From: AAAI-83 Proceedings. Copyright ©1983, AAAI (www.aaai.org). All rights reserved.

SOLVING the GENERAL CONSISTENT LABELING (or CONSTRAINT SATISFACTION) PROBLEM: TWO ALGORITHMS and their EXPECTED COMPLEXITIES

Bernard Nudel

Dept. Computer Science, Rutgers University, New Brunswick, N.J. 08903

ABSTRACT

The Consistent Labeling Problem is of considerable importance in Artificial Intelligence, Operations Research and Symbolic Logic. It has received much attention, but most work has addressed the specialized *binary* form of the problem. Furthermore, none of the relatively few papers that treat the general problem have dealt analytically with the issue of complexity. In this paper we present two algorithms for solving the *general* Consistent Labeling Problem and for each of these the expected complexity is given under a simple statistical model for the distribution of problems. This model is sufficient to expose certain interesting aspects of complexity for the two algorithms. Work currently in progress will address more subtle aspects by extension to more refined satistical models.

I INTRODUCTION

The problem we consider here has received much attention under various names such as the *Constraint Satisfaction Problem* [1], the *Consistent Labeling Problem* [2], the *Satisficing Assignment Problem* [3], the *Discrtete Relaxation Problem* [4] and the *Relation Synthesis Problem* [5]. We will use the term *Consistent Labeling Problem* or CLP.* Relatively little appears treating the general form of the Problem - some exceptions are [5, 7, 2, 8]. The specialized *binary* form is treated in [3, 9, 10, 11, 12] and notably in [13, 14] where analytic expected complexities are derived. A summary of [14] appears in [15].

The **general Consistent Labeling Problem CLP** is characterized by a finite list Z of n **variables**. Each variable z_i has an associated finite **domain** D_{z_i} from which it can take any of M_{z_i} **values** or **labels**. **Constraints** exist on which values are mutually compatible for various subsets of the n variables. A **solution-tuple** or **consistent-labeling** assigns to each of the n variables a value in its corresponding domain such that all problem constraints are simultaneously satisfied. The goal is to find one or more solution-tuples. We analyze algorithms that find *all* solution-tuples for a problem. A constraint involving exactly r variables is said to be of **arity** r or to be **r-ary**. The sub-Problem of CLP containing all, and only, problem instances with at least one r-ary constraint, and no constraints of arity greater than r we call **the r-ary Consistent Labeling Problem rCLP**. Note that an r-ary problem may or may not have constraints of arity less than r. For ease of presentation, the results here are for the **pure r-ary** Problem **πrCLP** on n variables each of equal domain size M. Instances of this Problem have exactly one r-ary constraint for each of the $\binom{n}{r}$ possible r-subsets of the n problem variables, and have *no* constraints of arity less than r. In [16] we generalize to instances whose variables may have different size domains and whose constraints may have different arities, with possibly zero, one or even several constraints, independently for any subset of variables. The theory there also distinguishes between otherwise identical problems whose constraints differ in their *degree of constraint* or **compatibility** of their argument variables. This makes possible relatively problem-specific prediction of complexity for any conceivable clp. It also allows capturing of the important search-order effects on complexity that one finds for clps.

*"CLP" denotes a family of problem instances [6]. A specific instance of CLP will be denoted by the lower-case "clp". Analogously, we write "Problem" (with an upper-case P) for a set of instances and "problem" for a single instance.

II STATISTICAL MODELS FOR PROBLEMS

In [13] Haralick carries out an expected complexity analysis for two pure binary CLP algorithms (BT and FC below) under a simple statistical model for problem generation. We call this **model 0**. This model simply considers that for any pair of values assigned to any pair of variables, the probability is p that they are compatibile with respect to the corresponding binary constraint. In [14] we extend Haralick's work by carrying out expected complexity analyses under more complex **models 1 and 2**, which have the important advantage of capturing the effect on complexity of changes in search orders used by an algorithm. Such analyses were thus useable to give theoretical insight into how to intelligently order the seach for solutions - often at significant savings in search effort. All analyses to date however treat only the pure binary Problem π2CLP.

In this first analytic work beyond the binary CLP case we will stay with the analogue of the binary statistical model 0. Again this will capture no order dependence effects - but we nevertheless will obtain useful insight into the *main* features of algorithm complexity for solving the more general clps. For the general CLP the analogue of the above statistical model 0 is simply this: for each possible tuple of values assigned to the argument variables of a constraint, the probability is p that the value-tuple belongs to (i.e. is compatibile with respect to) the constraint. We use this model for the pure r-ary CLP here. We expect shortly to complete the analysis for the fully general CLP, under this and the more refined statistical models 1 and 2. Results under the model 2 will provide theoretical insight into order effects in the general case analogous to those made available in [14] for the pure binary case, as well as providing the analogous precision in predicting a problem's complexity.

