Parallel Hardware for Constraint Satisfaction

Michael J. Swain & Paul R. Cooper

Department of Computer Science
University of Rochester
Rochester, NY 14627

Abstract

A parallel implementation of constraint satisfaction by arc consistency is presented. The implementation is constructed of standard digital hardware elements, used in a very fine-grained, massively parallel style. As an example of how to specialize the design, a parallel implementation for solving graph isomorphism with arc consistency is also given.

Complexity analyses are given for both circuits. Worst case running time for the algorithms turns out to be linear in the number of variables n and labels a, $O(an)$, and if the I/O must be serial, it will dominate the computation time. Fine-grained parallelism trades off time complexity for space complexity, but the number of gates required is only $O(a^2n^2)$.

1 Introduction

Constraint satisfaction is an important technique used in the solution of many artificial intelligence problems. Since the original applications such as Waltz filtering [Waltz, 1975], an essential aspect of most constraint satisfaction algorithms has been their cooperative or parallel nature (eg. [Davis and Rosenfeld, 1981]). While the parallel spreading activation nature of constraint propagation has been adopted whole-heartedly in specific applications such as connectionist relaxation [Feldman and Ballard, 1982; Hinton *et al.*, 1984], some of the most complete and generally useful formal results analyze sequential algorithms [Mackworth and Freuder, 1985; Mohr and Henderson, 1986]. Generating a formal analysis of one recent connectionist implementation of discrete relaxation [Cooper, 1988] inspired us to design a massively parallel implementation of the classic, more generally applicable arc consistency constraint satisfaction algorithm [Mackworth, 1977; Hummel and Zucker, 1983; Mohr and Henderson, 1986]. The implementation is constructed of standard digital hardware elements, used in a very fine-grained, massively parallel style. The resulting circuit is thus an obvious candidate for fabrication in VLSI, and is thus similar to the work of Mead [1987].

The paper also provides a parallel hardware implementation of the arc consistency algorithm for a specific application - labelled graph matching. Such matching by constraint propagation and relaxation is often used in visual recognition systems [Cooper, 1988; Kitchen and Rosenfeld, 1979].

Complexity analyses are given for both circuits. Worst case running time for the algorithms turns out to be linear in the number of variables n and labels a, $O(an)$, and if the I/O must be serial, it will dominate the computation time. Fine-grained parallelism trades off time complexity for space complexity, but the number of gates required is only $O(a^2n^2)$.

2 Constraint Satisfaction

In this section, we review constraint satisfaction as classically formulated [Mackworth, 1977; Hummel and Zucker, 1983; Mohr and Henderson, 1986]. A constraint satisfaction problem (CSP) is defined as follows: Given a set of n variables each with an associated domain and a set of constraining relations each involving a subset of the variables, find all possible n-tuples such that each n-tuple is an instantiation of the n variables satisfying the relations. We consider only those CSPs in which the domains are discrete, finite sets and the relations are unary and binary.

A k-consistency algorithm removes all inconsistencies involving all subsets of size k of the n variables. In particular, node and arc consistency algorithms detect and eliminate inconsistencies involving $k = 1$ and 2 variables, respectively.

More specifically, a typical arc consistency problem consists of a set of variables, a set of possible labels for the variables, a unary predicate, and a binary predicate with an associated constraint graph. For each i of the n variables, the unary predicate $P_i(x)$ defines the list of allowable labels x taken from the domain of the variables. For each pair of variables (i, j) in the constraint graph the binary predicate $Q_{ij}(x, y)$ defines the list of allowable label pairs (x, y). To compute the n-tuples which satisfy the overall problem requires that the local constraints are propagated among the variables and arcs.

Mohr and Henderson [1986] specify one such constraint satisfaction algorithm for arc consistency: AC-4. They show that the complexity of AC-4 is $O(a^2e)$, where a is the number of labels (or the cardinality of the domain), and e is the number of edges in the constraint graph associated with $Q_{ij}(x, y)$.

Hummel and Zucker describe a parallel version of the arc consistency algorithm as follows (using Mackworth's notation).

Arc consistency is accomplished by means of the *label discarding rule:* discard a label x at a node i if there exists a neighbor j of i such that *every* label y currently assigned to j is incompatible with x at i, that is, $\neg Q_{ij}(x, y)$ for all $y \in D_j$. The label discarding rule is applied in parallel at each node, until limiting label sets are obtained.

