Parallel Logical Inference and Energy Minimization

Dana H. Ballard

Computer Science Department
The University of Rochester
Rochester, NY 14627

Abstract

The inference capabilities of humans suggest that they might be using algorithms with high degrees of parallelism. This paper develops a completely parallel connectionist inference mechanism. The mechanism handles obvious inferences, where each clause is only used once, but may be extendable to harder cases. The main contribution of this paper is to show formally that some inference can be reduced to an energy minimization problem in a way that is potentially useful.

1. Motivation

This paper explores the possibility that a restricted class of inferences in first order logic can be made with a very large knowledge base using only a parallel relaxation algorithm. The main restriction is on the infrastructure of the logical formulae, but not on the *number* of such formulae. The relaxation algorithm requires that problems be formulated as the intersection of (possibly huge) numbers of local constraints represented in networks.

The formulation of the algorithm is in terms of a connectionist network [Feldman and Ballard, 1982]. Recently a class of algorithms for solving problems has emerged that has particularly economical formulations in terms of massively parallel architectures that use large networks of interconnected processors [Kirkpatrick et al., 1983; Hopfield, 1984; Hopfield and Tank, 1985; Hinton and Sejnowski, 1983]. For a survey, see [Feldman, 1985]. By "massively parallel," we mean that the number of processors is on the order of the size of the knowledge base. This class of algorithms has been described as "energy minimization" owing to analogies between the algorithms and models of physical processes.

The key contribution of this paper is to show that some theorem proving can be described in terms of this formalism. Formally, there is an algorithm to minimize the "energy" functional E given by

$$E = -\sum_{i}\sum_{i} w_{ij}s_i s_j + \theta_i s_i \qquad (2)$$

where s_i is the binary state of a unit, either *on* (0) or *off* (1), w_{ij} is a real number that describes a particular constraint, θ_i is a threshold (also a real number) [Hopfield, 1982], and the weights are symmetric, i.e., $w_{ij} = w_{ji}$. The energy functional has a related constraint network where there is a node for each state, and the weights are associated with the ends of arcs in the network and the thresholds are associated with each state. The technical status of algorithms for minimizing E are discussed in [Ballard, 1986]. This paper shows that the weights and thresholds can be chosen to encode theorem-proving problems.

A controversial aspect of our formulation is that it is not guaranteed to work in every case. For many scientific applications, an inference mechanism that handles only the simpler cases, and fails in many cases, might not be useful. In particular, this is true for research directed toward the development of mechanical theorem provers, that handle cases that are difficult for humans. However, for models of human inference mechanisms, this may not be the case. Our conjecture is that: *facts that humans can infer in a few hundred milliseconds have an efficient time solution on a parallel machine.* The key assumption we are willing to make is that the kind of inferences that humans can do quickly may be restricted cases of a general inference mechanism. One reason for this assumption is that the human inference mechanism can be viewed as one component of several in a perception-action process. For example, in our model, if the inference mechanism fails to identify a visual object, one of the options available is to move closer and gather more data. Thus our goal is to develop an inference mechanism that allows many inferences to be made in parallel with the understanding that it may also fail in many cases.

A general method of theorem proving is refutation. In other words, to prove $S \vdash W$ where S and W are sets of clauses, one attempts to show that $S \cup \neg W$ is unsatisfiable. One way of doing this is to use *resolution* [Robinson, 1965]. Our approach uses the unit resolution paradigm but has three important restrictions: (1) *clauses may be used only once in each proof*; (2) *the knowledge base must be logically consistent*; and (3) *the method uses a large network that must be preconnected.*

The overall organization of our parallel inference is shown in Figure 1. The process has three overall phases that are carried out sequentially:

Step 0: *Logical Consistency Constraints.* The first part has the goal of activating a logically consistent set of constraints. This is the focus of other research, and we assume that the enterprise is successful.

Step 1: *Filter Constraints.* Constraints derived from the clause structures [Sickel, 1976; Kowalski, 1975] deactivate parts of the network that are inconsistent.

Step 2: *Resolution.* The last part of the algorithm uses a second filtering technique based on unit resolution. In this phase, parts of the network are deactivated if they correspond to pairs of clauses that would resolve where one of the pair is a unit clause. If the entire network can be deactivated in this way, a proof has been found; otherwise, the result is inconclusive.

