Domain Dependence in Parallel Constraint Satisfaction

Paul R. Cooper and Michael J. Swain

Department of Computer Science
University of Rochester
Rochester, New York 14627
cooper@cs.rochester.edu, swain@cs.rochester.edu

Abstract

We describe a general technique for expressing domain knowledge in constraint satisfaction problems, and using it to develop optimized parallel arc consistency algorithms for the solution of problems in the domain. The technique is applied to reduce the space complexity of the the massively parallel AC Chip algorithm. Results of the optimizations are shown for an object recognition domain in which they reduce the complexity of the chip by many orders of magnitude. The technique can be applied analogously to reduce the time complexity of the uniprocessor arc consistency algorithm AC-4.

1 Introduction

When a problem domain is known in advance, it is possible to exploit characteristics of the domain in order to simplify algorithms working within that domain. In AI this approach has often been used to generate unprincipled heuristic solutions whose generality is ill-defined. But more principled approaches to domain constraint exploitation also exist. For example, Mackworth *et al* [1985] show how hierarchic domains can be exploited in the solution of constraint satisfaction problems.

In this paper we show how to exploit domain information about arbitrary domains to generate improved arc consistency algorithms for constraint satisfaction problems (CSP's). A typical CSP is described by a set of nodes, a set of labels for the nodes, and unary and binary constraint predicates P and Q. Solving the constraint satisfaction problem involves finding a set of labels for the nodes that is consistent with the constraints. Arc consistency is a technique for eliminating solutions that are locally inconsistent [Mackworth, 1977, Mackworth and Freuder, 1985, Mohr and Henderson, 1986]. We have previously developed a massively parallel algorithm for arc consistency that can be implemented directly in VLSI. This algorithm is called the AC Chip, and is closely related to the uniprocessor algorithm AC-4 [Mohr and Henderson, 1986]. The AC Chip finds arc consistent solutions to constraint satisfaction problems almost instantaneously, at the cost of a quadratic space complexity [Swain and Cooper, 1988].

This space complexity can be reduced by specializing the algorithm to accept problems from a restricted domain of all possible CSP's. To accomplish this, domain knowledge is expressed as meta-constraints on the constraint predicates P and Q. We describe a circuit minimization algorithm, *Circ-Min,* that takes such a domain description and produces an optimized constraint satisfaction circuit for that domain. The algorithm performs a set of optimizations that may reduce the numbers of gates needed by orders of magnitude. We illustrate the process with results from an example domain we are studying, the recognition of Tinkertoy objects from their compositional structure [Cooper and Hollbach, 1987, Cooper, 1988, Swain and Cooper, 1988]. Space complexity improvements for the AC Chip algorithm can also be translated to time complexity improvements in AC-4.

2 Problem Formulation

A particular problem or class of problems is formulated as a constraint satisfaction problem by defining the elements of the general constraint satisfaction problem in terms of the specific application being investigated. That is, a set of nodes and a set of labels must be defined in terms of the problem at hand. Likewise, the unary and binary predicates *P* and Q must be defined in terms relevant to the application problem.

For illustrative purposes, consider as an example problem the recognition of Tinkertoy stick figures such as the "horse" in Figure 1. Tinkertoy objects consist of disks attached by connecting rods of differing lengths. The rods are plugged into *slots* on the junction disks (Figure 2). In the Tinkertoy recognition problem, structural descriptions derived from an image must be matched to stored descriptions in a model base. This matching problem is posed as a CSP by using the nodes in the CSP to represent the parts of the image description, and the labels in the CSP to represent the parts of the corresponding model description. A unique solution to the CSP then indicates a unique correspondence between the parts of the object and model. In the following examples, we define the rods and slots of the Tinkertoy descriptions to be the primitive parts.