Others such as Waltz [1975] and Hinton [1977] have also suggested implementing constraint satisfaction in parallel. Wang, Gu and Henderson [1987] have designed and implemented a systolic architecture for arc consistency.

3 A Hardware Implementation

The Arc Consistency (AC) chip consists of two arrays of JK flip-flops and suitable amounts of combinational circuitry. The most important part of the design is the representation for the two constraint tables $P_i(x)$ and $Q_{ij}(x,y)$. In the massively parallel connectionist design style, we adopt the unit/value principle, and assign one memory element to represent every possible value of $P_i(x)$ and $Q_{ij}(x,y)$. (As will be seen, JK flip-flops are used as the memory elements because of their convenient reset characteristics). To allow the hardware to compute any arc consistency problem, the two arrays must be able to represent any given $P_i(x)$ and $Q_{ij}(x,y)$ of sizes bounded by n and a.

The first (node) array consists of an flip-flops we call $u(i,x)$ which are initialized to $P_i(x)$. That is, if x is a valid label at node i, then the the flip-flop $u(i,x)$ is initialized to **on**. Thus initially at least, the flip-flops which are **on** all correspond to labellings of a node which are valid considering only the local (unary) constraint at that node. Note that all flip-flops are initialized. The final answer to the computation (which labels are arc consistent at each node) will be contained in this array at the end of the computation.

The second (arc) array consists of $a^2n(n-1)$ flip-flops we designate $v(i,j,x,y)$ which are initialized to conform to the arc constraint table $Q_{ij}(x,y)$. Note that the table $Q_{ij}(x,y)$ can designate three things. If $Q_{ij}(x,y) = 1$, then the arc (i,j) is present in the constraint graph and the label pair (x,y) is a valid labelling of the pair. If $Q_{ij}(x,y) = 0$, the arc (i,j) is again present in the constraint graph, but the label pair (i,j) in not allowed on that arc. But $Q_{ij}(x,y)$ might also just not be present in the arc constraint table, which indicates that there is no consistency constraint between nodes i and j. To account for the fact that $Q_{ij}(x,y)$ might be incomplete, $v(i,j,x,y)$ is initialized as follows: if i is adjacent to j in the constraint graph

$$v(i,j,x,y) = Q_{ij}(x,y)$$

otherwise

$$v(i,j,x,y) = 1$$

Note that the arc array is static; it does not change throughout the computation.

The basic structure of the two arrays of flip-flops is shown in Figure 1.

It remains only to develop combinational circuitry which implements the label discarding rule - ie. that causes the flip-flop representing the label x at node i to be reset to zero if it becomes inconsistent. The combinational circuitry is thus designed so that the K (reset) input of the JK-flip-flop $u(i,x)$ receives the value:

$$reset(u(i,x)) = \neg \bigwedge_{j=1,j\neq i}^{n} \bigvee_{y=1}^{a} (u(j,y) \wedge v(i,j,x,y))$$

The J input of each JK-flip-flop is tied to 0. A partial circuit diagram for this equation is given in Figure 2. This

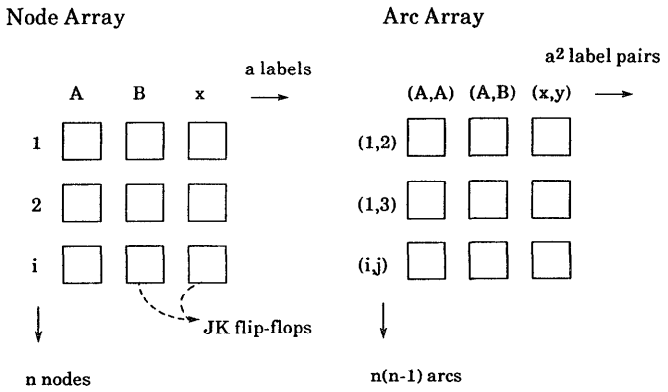

Figure 1: Unary and Binary Constraint Tables

figure show the reset circuitry for one flip-flop in the node table $u(i,x)$. In the figure, the entire node table is present, but only the part of the arc table $v(i,j,x,y)$ useful for this node is drawn. An analogous circuit for each node completes the whole circuit.

