lv}$ and their join cannot add an edge in the same partition.

Hence $G_{\alpha.\beta}$ is *partition-preserving.*

We now have to show that $G_{\alpha.\beta}$ is *string-preserving.* Consider $s \in L_{\alpha.\beta}$. Then $s = s_\alpha\, s_\beta$ where $s_\alpha \in L_\alpha$, $s_\beta \in L_\beta$. By I.H., G_α (G_β) has a maximal string-clique $C_\alpha(C_\beta)$ corresponding to $s_\alpha(s_\beta)$. Let C_α^- denote the clique $C_\alpha - V_s(G_\alpha)$. Let lv be value of loop variable in algorithm iteration such that $s(lv) = |C_\alpha^-|$. In algorithm step b-ii, $G_\beta$copy contains the string-clique C'_β, formed from C_β with vertices shifted to the right (partition components renamed) by $|C_\alpha^-|$ partitions. In algorithm step b-iii, the join of $G_{\alpha lv}$ and $G_\beta$copy ensures that $C_\alpha^- \cup C'_\beta$ is a string-clique of $G_{\alpha\beta lv}$ and since step b-iv involves basically a graph union of $G_{\alpha.\beta}$ and $G_{\alpha\beta lv}$ which preserves partitions, $C_\alpha^- \cup C'_\beta$ remains a string-clique in $G_{\alpha.\beta}$. Since the only subsequent changes to $G_{\alpha.\beta}$ will be in algorithm step b-iv where $G_{\alpha\beta lv}$ will be repeatedly added to $G_{\alpha.\beta}$ via graph union, $C_\alpha^- \cup C'_\beta$ will remain a string-clique after the algorithm terminates. We have to show that it is maximal then. Suppose $C' \supset C$ is a larger clique in $G_{\alpha.\beta}$ after the algorithm terminates.

(a) By Remark 1, C' cannot contain a second vertex from the V_s partition.

(b) Since $G_{\alpha.\beta}$ is *partition-preserving*, C' cannot contain a second vertex from partitions 1 to $|C| - 1$.

(c) Since $G_{\alpha.\beta}$ is *proper-size*, C' cannot contain any vertex from partitions numbered greater than $|C| - 1$.

Hence $C' = C$. Hence C is a maximal string-clique in $G_{\alpha.\beta}$ corresponding to $s \in L_{\alpha.\beta}$. Hence $G_{\alpha.\beta}$ is *string-preserving* □.

Finite Regular Language Problem on Network - Concluding Remarks

We assume G_α has been obtained from α (programmatically or otherwise). We store G_α in the network as usual, by presenting its edges, which takes $|E(G_\alpha)|$ presentations. We discuss one problem for which we informally argue that the network is a near-optimal device, in space and time complexity. Let $||\alpha||$ denote the number of alphabet (in Σ) symbol occurences in α. Consider α to be a *fully-factored* regular expression with an exponential number (in terms of $||\alpha||$) of strings in L_α . The network can build an acceptor in low-order polynomial time because this only entails storing $|E(G_\alpha)|$ $[O(||\alpha||^2)]$ edges. We ignore, for the moment, the transformation from α to G_α which can also be done in polynomial time. The storage requirement is determined by the number of network connections (weights) which is also $|E(G_\alpha)|$ $[O(||\alpha||^2)]$, since G_α is dense. The network makes a decision (accept, reject input string) in one network cycle.

We also note that as an *acceptor* is only one of the uses the network can be put to. If the absence of *spurious* stable states can be guaranteed (as would be required even for an *acceptor*), like for languages (L_α) having *fully-factored*) regular expressions, the network can also be used to transform some string not in L_α to a string in L_α (via **transducer computations**, that is according to network energy-descent rule), to find a large string in L_α (via **large stable state computation**) etc.

We conclude this topic by noting that further investigations are required of properties of G_α , for L_α that don't have *fully-factored* regular expressions. One important issue is of showing 1-1 correspondence (of strings, maximal cliques) for restricted cases of such languages. In this regard, we have been trying to prove that 1-1 correspondence is preserved under concatenation, which, if true, would be quite an important result. At this time, we leave it as a conjecture in the following form.