III ALGORITHMS

In [13] Haralick empirically compares seven different algorithms for solving *pure binary* clps. For the experiments he conducted the ranking of algorithms obtained was essentially:

best = wFC > FC > BM > PL > FL > BC > BT = worst

where the abbreviations denoting algorithms are as used in [14]. In particular, BT denotes the **Backtracking** algorithm [17, 18], FC denotes the **Forward Checking** algorithm and wFC is a variant of FC we call **word-wise Forward Checking** that exploits the bit-parallelism available in present machines. FC and wFC seem to have been independently discovered by Haralick [13] and McGregor [12], but in fact FC also appears in the earlier paper of Golomb and Baumert [17] where it is referred to as **Backtracking with preclusion**. In figures III-1 and III-2 respectively we present recursive versions of our general Backtracking (gBT) and general Forward Checking (gFC) algorithms for solving *arbitrary* clps. These are quite natural generalizations of the corresponding pure binary algorithms BT and FC, which we now rename to be π2BT and π2FC.

Common to both algorithms is the notion of an **instantiation order**** $X = [\ x_1\ x_2\ .\ .\ x_n\ ]$ being some permutation of the conventionally ordered problem variables $Z = [\ z_1\ z_2\ .\ .\ z_n\ ]$. By definition, a **k-th level node** of the gBT or gFC search tree is formed when variable x_k is instantiated to (assigned) some value $\bar{x}_k$ from its corresponding domain D_{x_k} - this happens at lines 2 of gBT and gFC. At all k-th level nodes in either algorithm, instantiations or value assignments have been made for the *same* ordered sequence $A_k = [\ x_1\ x_2\ .\ .\ x_k\ ]$ of the first k variables of (globally available) list X. Nodes at level k are distinct only in that they correspond to different instantiations made to the variables of A_k. An instantiation sequence for the variables of A_k is a list $\bar{A}_k = [\ \bar{x}_1\ \bar{x}_2\ .\ .\ \bar{x}_k\ ]$ of values $\bar{x}_i \in D_{x_i}$ for the variables of A_k. It is built up in these algorithms using $||$ the list concatenation operator. Such an **instantiation sequence** $\bar{A}_k$ corresponds to a **path** through the search tree. Note that the node for a given instantiation sequence $\bar{A}_{k+i}$ may not actually be generated due to the discovery of a violation of problem constraints at an earlier node $\bar{A}_k \subset \bar{A}_{k+i}$ on that path. This is in fact where the usefulness of such algorithms arises since attempts to generate any candidate solution-tuples containing $\bar{A}_k$ are then avoided. Initially for both alorithms k = 1, $A_{k-1} = \emptyset$. In gBT all domains D_f remain unchanged and are global. In gFC domains of some variables are in

**We use square brackets [. .] throughout to denote a list or vector.

```
1  gBT( k $\bar{A}_{k-1}$ )
2    Do for all $\bar{x}_k \in D_{x_k}$
3    $\bar{A}_k \leftarrow \bar{A}_{k-1}\ ||\ [\bar{x}_k]$
4    If gCheck( k $\bar{A}_k$ ) $\neq$ "$\bar{x}_k$ wipe-out"
5      then if k < n then  gBT( k+1 $\bar{A}_k$ )
6                          else print $\bar{A}_k$
7    end
8  end gBT

1  gCheck( k $\bar{A}_k$ )
2    Do for i = 1 to $m_k$
3    If $V_{ki}$( $\bar{A}_k$ ) $\notin T_{ki}$
4      then return "$\bar{x}_k$ wipe-out"
5    end
6  Return "No $\bar{x}_k$ wipe-out"
7  end gCheck
```

Figure III-1 gBT and its subroutine gCheck

```
1   gFC( k $\bar{A}_{k-1}$ D )
2     Do for all $\bar{x}_k \in D_{x_k}$
3     $\bar{A}_k \leftarrow \bar{A}_{k-1}\ ||\ [\bar{x}_k]$
4     If k < n then Do
5                   D' $\leftarrow$ gFilter( k D $\bar{A}_k$ )
6                   If D' $\neq$ "$D_f$ wipe-out"
7                     then gFC( k+1 $\bar{A}_k$ D' )
8                   end
9               else print $\bar{A}_k$
10    end
11  end gFC

1   gFilter( k D $\bar{A}_k$ )
2     Do for all f $\in F_k$
3       Do for i = 1 to $m_{kf}$
4         Do for all $\bar{f} \in D_f$
5         If $V_{kfi}$( $\bar{A}_k$ $\bar{f}$ ) $\notin T_{kfi}$
6           then $D_f \leftarrow D_f$ - [$\bar{f}$]
7         end
8       If $D_f = \emptyset$ then return "$D_f$ wipe-out"
9       end
10    end
11  D' $\leftarrow$ [ $D_{x_{k+1}}$ $D_{x_{k+2}}$ ... $D_{x_n}$ ]
12  Return D'
13  end gFilter
```