To interpret the equation and circuit, consider first the inner term $u(j,y) \wedge v(i,j,x,y)$ for a particular case of $u(i,x)$. The fact that $v(i,j,x,y)$ is true tells us that there is an arc between i and j, and (x,y) is a consistent label pair for this arc. We already know that $u(i,x)$ is true; **and**ing with $u(j,y)$ checks that the other end of the arc has a valid label. Point A on the circuit diagram in Figure 2 shows where this term is computed.

At this point, as far as node i is concerned, x is a label consistent with node neighbor j's label y. The $\bigvee_{y=1}^{a}$ simply ensures that at least *one* label y on neighboring node j is consistent. This function has been computed after the **or** gate at point B in Figure 2

Label x on node i is thus consistent with its neighbor j. But what about node i's other neighbors? The $\bigwedge_{j=1,j\neq i}^{n}$ ensures that there is arc consistency among *all* node i's neighbor's. The **and** gate at C in Figure 2 ensures this.

If the signal is **on** at point C, that means that label x is consistent for node i - therefore, the flip-flop need **not** be reset. Thus the not gate.

To reverse the analysis, if some node j does **not** have a consistent labelling, then at point B, the signal will be **off**. The **and** will fail, so the signal at C will also be 0, and then the **not** gate will cause flip-flop $u(i,x)$ to be reset.

3.1 Correctness

To begin with, recall that we are interested in discarding labels, an operation which corresponds to resetting **on** flip-flops to 0. Furthermore, since the J input of each JK-flip-flop in the node array is tied to zero, the flip-flops can only ever be reset to 0, never set. Once they are **off** they must stay off, so the whole process is clearly monotonic. Therefore, all we need to show for correctness is to show that the network correctly applies the label discarding rule. If the network discards labels when they should be discarded, and does not discard them when the should be kept, then it implements the label discarding rule correctly.

The label discarding rule can be formally expressed as

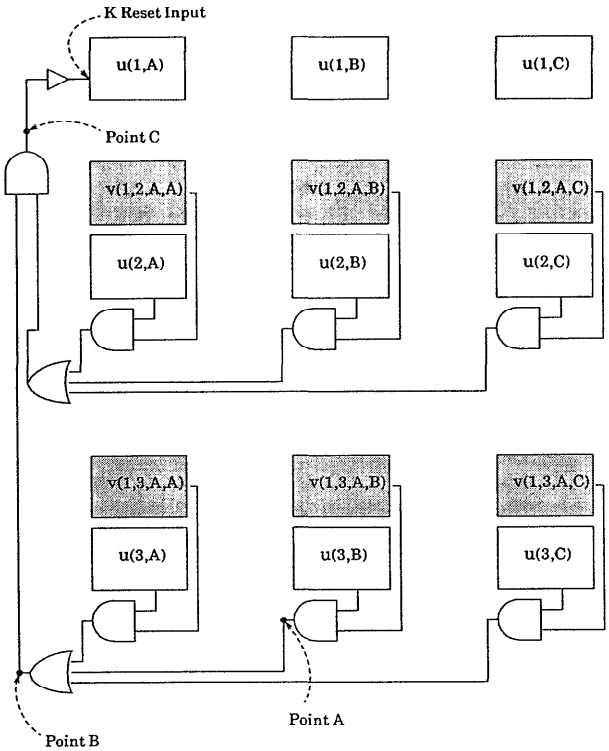

Figure 2: Partial Circuit Diagram for the AC Chip

follows:

$$\exists j(j \neq i)\forall y[u(j,y) \wedge v(i,j,x,y) = 0]$$

But this expression is equivalent to

$$\bigwedge_{j=1,j\neq i}^{n} \bigvee_{y=1}^{a} (u(j,y) \wedge v(i,j,x,y)) = 0$$

or

$$\neg \bigwedge_{j=1,j\neq i}^{n} \bigvee_{y=1}^{a} (u(j,y) \wedge v(i,j,x,y)) = 1$$

which is just the condition under which (i,x) is reset. Therefore, the network correctly discards labels when it should. The converse follows from negating the above equations.