To complete the formulation of Tinkertoy matching as a CSP, the unary and binary constraints are defined in terms of properties of the nodes and labels as follows[Swain and Cooper, 1988]:

$$\begin{array}{rcl} P(i,x) & = & (rod(i) \wedge rod(x)) \wedge (length(i) = length(x)) \\ & \vee & (slot(i) \wedge slot(x)) \wedge (filled(i) = filled(x)) \end{array}$$

$$Q(i,x,j,y) = \begin{cases} \mathbf{t} \text{ if } i = j \text{ and } x = y \\ \mathbf{f} \text{ if } i = j \text{ or } x = y \text{ but not both} \\ \\ sd(i,j) \wedge sd(x,y) \wedge \\ \quad [p(i) - p(j)]_s = [p(x) - p(y)]_s \vee \\ sr(i,j) \wedge sr(x,y) \wedge \\ \quad connected(i,j) = connected(x,y) \vee \\ rr(i,j) \wedge rr(x,y) \quad \text{otherwise.} \end{cases}$$

where some of the predicates can be expanded as follows:

$$\begin{array}{rcl} p(i) & = & position(i) \\ rr(i,j) & = & rod(i) \wedge rod(j), \\ sd(i,j) & = & same\_disk(i,j), \\ sr(i,j) & = & slot(i) \wedge rod(j) \vee rod(i) \wedge slot(j) \end{array}$$

and the terms are defined:

connected(i,j) True iff rod i is plugged into slot j or rod j is plugged into slot i.

filled(i) True iff a rod is plugged into slot i.

length(i) The length of the rod i.

position(i) The position of slot i on the disk, a number between 0 and $s-1$, where s is the number of slots per disk.

same_disk(i,j) True iff i and j are slots on the same disk.

A solution to the CSP framed in this way constitutes a correspondence between a Tinkertoy object and model. But these definitions provide no restrictions on the problem instances that can arise, so the resulting CSP still requires a solution algorithm of complete generality.

3 Domain Description

If the space of input problem instances is in some way restricted, the potential for specialized and more efficient solution algorithms exists. A problem domain is just such a restricted set of problem instances. Domains can be thought of as classes of problem instances with invariant properties. Consider for example the Tinkertoy matching problem as framed above. For all problem instances, it is never the case that a rod can match a slot. To exploit such characteristics in general requires a domain description language.

The purpose of the domain description is to represent invariant properties that hold over different problem instances. In constraint satisfaction problems, the nodes and labels are the entities that can have properties. In a Tinkertoy problem instance, the node t might represent a rod part, and thus have the property *rod{i).* If the node *i* is used to represent a rod part for all problem instances, *rod(i)* is a property invariant for all problem instances. Note that the mapping between Tinkertoy parts and the nodes of the CSP must maintain the invariant property

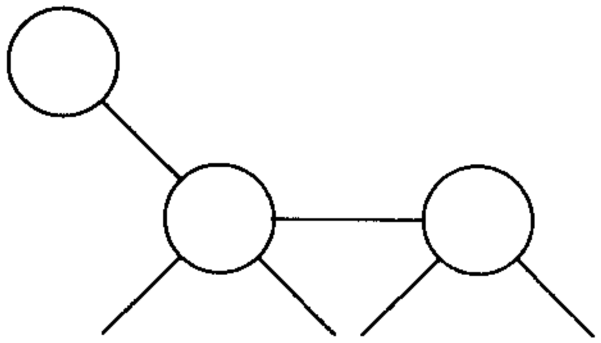

Figure 1: Example Tinker Toy Object

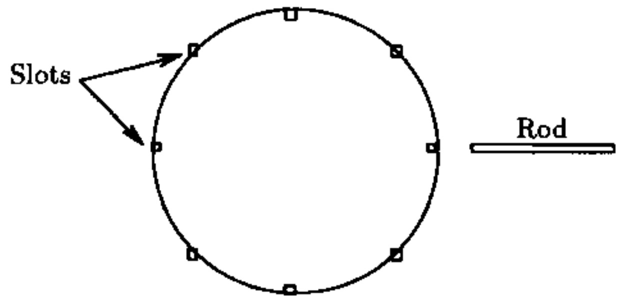

Figure 2: Example Slots (Rod Attachment Points) on Tinker Toy Disk

across problem instances. It is straightforward to arrange that a certain subset of nodes represent rods and a different subset of nodes represent slots for all problem instances.