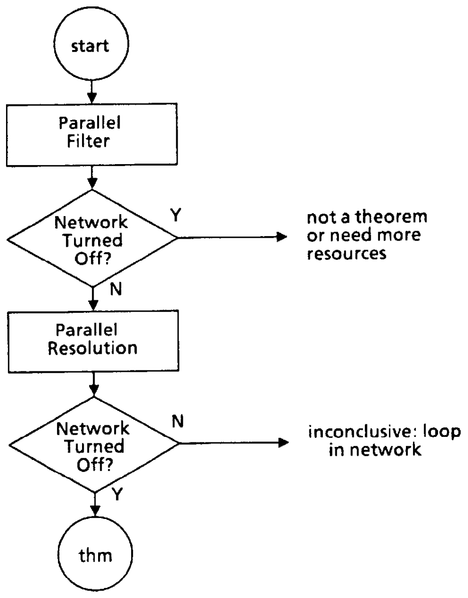


Figure 1.

2. The Constraint Network

The constraint network has five sets of nodes: (1) C, the set of clause nodes; (2) L, the set of predicate letters and their complements; (3) F, the set of clause fragments; (4) U, the set of unifications between fragments; and (5) B, the set of substitutions. In any set of clauses there will be one clause node, $c \in C$ for each clause in the set. There will be one clause fragment node $f \in F$ for each predicate letter and its complement that are mentioned in different clauses. There will be a separate unification node $u \in U$ for each possible resolution between complementary literals in different clauses. Finally there will be a substitution node $b \in B$ for each possible substitution involving a unification. For example, in the following set $\{S \cup \neg W\} = \{c_1: P(x,a), c_2: \neg P(b,y)\}$,

$$\begin{aligned} C &= \{c_1, c_2\} \\ L &= \{P, \neg P\} \\ F &= \{(c_1, P), (c_2, \neg P)\} \\ U &= \{((c_1, P)\,(c_2, \neg P))\} \\ B &= \{xb, ya\} \end{aligned} \tag{1}$$

There are six different kinds of constraints: (1) a predicate letter constraint; (2) a clause-predicate substitution constraint; (3) a clause constraint; (4) unification constraints; (5) a substitution constraint; and (6) a unit clause constraint. The first five capture constraints implied by the clause syntax and unification. The sixth is an additional constraint which anticipates the unit resolution proof procedure. Table 1 summarizes these constraints, which are described in detail below. All of the constraints can be obtained directly from the clause syntax.

The Clause Constraint. The clause constraint captures the notion that a clause can only be part of the solution if all of its fragments have viable bindings. Thus the fragments must be connected to the node in a way that exhibits conjunctive behavior. Table 1a shows an example of a clause with n fragments.