3.2 Complexity

The circuit requires an JK-flip-flops for the node array, and $a^2n(n-1)$ flip-flops for the arc array. From Figure 2 we see that there is an **and** gate for every flip-flop in the arc array, so $a^2n(n-1)$ 2-input **and** gates are required for this purpose. For each of the an flip-flops in the *node* array there is $n-1$ **or** gates required, each taking a inputs - a total of $an(n-1)$ **or** gates. Finally, there are an **and** and **not** gates (**nand** gates), each taking $n-1$ inputs. There are also $O(a^2n^2)$ wires.

The worst case time complexity of the network occurs when only one JK-flip-flop is free to reset at a time. So if propagation through the **and** and **or** gates is considered instantaneous, the worst case time complexity is an. If a logarithmic time cost is assigned to the large fan-in **and** and **or** gates the worst case time complexity is $O(a\log(a)n\log(n))$.

Kasif [1986] has shown that solving constraint satisfaction with arc consistency is log-space complete for P (the class of polynomial time deterministic sequential algorithms). This suggests that it is likely no poly-log time algorithm will be found, so $O(na)$ time is liable to be near the minimum achievable with polynomially-many processors [Swain and Cooper, 1988].

Note that if the node and arc arrays must be initialized serially, loading them takes more time ($O(a^2n^2)$ steps) than executing the algorithm. For almost all applications of constraint satisfaction the binary predicate $Q_{ij}(x,y)$ can be specified with less than $O(a^2n^2)$ information, and so instead of the arc array a circuit could be built that supplies the correct values to the **and** gates without needing so many memory elements to fill. An application in which this is true is graph matching, which we describe in the next section.

4 Graph Matching

Graph matching can be defined as a constraint satisfaction problem. General graph matching requires k-consistency [Freuder, 1978] (and is NP-complete, in fact). With just arc consistency, a restricted yet still interesting class of graphs may be matched. Furthermore, the effectiveness of matching graphs by constraint satisfaction with only arc consistency can be enhanced if the graphs are labelled. This kind of restricted matching of labelled graphs is particularly suited to the visual indexing problem [Cooper, 1988]. In this problem, labelled graphs are used to represent structurally composed objects. The constraint satisfaction process is used only to filter recognition candidates, and the few graphs not discriminable with the limited power of arc consistency can be addressed in other ways.

If labelled graph matching is framed as a constraint satisfaction process, the unary constraint is that the labels of corresponding vertices be the same. The binary (arc) constraint ensures that the connectedness between pairs of corresponding vertices be the same. In other words, if there is an edge between 2 vertices in one graph, there better be an edge between the corresponding vertices in the other graph. In this section, we describe without loss of generality the matching of undirected graphs.

So, for the graph matching problem:

$$P_i(x) = (\text{label}(i) = \text{label}(x))$$

and

$$Q_{ij}(x,y) = (\text{adjacent}(i,j) = \text{adjacent}(x,y))$$

For the graph matching problem the number of possible labels equals the number of vertices so $a = n$.

There are some modifications we can make to the general arc consistency circuit that are to our advantage for this particular application.

Constraint Table Computation by Special-Purpose Circuitry

One modification is to replace the arc array by a circuit designed as follows. Construct two arrays of

$$\binom{n}{2} = \frac{n(n-1)}{2}$$

flip-flops representing adjacent(i,j) and adjacent(x,y) respectively. Note that these are adjacencies in the input graphs, not in the constraint graph. For all possible $(i,j)(x,y)$ pairs, wire one flip-flop from the (i,j) array and one flip-flop from the (x,y) array to a gate computing the equality function $\overline{xor}$. Then the output of the $((i,j),(x,y))$'th gate represents $Q_{ij}(x,y)$. Then the network will have only $O(n^2)$ flip-flops to load prior to the computation.

Analogous special purpose circuitry to compute $P_i(x)$ from the vertex/label sets of each graph can easily be imagined as well. In the case, the equality gate must check equality of the labels, so is likely comparing more than just a single bit.

In any case, it is clear that actually computing the constraint tables $P_i(x)$ and $Q_{ij}(x,y)$ may be a significant part of the overall computation. In many specialized cases, it is clearly possible to actually build parallel circuitry to assist in computing the constraint tables, rather than serially computing the predicates beforehand and then loading them into the parallel hardware.