Once an invariant is established in this way, it can be conveniently described with a three-valued meta-term. A meta-term *C* is defined in terms of an object property *C* as follows. By defining a domain as a (possibly infinite) set *6* and designating problem instances as a,

$$C'(i) = \begin{cases} \mathbf{t} \text{ if } C(i) = \mathbf{t} \text{ for all } \alpha \text{ in } \delta \\ \mathbf{f} \text{ if } C(i) = \mathbf{f} \text{ for all } \alpha \text{ in } \delta \\ \mathbf{u} \text{ otherwise} \\ (\text{ if } C(i) = \mathbf{t} \text{ for some } \alpha \text{ in } \delta \\ \text{and } C(i) = \mathbf{f} \text{ for some } \alpha \text{ in } \delta \text{)} \end{cases}$$

Invariants in a domain are, of course, represented by those meta-properties that evaluate to t or f. For example, *rod'(i)* would be t if *rod(i)* was true for all problem instances in the domain. Many properties will not be invariant, and their meta-description will evaluate to u.

Once the meta-terms describing the invariants are defined, it is a simple matter to define meta-predicates *P'* and *Q'.* These are exactly the same as the unary and binary constraint predicates *P* and *Q,* with meta-terms used in place of the object terms of *P* and *Q.* The meta-dedicates are evaluated with Kleene's three-valued logic !Turner, 1984]. Consider for example:

$$\begin{array}{rcl} P'(i,x) & = & (rod'(i) \wedge rod'(x) \wedge (length(i) = length(x))') \\ & \vee & (slot'(i) \wedge slot'(x) \wedge (filled(i) = filled(x))') \end{array}$$

Domain invariants can exploited when the meta-predicates *P'* or *Q'* evaluate as invariant on some arguments. As a concrete example, consider the meta unary predicate P'(i, *x)* for some *i* where *rod(i)* is always true

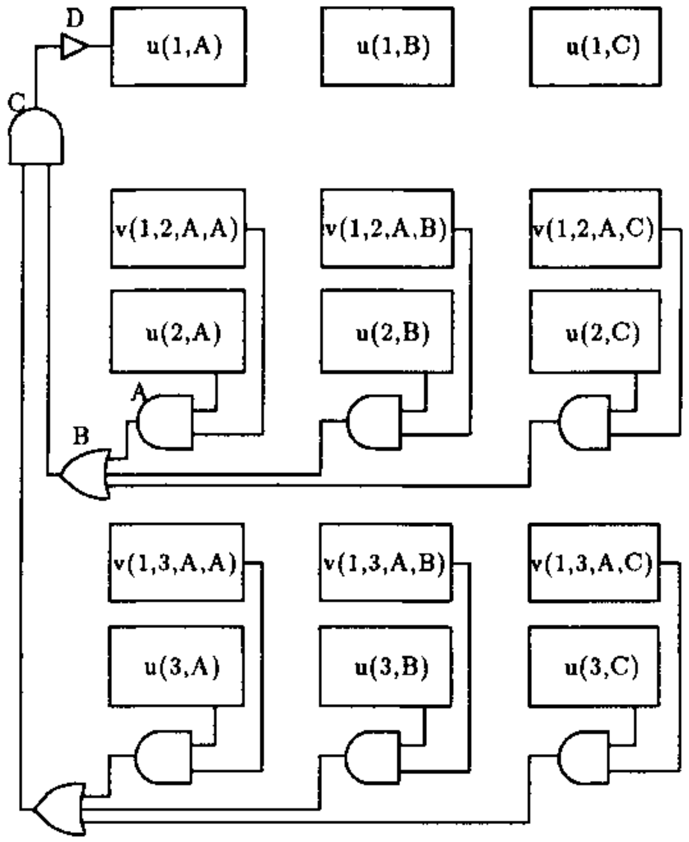

Figure 3: Partial Circuit Diagram for the AC Chip

(rod'(i) = t), and for some x where *slot(x)* is always true. Because a rod does not correctly correspond to a slot, the predicate P(i, x) will evaluate to f *for all problem instances* (P'(i, *x)* = f). This fact is known in advance of runtime, and can be exploited in developing the algorithms to solve the CSP in this domain. We demonstrate this optimization with a particular parallel algorithm for arc consistency, the AC Chip.