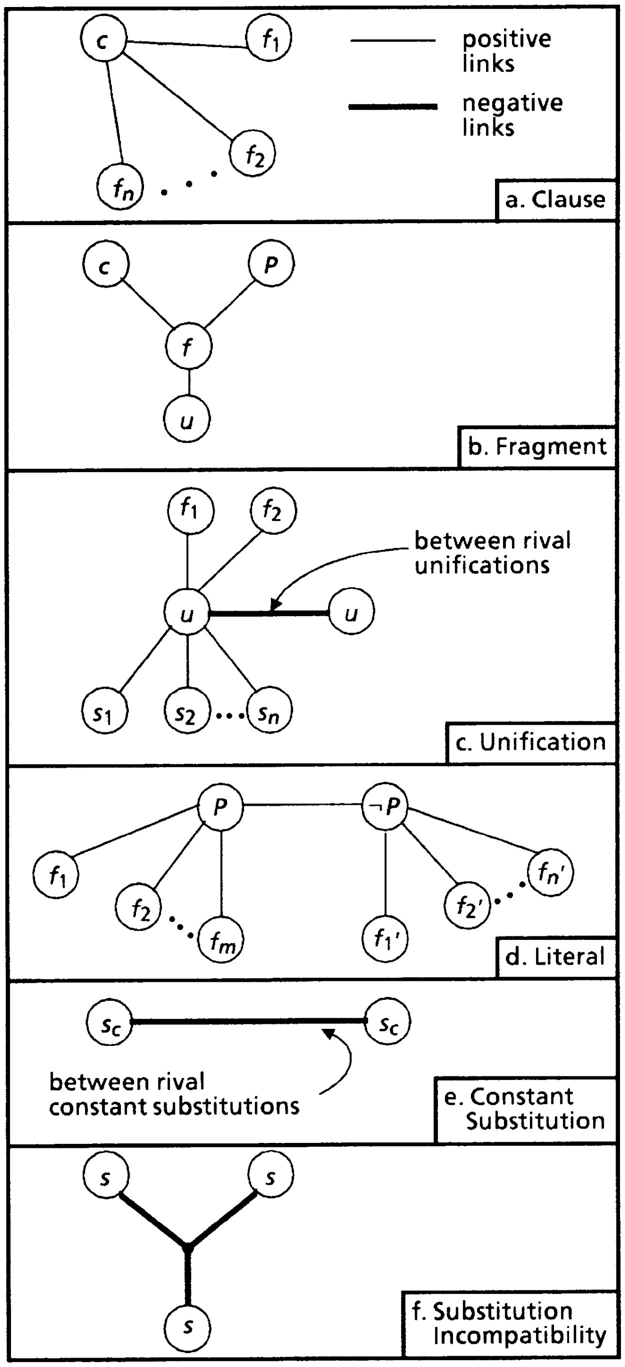
The Clause-Predicate-Substitution Constraint (or Clause Fragment Constraint). This constraint is derived from the clauses in a straightforward way. Each clause may be decomposed into triples consisting of: (clause symbol, predicate letter, substitution). For example, c_1: $P(x)Q(a)$ may be decomposed into (c_1, P, u_1) and (c_1, Q, u_2) where u_1 and u_2 are appropriate substitutions (these will be discussed further as part of the substitution constraints). In the filter network, there are a set of clause fragment nodes F, one for each triple. A clause fragment node f is connected to each node in the triple with positively weighted connections as shown in Table 1b.

Unification Constraints. Complementary literals in different clauses that can unify constrain the network in two important ways. These can be captured by positively weighted links to unification nodes. Any two clause fragment nodes that are connected to complementary literals are linked to a unique unification node. That node also has links to substitution nodes for each of the substitutions that result from the unification. Thus in the example given by Equation (2), one unification node was linked to the two appropriate fragment nodes and the two appropriate substitution nodes.

The Literal Constraint. The literal constraint is derived from propositional logic. If in the set of clauses, a literal appears without its complement or vice versa, then that clause can be pruned from the solution. In terms of the filter network, this constraint is easily expressed as a positively weighted arc between different nodes representing predicate letters, as shown in Table 1d.

The Substitution Constraints. The substitution constraints limit possible bindings between terms. The clauses that can potentially resolve constrain possible substitutions, and these possible substitutions are realized by a set of substitution nodes S. Substitutions that are incompatible are connected by negatively weighted connections. For example, in the set of clauses $\neg P(a,b)$, $P(x,y)Q(y,z)$, $\neg Q(c,d)$, $\neg P(a,c)$, the possible substitutions are xa, yb, yc, and zd. Of these,

Table 1: Summary of Constraints



compatible pairs are: (xa, yb), (xa, yc) and (yc, zd), and there is one incompatible pair: (yc, yd). This example is simple and does not capture all the constraints possible in unification. At least one other is necessary. This relates bindings between constants and variables. If a variable is substituted with a constant and another variable is substituted with a different constant, then the two variables cannot be substituted with each other. These constraints are summarized below:

x, y : var ; c, d const
(xc, xy, yd) are incompatible
(xc, xd) are incompatible

In the network there are potentially $N_v(N_c + N_v)$ nodes where N_c is the number of constants and N_v is the number of variables. Thus the above constraints are connected between all relevant groupings. Representative network fragments are shown in Table 1e and 1f. These constraints can be extended to handle some function symbol constraints, but the development herein will assume only constants and variables.

The substitution constraint can be easily implemented if we allow multiplicative effects. Multiplicative effects cause a node to be turned off if any one of the inputs becomes zero. A way of handling this problem that also adheres to the symmetric weight requirement needed for convergence is to use ternary nodes. Table 1f shows a multiplicative connection in terms of symmetric ternary connections, and Figure 2 shows the detailed connections.