Symmetric Matching

Graph matching need not be simply isomorphism, as many vision applications emphasize [Shapiro and Haralick, 1981]. If we restrict ourselves to pure isomorphism however, the graph matching problem is symmetric. In terms of the constraint satisfaction formulation, the symmetry means that the vertices of graph A have to be possible labels for graph B as well as vice versa. Therefore for a flip-flop (i,x) to stay, one may require it to be consistent regarding x as the label and i as the vertex and vice versa. So in addition to the **and-or** network described for the general constraint satisfaction problem the graph matching circuit has a complementary network in the opposite direction. The two circuits are **and**ed together before the inverter at the K input of the JK latch. Together these circuits compute

$$\neg\left(\left(\bigwedge_{j=1,j\neq i}^{n}\bigvee_{y=1}^{n}(v(j,y)\wedge Q_{ij}(x,y))\right)\wedge\left(\bigwedge_{y=1,y\neq x}^{n}\bigvee_{j=1}^{n}(v(j,y)\wedge Q_{ij}(x,y))\right)\right)$$

The circuit which implements this equation finds all possible labelings that are pairwise consistent both for matching graph A to graph B and for matching graph B to graph A.

4.1 Complexity

If no special purpose circuitry is used to compute$P_i(x)$ and it is input as a table of an or n^2 entries (in this case, $a = n$), then the complexity is as follows. The node array requires n^2 JK-flip-flops. The reduced arrays representing the input graphs require a total of $n(n-1)$ flip-flops. To replace the arc array, there are $n^2(n-1)^2$ $\overline{\textbf{xor}}$ gates. Analogous to the earlier design $n(n-1)$ 2-input **and** gates are required, $n^2(n-1)$ **or** gates, and n^2 **nand** gates. There are $O(n^4)$ wires, as for the general constraint satisfaction network.

The worst case time complexity for the graph matching network is the same as for the constraint satisfaction network, $O(n^2)$ ignoring propagation time and $O(n^2\log^2 n)$ taking it into account. Loading and unloading the network takes $O(n^2)$ sequential time, and so does not affect the worst-case performance of the network. Since the expected time of the constraint satisfaction step could be much less than the worst-case performance, sequential loading and unloading is still likely to be the performance bottleneck.

4.2 Comparison with Connectionist Network

Cooper [1988] gives a connectionist network design for solving the same labelled graph matching problem addressed here. Interestingly, although the two networks were developed from completely different heritages, and for different reasons, they are remarkably alike. In particular, the central aspect of both designs - the representation of the unary and binary constraint predicates as completely filled-in tables - is exactly the same. This reflects the adoption of the unit/value design principle, which is useful for obtaining a very high degree of parallelism, no matter what the primitive units of computation. In fact, it is straightforward to realize our current design as a connectionist network with simple unit functions such as **and** and **or**. We describe a connectionist simulation of this network implementation in Swain and Cooper [1988].

Unlike the chip design, a connectionist network is never intended to interface with sequential processes, so the input constraint tables can be filled by parallel spreading activation. As a result, the I/O bottleneck does not occur. Of course, if the digital network were to receive parallel input, the same would be true.

5 Discussion and Conclusions

The utility of constraint satisfaction methods in the solution of many AI problems suggests that efficient implementations might be widely useful. Furthermore, constraint satisfaction methods have an obvious parallel character.

In this paper, we have given a massively parallel design which provably implements one classic constraint satisfaction algorithm. Our implementation thus inherits the correctness characteristics of the original formulation. We have also shown how this design is easily specializable for particular problems. This specialization process provides a desirable alternative to designing and proving a new parallel network for each particular problem.

As might be expected, the highly parallel implementation runs very fast. Although worst case running time is linear in the number of variables and labels, it is more reasonable to expect that the network runs in a small constant number of time steps. Overall, if I/O time is not included, the performance of the network can be expected

to be much better than that of the best sequential implementations.

For sufficiently small problems it would be straightforward to construct our arc consistency chip, even for the general case. If, however, the parallel machine is forced to interface with sequential processes, the run-time complexity becomes similar to that expected from standard sequential implementations of arc consistency. This I/O bottleneck can be overcome by supplying parallel input or by specializing the chip to solve a particular problem, as we showed in the graph matching example.

Specialization also helps address the issues that arise in solving larger problems. It is easy to see that the limits of current VLSI technology arise quickly when $O(n^2a^2)$ space is required. But in some current work, we have discovered that it is possible to reduce these resource requirements by as much as three or four orders of magnitude for some classes of problems, even using the same basic design[Swain and Cooper, 1988].