4 The AC Chip

The AC Chip is a massively parallel digital circuit that computes arc consistency in worst case time *0(na).* The design is based upon the unit/value principle [Barlow, 1972, Ballard, 1986], and is limited to computing problems of maximum size *n* nodes and *a* labels. In short, the important parts of the design are an explicit representation of every value of the P and Q predicates in flip-flops, and combinational circuitry that eliminates inconsistent label candidates by implementing the equation:

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

There are two arrays of flip-flops, *u(i,x)* and *v{i,j,x,y)* corresponding to the predicates P and Q respectively. The circuit for one candidate labeling (one element of P(i,x)) is given in Figure 3.

The gates that implement the above expression are designated, from the inside of the expression to the outside, and gates *A(i, j,* x, y), or gates *B(i, j,* x), and gates C(i,x) and not gates D(i,x). They are labeled in Figure 3. Identical circuitry exists for all *na* flip flops in the array representing P. More details concerning the AC Chip may be found in Swain and Cooper [1988].

```
procedure Circ_Min
    inputs:       domain meta-predicates $P'$ and $Q'$
                  node tree node_hier
                  label tree label_hier
    modifies:     circuit, as in Figure 3
begin
    (* P elimination *)
    forall $(i,x)$ in $P'$ do
        if $P'(i,x) = f$ then
            Recursive_Delete(flip-flop $u(i,x)$)

    (* Q elimination *)
    forall $(i,j,x,y)$ in $Q'$ do
        if $P'(j,y) = f$ or $Q'(i,j,x,y) = f$ then
            Recursive_Delete(and gate $A(i,j,x,y)$)
        else if $Q'(i,j,x,y) = t$ then
            begin
                wire output of flip-flop $u(j,y)$ to or gate $B(j)$
                Recursive_Delete(and gate $A(i,j,x,y)$)
            end

    Replace_By_Hierarchical_Gates(circ, label_hier, node_hier);
    Merge_With_Same_Inputs(circ);
    Flatten_Hierarchies(circ);
end
```

Figure 4: Procedure Circ_Min

For the matching problem we investigate below, *a* = *n* = 29. To solve such a problem the general purpose AC Chip would require 1.4 million gates, already a very large chip! The domain optimizations described in the next section reduce this value to 66,000.

5 Optimizing the Algorithm for the Domain

If the meta-predicates P' or *Q'* are t or f for some arguments, this translates to flip-flops which are always on or off in the AC Chip arc consistency algorithm. Systematically modifying the circuit to exploit the ramifications of these constants constitutes the domain dependent optimization of the algorithm. For example, if for some *i* and x P'(i,x) is f, the entire circuit of Figure 3 is not required for the *u(i, x)* element. The algorithm that performs these simplifications in general is given in Figure 4, and is called Circ_Min.

Circ_Min consists of five phases: *P elimination, Q elimination, Replace-By-Hierarchical-Gates, Merge-WithSame-Inputs,* and *Flatten-Hierarchies.*

P-eliminatton deletes any flip-flops *u(i,x)* that represent impossible candidate labels along with the trees of gates that feed into them. *Q elimination* removes type *A* and gates that are not necessary and replaces each one by a wire when *Q'* == true. These two optimizations are often the most important.

The purpose of the remaining three procedures is to factor out common sub-terms in the circuit and eliminate the redundant circuitry that computes them. The procedure *Replace_By-HierarchicaLGates* replaces a single gate by a hierarchy of gates in the pattern set by