The final constraint to be added is a *single use constraint*. This constraint is not dictated by the clause syntax but anticipates a unit clause inference rule. The constraint is simply this: *literals in different clauses that can resolve with the same literal in a given clause have mutually inhibitory connections.* To clarify this, consider the example $c_1{:}P(x)$, $c_2{:}\neg P(a)$, $c_3{:}\neg P(b)$. Either c_2 or c_3 could resolve with c_1. However, to force the network to "choose" one or the other, a negatively weighted arc is introduced between the corresponding fragments (c_2, $\neg P$) and (c_3, $\neg P$).

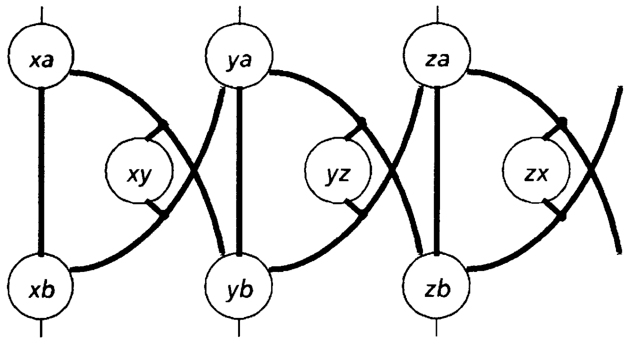


Figure 2: The substitution network showing only consistency connections for constants $\{a, b\}$ and variables $\{x, y, z\}$.

3. Choosing the Weights

In the previous sections it was shown that the formulae of first order predicate calculus and the inference rules of a proof producer (viz. resolution) can be uniquely expressed in terms of a network. Such a network has a particularly simple form, consisting only of undirected links between nodes. To relate this network to Equation (2), we add real-valued weights at the ends of each arc and real-valued thresholds to each node.

Owing to space limitations, we will omit the proof that the weights and thresholds that we specify guarantee the desired behavior. This can be found in [Ballard, 1986]. Instead, the basic ideas will be outlined using Figure 3. The figure shows a hypothetical plot of the energy of the network as a function of the states. The two most important constraints are those that guarantee that: (1) each literal has exactly one complement; and (2) the substitutions are consistent. These are to be weighted so that violating them incurs very large penalties. Remaining states that satisfy them are defined to be *admissible states*. These are weighted so that the more of the network structure that can be turned on, the better. The global minimum of the network is simply the admissible state with the most clause structure.

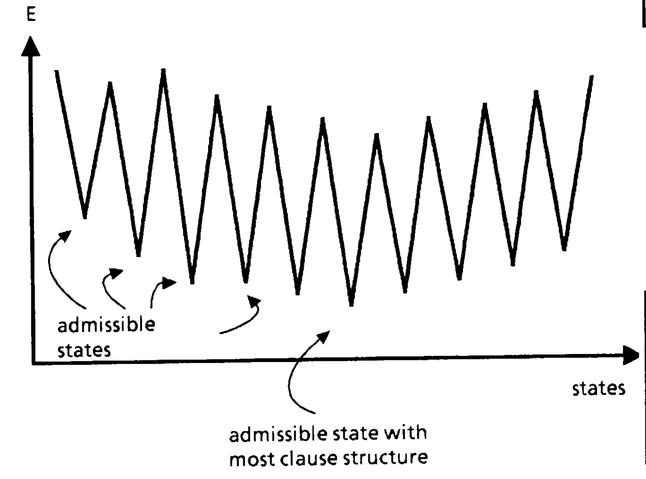


Figure 3: Hypothetical plot of energy vs. states showing desirable network properties.

The complete summary of weights is shown in Table 2. From this table, it is not intuitive how these weights function. To help overcome this problem we classify the nodes into three types, AND, OR, and AND/OR, as shown in Figure 4. All of the node types will have negative thresholds: AND nodes will have a threshold that must be less (in absolute value) than the sum of all the arc weights but greater than any subset of arc weights; OR nodes will have a threshold that is less than any of the arc weights; an AND-OR node may be constructed if the sum of the OR arcs is equal to the value of the weights on the AND arcs, assuming that the latter are all identical.