Constructing special-purpose hardware is effective in environments where classes of problem instances are well-understood and repeat frequently. (For example, a robot vision system designed for industrial application). An alternative to using special purpose hardware is to implement a parallel algorithm on a general purpose parallel computer, such as the Connection Machine. This alternative becomes especially interesting if it yields run-time complexity comparable to our current design. We have been investigating this possibility [Swain and Cooper, 1988], as have other researchers [Henderson and Samal, 1988].

Acknowledgements

This work was supported by a Canadian NSERC postgraduate scholarship, by the Air Force Systems Command, Rome Air Development Center, Griffis Air Force Base, New York 13441-15700 and the Air Force Office of Scientific Research, Bolling AFB, DC 20332 under Contract No. F30602-85-C-0008. The latter contract support the Northeast Artificial Intelligence Consortium (NAIC). We thank the Xerox Corporation University Grants Program for providing equipment used in the preparation of this paper.

References

[Cooper, 1988] Paul R. Cooper. Structure recognition by connectionist relaxation: Formal analysis. In *Proceedings of the Canadian Artificial Intelligence Conference: CSCSI-88*, Edmonton, Alberta, June 1988.

[Davis and Rosenfeld, 1981] L. S. Davis and A. Rosenfeld. Cooperating processes for low-level vision: A survey. *Artificial Intelligence*, 17:245–263, 1981.

[Feldman and Ballard, 1982] J. A. Feldman and D. H. Ballard. Connectionist models and their properties. *Cognitive Science*, 6:205–254, 1982.

[Freuder, 1978] Eugene C. Freuder. Synthesizing constraint expressions. *Communications of the ACM*, 21:958–966, 1978.

[Gu *et al.*, 1987] Jun Gu, Wei Wang, and Thomas C. Henderson. A parallel architecture for discrete relaxation algorithm. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-9:816–831, 1987.

[Henderson and Samal, 1988] Tom Henderson and Ashok Samal. Parallel consistent labeling algorithms. *International Journal of Parallel Algorithms*, 1988.

[Hinton *et al.*, 1984] G. E. Hinton, T. J. Sejnowski, and D. H. Ackley. Boltzmann machines: Constraint satisfaction networks that learn. Technical Report CMU-CS-84-119, Department of Computer Science, Carnegie-Mellon University, 1984.

[Hinton, 1977] Geoffrey E. Hinton. *Relaxation and Its Role in Vision*. PhD thesis, University of Edinburgh, 1977.

[Hummel and Zucker, 1983] Robert A. Hummel and Steven W. Zucker. On the foundations of relaxation labeling processes. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-5:267–287, 1983.

[Kasif, 1986] Simon Kasif. On the parallel complexity of some constraint satisfaction problems. In *Proceedings of AAAI-86*, pages 349–353, 1986.

[Kitchen and Rosenfeld, 1979] Les Kitchen and Azriel Rosenfeld. Discrete relaxation for matching relational structures. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-9:869–874, 1979.

[Mackworth and Freuder, 1985] Alan K. Mackworth and Eugene C. Freuder. The complexity of some polynomial network consistency algorithms for constraint satisfaction problems. *Artificial Intelligence*, 25:65–74, 1985.

[Mackworth, 1977] Alan K. Mackworth. Consistency in networks of relations. *Artificial Intelligence*, 8:99–118, 1977.

[Mead, 1987] Carver Mead. Silicon models of neural computation. In *Proceedings of the First IEEE International Conference on Neural Networks, Vol. 1*, pages 93–106, 1987.

[Mohr and Henderson, 1986] R. Mohr and T. C. Henderson. Arc and path consistency revisited. *Artificial Intelligence*, 28:225–233, 1986.

[Shapiro and Haralick, 1981] Linda G. Shapiro and Robert M. Haralick. Structural descriptions and inexact matching. *IEEE-PAMI*, 3(5), 1981.

[Swain and Cooper, 1988] Michael J. Swain and Paul R. Cooper. Parallel constraint satisfaction. Technical Report TR 255, Department of Computer Science, University of Rochester, June 1988.

[Waltz, 1975] D. Waltz. Understanding line drawings of scenes with shadows. In P. H. Winston, editor, *The Psychology of Computer Vision*, pages 19–91. McGraw-Hill, 1975.