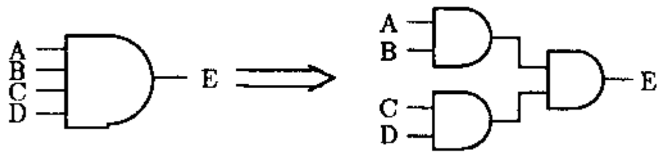

Figure 5: Procedure Replace_By_Hierarchical_Gates

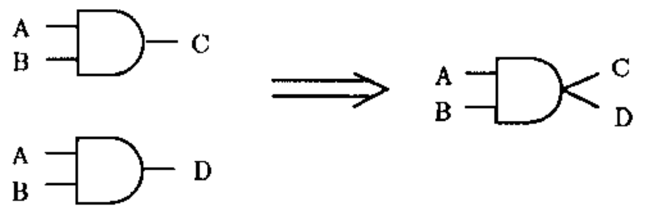

Figure 6: Procedure Merge_With_Same_Inputs

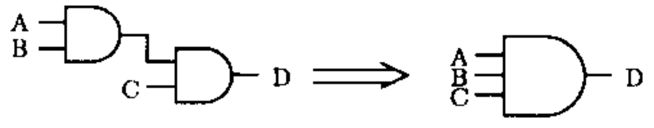

Figure 7: Procedure Flatten_Hierarchy

the template *or_hier* (for or gates) or *and-hier* (for and gates) (Figure 5). These hierarchies are supplied by the user, and represent additional description of the domain. The hierarchy must correspond to natural groupings of the labels for hierarchical gate replacement to enable more wires to be eliminated in the merge phase. The procedure *Merge-With_Same-Inputs* replaces redundant gates by one gate (thus also eliminating redundant input wires), and wires the output appropriately to the multiple places it is used (Figure 6). Finally, the procedure *Flatten-Hierarchy* replaces hierarchies of gates of the same type with functionally equivalent multiple-input gates. *Flatten-Hierarchy* thus inverts the function of Replace.by .Hierarchical-Gates where it is still possible after common subterms have been eliminated (Figure 7). It is reasonable to apply this function only to minimize the number of gates and wires. If other complexity measures (wire length or wire crossings) are important, the functionality of *Flatten-Hierarchy* might be undesirable.

The procedure *Recursive-Delete* deletes the gate that is its argument, and recursively deletes any gates which become redundant by its deletion because they have no inputs or no outputs.

Analysis and Performance

The correctness of the algorithm can be determined by noting that each transformation converts one or more gates into an equivalent set of gates. Therefore, the function of the AC circuit is preserved throughout the algorithm.

A straight-forward analysis shows that the algorithm takes $0(a^2n^2)$ time and space. Since the circuit reduction algorithm is run offline, the exact value of its complexity is not important. However, it should be noted that the algorithm does not need combinatorial resources; in fact it needs time and space linear in the original size of the AC circuit.

Circ-Min may reduce the complexity of an AC circuit by orders of magnitude. For instance, for Tinkertoy do-

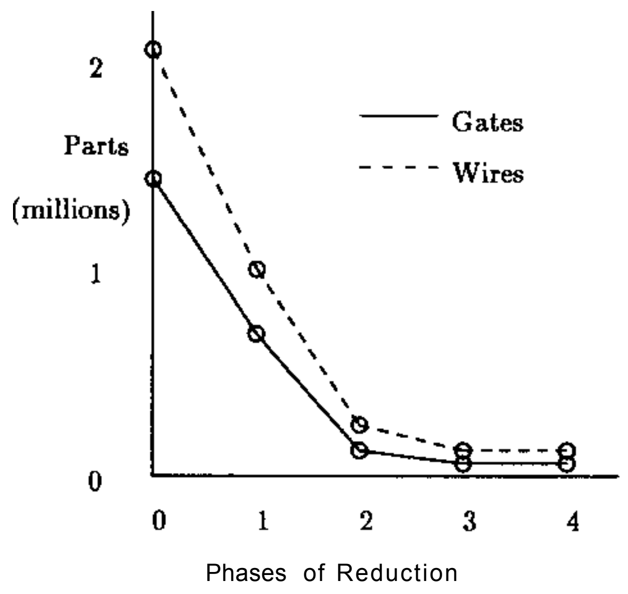

Figure 8: *Example Circuit Optimization Results*
Tinkertoy domain problem: 5 rods, 3 disks, 8 slots
Reduction Phases:
1: P flip-flop elimination
2: Q flip-flop elimination
3: gate merging
4: hierarchical gate substitution

main problems with a number of disks *d,* number of rods r, and number of slots per disk s, the number of gates is reduced from almost $2(r + ds)^4$ to about $2r^2d'^2s^2$. For the Tinkertoy domain with a maximum of 5 rods and 3 disks, each with 8 slots, this is a twenty-fold decrease (see Figure 8).