Owing to the various constraints, some unification nodes or substitution nodes may be forced off. Under this circumstance, just the clause structure that depends on these should be turned off. Building the networks out of AND, OR, and AND-OR nodes guarantees that this happens, since the energy efficiency of the desired state can be shown to be optimal by direct calculation [Ballard, 1986]. Thus parts that are consistent according to the logical syntax form local energy minima.

Table 2: (a) Weights and (b) thresholds for filtering stage. ($\alpha = (1/2)(\theta + \varepsilon)$; m = number of instances of a predicate P; n = number of instances of a complementary predicate $\neg P$; N_c = number of literals in the clause.)

	c	f	P	$\neg P$	u	s
c		α				
f	α		α/m	α/n	$\frac{\alpha}{mn}$	
P		α/m		α		
$\neg P$		α/n	α			
u		$\frac{\alpha}{mn}$				α
s					α	

	S_{cv}	$S_{cv}S_{vv}$
S_{cv}	$-k^*$	$-k^*$
$S_{cv}S_{vv}$	$-k^*$	$-k^*$

* between appropriate pairs and triples

(a)

(b)

	c	$f(P)$	P	$\neg P$	$f(\neg P)$	u	s
Threshold	$(-N_c\theta)/2$	$\alpha(1 + 1/m + 1/mn)$	$\alpha(1 + 1/m)$	$\alpha(1 + 1/n)$	$\alpha(1 + 1/n + 1/mn)$	- sum of input, ignoring ε's	0
Node Type	AND	AND/OR	AND/OR	AND/OR	AND/OR	AND/OR	AND

4. How It Works

Consider the set of clauses $\{P(a), P(b), P(c), \neg P(x), \neg P(y)\}$. The network for this example is shown in Figure 5, with the weights and thresholds chosen according to Table 2 with $\theta = 1$. To understand the example, note that if there are m instances of a literal P and n of its complement in different clauses, then:

1) for the P node there will be m OR connections;

2) for the $\neg P$ node there will be n OR connections;

3) for each fragment node related to P there will be n OR connections;

4) for each fragment node related to $\neg P$ there will be m OR connections.

At the beginning of stage two the filtering process has pruned the network so that only portions with consistent substitutions are left in the *on* state. Therefore in the resolution process there is no need to recheck the substitutions since they are known to be consistent. For this reason, the substitution network may be ignored. It is removed from the computation and the threshold on the unification node is adjusted accordingly.

The thresholds on the clauses are now lowered (remember that they are negative) to the point where they are each greater in absolute value than the weights from the clause fragment link. This means that it is now profitable to turn the singleton clause nodes off. Ideally, this should cause other nodes to be turned off as well. If the entire network can be turned off, a proof by unit resolution exists. [Ballard, 1986] elaborates on this point. The one case where the network cannot be turned off is where there is a *loop*, e.g., $c_1 fufc_2 fufc_1 fufc_1$. The energy of a loop is negative, whereas the energy of all nodes in the off state is zero, so it is never profitable to turn off the nodes in a loop. The main change to the weights is to leave out the substitution network and make each clause node an OR node with threshold $\theta/2$.

5. Summary and Conclusions

The implementation of the first order logic constraints results in two coupled networks: (1) a clause network that represents the clause syntax; and (2) a binding network that represents the relationships between terms in different clauses. The method for resolving bindings, unification, can be as complex as the entire inference mechanism. Thus for the purposes of computing efficiently, we would expect the actual bindings in the knowledge base to have a simple structure.

At the outset, the possibility of reusing clauses was ruled out, but there are some limited cases that can be handled. To see the necessity of reusing clauses, consider $\{S \cup \neg W\} = \{c_1{:}P(a),\ c_2{:}P(b),\ c_3{:}\neg P(x)Q(x),\ c_4{:}\neg Q(a)\neg Q(b)\}$. This can be handled by resolution in a straightforward way. The resolution tree is: $((c_1, c_3), ((c_2, c_3), c_4))$. However, note that c_3 appears twice. The consequence of this is that since the unification constraints do not allow xa and xb simultaneously, the network will not pass the filter test. To handle this case we note that both possibilities for c_3 involve constant bindings. Thus we can resolve this by making two copies of c_3: $\neg P(a)Q(a)$ and $\neg P(b)Q(b)$. Once this is done, the inference mechanism will find the proof. However, this is not a very elegant strategy. As noted by Josh Tenenberg, if c_3 were $\neg P(x)Q(y)$ one would need four copies, $\neg P(a)Q(a)$, $\neg P(a)Q(b)$, $\neg P(b)Q(a)$, $\neg P(b)Q(b)$, and in general a clause with k literals, each with a different variable, would generate k^{N_c} possibilities, where N_c is the number of constants.