Conjecture 1 *Let α, β be regular expressions such that there is a 1-1 correspondence between strings in L_α (L_β) and maximal cliques in G_α (G_β). Then there is a 1-1 correspondence between strings in $L_{\alpha.\beta}$ and maximal cliques in $G_{\alpha.\beta}$.*

8 Conclusion

We have shown that the Hopfield-style network can be viewed as a Maximal-Clique graph machine and that this view is a very productive one. Specifically, we have seen that the network stable states are maximal cliques and, in general, f-uniformly-dense sets. We have categorised network computations into different kinds. Categorisation has provided us with a nice problem independent framework to study the different computations and their properties and also abstract out unnecessary detail when describing which question can be answered via which computation. We have outlined a general framework for mapping problems onto the network. We

have presented the SAT transformation in detail and shown that (and how) the network can test if a truth-value assignment to a CNF formula satisfies it. We have presented a novel transformation of finite regular languages to the Clique problem and shown how the network can be used as a partial rejector in general and as a total acceptor for languages that have *fully-factored* regular expressions.

In a more general sense, we feel that our paper makes contributions along two fronts. Firstly, it motivates the study of transformations from different problems to the Clique problem, suggesting that cliques have important properties for information storage and computation. Secondly, it shows that such problems can be translated into constraint satisfaction (energy minimization) problems on a simple one-layer network that is quite feasible to implement in hardware. It is very important to hold the dual views of network computations as being energy-minimization and as finding maximal cliques by a greedy algorithm, because this permits a cross-fertilization of ideas. For example, it might be profitable to explore using pseudo-global energy-minimization techniques like simulated annealing [5, 6] or mean-field annealing (see [4]) as the energy-dynamics rule of our network, while still retaining maximal cliques (or f-uniformly-dense vertex sets) as stable states. This would correspond to finding pseudo-optimally large cliques (or f-uniformly-dense sets) in graphs. That the maximum clique problem is NP-Complete also directly implies that the **problem of finding the global energy minimum in our network is NP-Complete**. This follows because the global energy minimum of our network corresponds to the maximum clique in the underlying graph G_n. This means that the maximum clique of any graph can be found by storing it in the network and finding the global energy minimum of the network, from which it follows that the latter problem is NP-Complete. A more formal proof will appear elsewhere.

We would like to thank Dr. Kenneth Regan for reviewing this paper and for previous and ensuing discussions and suggestions that have improved its quality. We would also like to thank Dr. Xinhe for discussions.

References

[1] Shun-Ichi Amari (1989). Characteristics of Sparsely Encoded Associative Memory. *Neural Networks, Volume 2, Number 6, 1989*

[2] J.A. Bondy, U.S.R Murty (1976). Graph Theory with Applications, *North-Holland.*

[3] Garey Michael R., Johnson David S. (1979). Computers and Intractability - A guide to the theory of NP-completeness.

[4] G. Bilbro, R. Mann, T. Miller, W. Snyder, D. Van den Bout, M. White (1989). Optimization by mean field annealing. *Advances In Neural Information Processing Systems 1, pp 91-98, Morgan Kaufmann.*

[5] S. Kirkpatrick, C.D. Gelatt, Jr., and M.P. Vecchi (1983). Optimization by simulated annealing. *Science 220:671-680.*

[6] Ackley David H., Hinton Geoffrey E., Sejnowski Terrence J. (1985). A learning algorithm for Boltzmann machines. *Cognitive Science 9:147-169.*

[7] Hopfield, J.J. (1982). Neural networks and physical systems with emergent computational properties. *Proceedings of the National Academy of Sciences, USA, 79, 2554-2558.*

[8] Hopfield, J.J. (1984). Neurons with graded response have collective computational properties like those of two-state neurons *Proceedings of the National Academy of Sciences, USA, 81, 3088-3092.*

[9] Arun Jagota and Y.T. Hung (1990). A neural lexicon in a Hopfield-style network, *Proceedings of the International Joint Conference on Neural Networks, 1990.*

[10] Arun Jagota (1990). A new Hopfield-style network for content-addressable memories. *Technical Report 90-02, Dept Of Computer Science, State University Of New York At Buffalo, NY 14260.*

[11] Arun Jagota (1990). Applying a Hopfield-style network to Degraded Text Recognition, *Proceedings of the International Joint Conference on Neural Networks, San Diego, 1990.*

[12] Arun Jagota (1990). Applying a Hopfield-style network to Degraded Printed Text Restoration, To appear in *Proceedings of the 3rd Conference on NN and PDP - Indiana-Purdue University.*

[13] Arun Jagota (1990). What can be done with about 1/4 million words in a Hopfield-style network? *In review.*

[14] Godbeer Gail H., Lipscomb John, Luby Michael (1988). On the Computational Complexity of Finding Stable State Vectors in Connectionist Models (Hopfield Nets). *Technical Report 208/88, Dept Of Computer Science, University Of Toronto, Ontario Canada M5S 1A4.*