Figure 9 gives part of the reduced circuit for a Tinkertoy matching problem with at most 3 rods and 3 disks, with 2 slots per disk. The Figure gives the input circuit to a single target candidate labeling - the match of slot 1 on disk 1 of the object, to slot 2 of disk 3 of the model. This candidate labeling is shaded in Figure 9.

In the figure, the square and rectangular boxes represent the array of flip-flops *u(i,x)* (holding *P(i,x))* and the array of flip-flops *v(i,j, x,y)* (holding *Q(i,j,x,y))* respectively. With no optimizations, the circuit would look everywhere as it does in the lower left corner of the Figure. But all four types of optimizations have been performed. P-elimination has eliminated *u(i, x)* flip flops (and their associated circuitry) from the upper left-hand corner and lower right-hand corners of the *u(i,x)* array. Q elimination has eliminated the *v(i,j,x,y)* flip flops associated with all of the slot-slot matches. Slot-slot matches have been be divided into two groups - those compatible with the target candidate labeling *(Q'* = t), and those incompatible with the target candidate labeling *{Q'* = f). Only those that are compatible have an output wire and can affect the consistency of the candidate labeling. Because of gate merging the same or gates shown in the diagram that are receiving input from the slot-slot matches feed to many other slot-slot matches. Hierarchical gate substitution has broken these or gates into a two-level hierarchy, with slots from the same disk