The main intent of this paper has been to provide a new look at formal inference mechanisms from the standpoint of performance. Our contention is that models that do not have a parallel implementation are unlikely candidates for models of human inference.

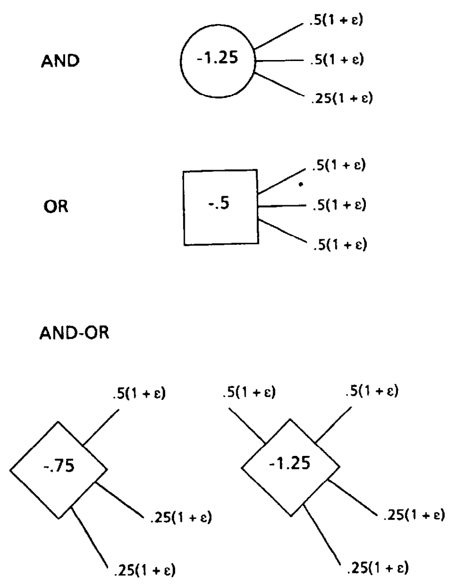


Figure 4: AND, OR, and AND/OR nodes. Numbers inside tokens are thresholds (that appear next to the tokens in Figure 5). Epsilon is a small positive number required for correctness [Ballard, 1986].

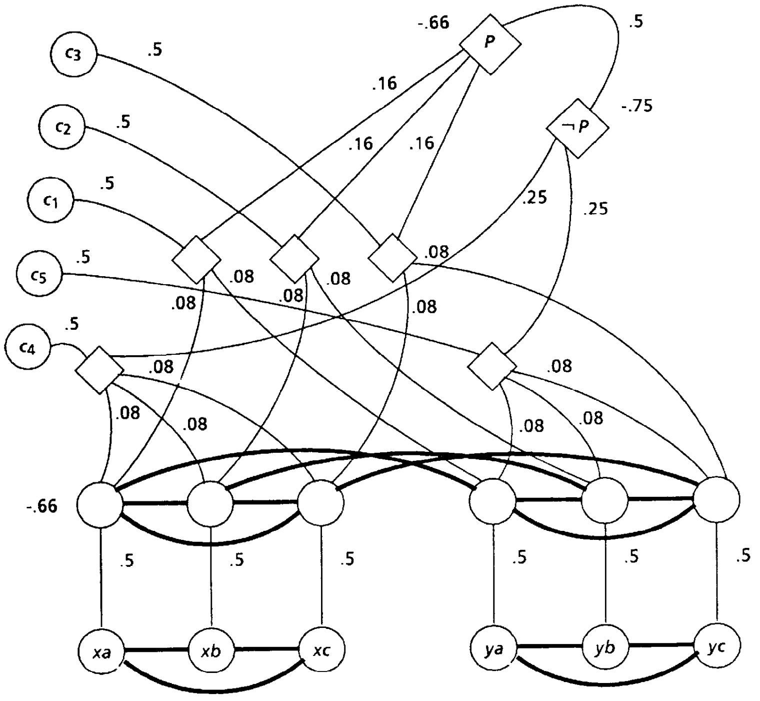


Figure 5: Network for $\{P(a), P(b), P(c), \neg P(x), \neg P(y)\}$. Epsilons omitted.

This realization may prove catalytic for approaches that try to unify the complementary goals of competence and performance. The technical contribution of this paper is in the detailed specification of a network inference mechanism. The network runs in parallel and can handle obvious inferences in first order logic. We have described how the problem of proving theorems or making inferences of the form $S \vdash W$ can be reduced to two sequential network optimization problems. The first checks the formulae for the constraints defined in Section 2 and settles in a state where each literal instance has a unique complement. The second minimization is equivalent to a unit resolution proof. If a proof by unit resolution exists, it will be manifested as a global energy minimum. While no computer simulations have been done, the proofs provided in [Ballard, 1986] show that the problem reduction works. The stable states of the two optimization problems are just those desired.

The reduction of theorem proving to energy minimization is an important step, but much additional work needs to be done. At present, the one convergence proof available [Geman and Geman, 1984] does not provide an encouraging estimate on the running time of such algorithms, and simulations that have been done give varying results for different problems. Algorithms that require global minima are still comparable to conventional approximation techniques [Johnson et al., 1984]. However, studies of the Traveling Salesman problem using analog processing units have shown that good solutions can be found quickly [Hopfield and Tank, 1985]. These encouraging results are a source for some optimism: perhaps in the case of inferences, if a measure of good, average performance is used instead of the classical best-, worst-case performance, these algorithms will exhibit behavior closer to the Traveling Salesman result.

Acknowledgements

Pat Hayes was enormously helpful when these ideas were in formative stages. The artful figures and carefully formatted text are the work of Peggy Meeker. Peggy also edited many earlier drafts of this report. I am grateful to John Mellor-Crummey for pointing out several areas that needed improvement in the original design. Jerry Feldman, Josh Tenenberg, Leo Hartman, and Jay Weber each made several helpful suggestions on earlier drafts. This work was supported in part by the National Science Foundation under Grant DCR-8405720.

References

Ballard, D.H., "Parallel logical inference and energy minimization," TR 142, Computer Science Dept., U. Rochester, March 1986.

Davis, M., "Obvious logical inferences," Courant Institute, 1983.

Fahlman, S.E., D.S. Touretzky, and W. van Roggen, "Cancellation in a parallel semantic network," *Proc., 7th Intl. Joint Conf. on Artificial Intelligence*, Vancouver, BC, Canada, August 1981.

Feldman, J.A., "Energy and the behavior of connectionist models," TR 155, Computer Science Dept., U. Rochester, November 1985.

Feldman, J.A. and D.H. Ballard, "Connectionist models and their properties," *Cognitive Science 6*, 205-254, 1982.

Freuder, E.C., "Synthesizing constraint expressions," *CACM 21*, 11, 958-965, November 1978.

Garey, M.R. and D.S. Johnson. *Computers and Intractability*. W.H. Freeman, 1979.

Geman, S. and D. Geman, "Stochastic relaxation, Gibbs distributions, and the Bayesian restoration of images," *IEEE Trans. PAMI 6*, 6, 721-741, November 1984.

Henschen, L.J., "A tutorial on resolution," *IEEE Trans. Computers C-25*, 8, 770-772, August 1976.

Hinton, G.E. and T.J. Sejnowski, "Optimal perceptual inference," *Proc., IEEE Computer Vision and Pattern Recognition Conf.*, 448-453, Washington, DC, 1983.

Hopfield, J.J., "Neural networks and physical systems with emergent collective computational abilities," *Proc., National Academy of Sciences USA 79*, 2554-2558, 1982.

Hopfield, J.J., "Neurons with graded response have collective computational properties like those of two-state neurons," *Proc., Natl. Acad. Sci. 81*, 3088-3092, May 1984.

Hopfield, J.J. and D.W. Tank, "'Neural' computation of decisions in optimization problems," to appear, *Biological Cybernetics*, 1985.

Johnson et al., Lecture Notes, seminar presentation at Yale, 1984.

Kirkpatrick, S., C.D. Gelatt, and M.P. Vecchi, "Optimization by simulated annealing," *Science 220*, 4598, 671-680, 1983.

Kowalski, R., "A proof procedure using connection graphs," *JACM 22*, 4, 572-595, 1975.

Nilsson, N.J. *Principles of Artificial Intelligence*. Palo Alto, CA: Tioga Pub. Co., 1980.

Nilsson, N.J. *Problem-Solving Methods in Artificial Intelligence*. New York: McGraw Hill Book Co., 1971.

Robinson, J.A., "A machine-oriented logic based on the resolution principle," *JACM 12*, 1, 23-41, January 1965.

Sickel, S., "A search technique for clause interconectivity graphs," *IEEE Trans. Computers C-25*, 8, 823-835, August 1976.