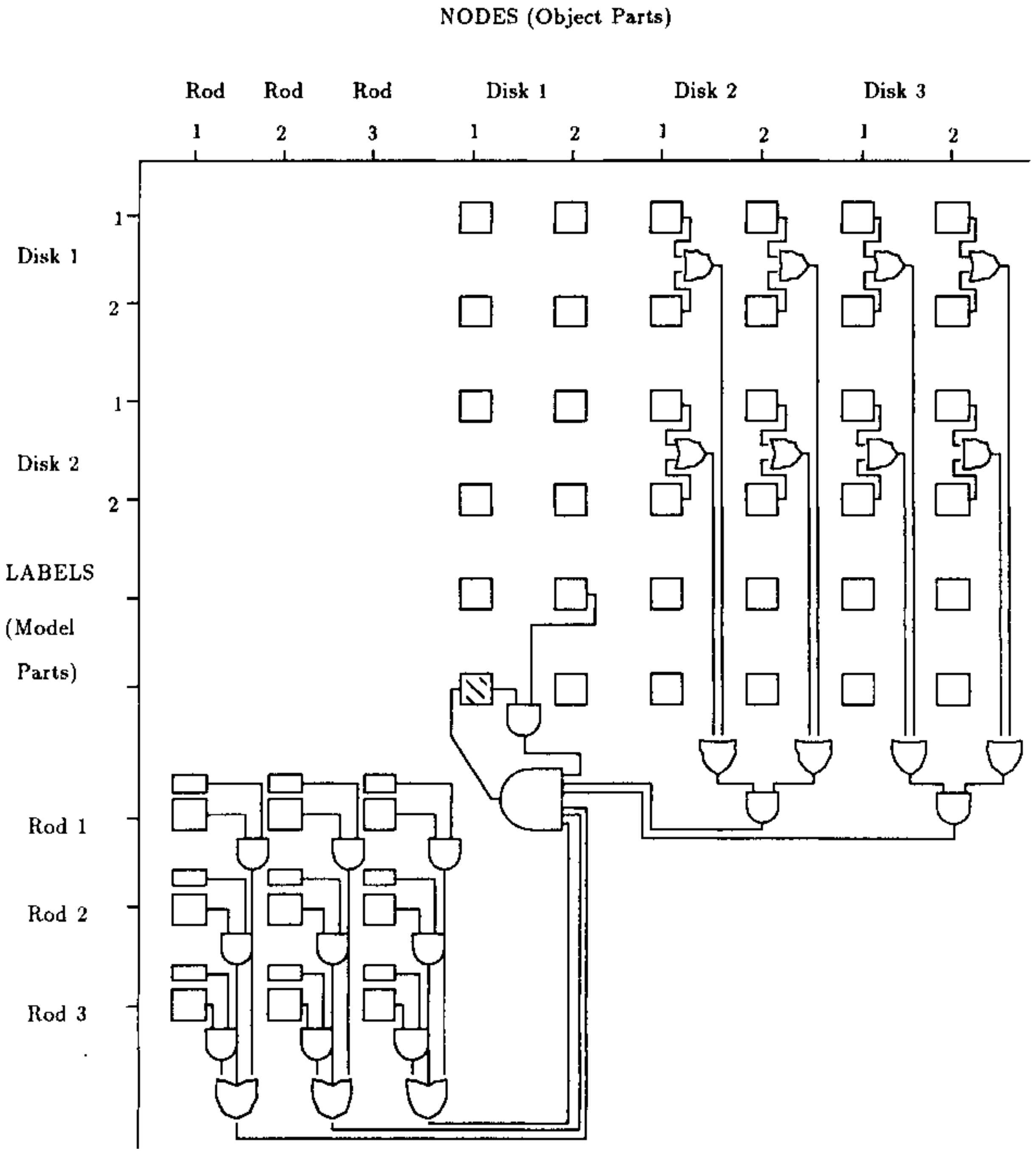

Figure 9: Reduced circuit for slot-slot Tinkertoy match

grouped together at the lower level.

We do not claim to be able to generate the optimal (most reduced) circuit for any problem. Domain knowledge not expressed in the meta-predicates might be exploitable [Cooper and Swain, 1988], or subtle optimizations not discovered by Circ_Min might be feasible. Since, in general, circuit optimization is a difficult problem and must be approached heuristically [Brayton et al, 1984], we do not expect there to be a tractable algorithm that finds the optimal circuit.

Because of the close relationship between the sequential arc consistency algorithm AC-4 [Mohr and Henderson, 1986] and the AC-Chip, the domain specific optimizations which reduce the space complexity of the AC Chip algorithm can be used to reduce the time complexity of AC-4, by modifying AC-4 so that it simulates the optimized chip [Cooper and Swain, 1988].

6 Conclusions

We have shown how to exploit information about arbitrary problem domains in order to generate more efficient arc consistency algorithms for constraint satisfaction problems in the domain. The domain description is formalized as meta-predicates whose purpose is to represent invariant properties of domain problem instances. The meta-predicates are the input to an algorithm that produces an optimized Arc Consistency (AC) Chip that may have space complexity many orders of magnitude lower than the general-purpose AC Chip. Because of the close relationship between the AC Chip and sequential arc consistency algorithm AC-4, the time complexity of AC-4 can be reduced by the same technique.

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, Boiling AFB, DC 20332 under Contract No. F30602-85-C-0008. The latter contract support the Northeast Artificial Intelligence Consortium (NAIC).

References

[Ballard, 1986] Dana H. Ballard. Cortical connections and parallel processing: An alternative model for cognitive science. Behavioral and Brain Sciences, 9(1):67-120,1986.

[Barlow, 1972] H. B. Barlow. Single units and sensation: A neuron doctrine for perceptual psychology? Perception, 1:371-394, 1972.

[Brayton et al, 1984] Robert K. Brayton, Gary D. Hachtel, Curtis T. McMullen, and Alberto L. Sangiovanni-Vincentelli. Logic Minimization Algorithms for VLSI Synthesis. Kluwer Academic Publishers, 1984.

[Cooper and Hollbach, 1987] Paul R. Cooper and Susan C. Hollbach. Parallel recognition of objects comprised of pure structure. In Proceedings of the Image Understanding Workshop, pages 381-391, 1987.

[Cooper and Swain, 1988] Paul R. Cooper and Michael J. Swain. Parallelism and domain dependence in constraint satisfaction. Technical Report TR 255, Department of Computer Science, University of Rochester, December 1988.

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

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

[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.

[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 et al., 1985] Alan K. Mackworth, Jan A. Mulder, and William S. Havens. Hierarchical arc consistency: Exploiting structured domains in constraint satisfaction problems. Computational Intelligence, 1:118-126, 1985.

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

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

[Swain and Cooper, 1988] Michael J. Swain and Paul R. Cooper. Parallel hardware for constraint satisfaction. In Proceedings AAAI-88, the American Association for Artificial Intelligence Conference, St. Paul, Minn., August 1988.

[Turner, 1984] Raymond Turner. Logics for Artificial Intelligence. Ellis Horwood Ltd., 1984.

[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.