Figure III-2 gFC and its subroutine gFilter general updated at each node and are passed as an argument in the vector of domains $D = [\ D_{x_1}\ D_{x_2}\ \ldots\ D_{x_n}\ ]$. Testing tuples of instantiations for compatibility with respect to constraints is denoted as a test of membership of the tuple in the set of compatible tuples for the corresponding constraint (line 3 of gCheck and line 5 of gFilter). In practice these tests will usually be carried out on a constraint defined *intensively* via a procedure rather than defined *extensively* as a set. The algorithm and our analysis below are compatible with both representations.

Algorithm gBT: At a level k node the **current instantiation** $\bar{x}_k$ is tested for compatibility with respect to all m_k constraints that involve x_k and some of the k−1 earlier variables of A_k. For a pure r-ary clp $m_k = \binom{k-1}{r-1}$ since this is the number of possible r-ary constraints over x_k and r−1 other variables to be chosen from the k−1 variables of A_k besides x_k. Note that $\Sigma_{1\le k\le n} \binom{k-1}{r-1} = \binom{n}{r}$ as required. The i-th constraint tested (if no wipe-out prevents it) at each level k node we denote T_{ki}. Its list of argument variables we denote by V_{ki}. The corresponding list of values for these variables, given the assignments of $\bar{A}_k$, is denoted $\bar{V}_{ki}(\ \bar{A}_k\ )$. This is simply the projection of $\bar{A}_k$ onto V_{ki}. For example if $A_k = [\ z_3\ z_6\ z_2\ z_9\ z_4\ ]$, $\bar{A}_k$ = [e a e g b] and the argument-list for constraint V_{ki} is $[\ z_2\ z_4\ z_6\ ]$, then $\bar{V}_{ki}(\ \bar{A}_k\ )$ = [e b a]. Given the instantiations of $\bar{A}_{k-1}$, the current instantiation $\bar{x}_k$ violates constraint T_{ki} if $\bar{V}_{ki}(\ \bar{A}_k\ ) \notin T_{ki}$ and this is tested at line 3 of gCheck.

Algorithm gFC: In addition to A_k, gFC also uses $F_k = [\ x_{k+1}\ x_{k+2}\ \ldots\ x_n\ ]$ the list of **future variables**, or not yet instantiated variables, at level k. At a level k node, each future variable $f \in F_k$ has each of its potential future instantiations $\bar{f} \in D_f$ (as contained in D, the list of updated domains for future variables) tested for compatibility with respect to all m_{kf} constraints that involve x_k, f and some of the k−1 earlier variables of A_k. For pure r-ary clps, $m_{kf} = \binom{k-1}{r-2}$ since this is the number of possible r-ary constraints over variables x_k, f and r−2 other variables to be chosen from the k−1 variables of A_k besides x_k. Note that since there are n−k future variables f at level k, there is for the pure r-ary CLP $(\ n-k\ )\binom{k-1}{r-2}$ new constraints to check at level k and $\Sigma_{1\le k\le n}\ (\ n-k\ )\binom{k-1}{r-2} = \binom{n}{r}$ as required. The i-th such constraint involving f, tested at each level k node (if no wipe-out prevents it) we denote T_{kfi}. Its list of argument variables is V_{kfi} and the list of values assigned respectively to these variables given the instantiations of $\bar{A}_k$ and the value $\bar{f}$ being tested for f, we denote $\bar{V}_{kfi}(\ \bar{A}_k\ \bar{f}\ )$. Given $\bar{A}_k$, value $\bar{f}$ violates constraint T_{kfi} if $\bar{V}_{kfi}(\ \bar{A}_k\ \bar{f}\ ) \notin T_{kfi}$ and this is tested at line 5 of gFilter. Lack of compatibility leads to $\bar{f}$ being removed or filtered from its domain D_f. The sample trace for 2FC appearing in fig 4 of [14] may be helpful.

Note that the selection order for $f \in F_k$ at line 2 of gFilter may be a function of level, or even a function of the node. In fact one could generalize the algorithm by merging the loops at lines 2 and 3 of gFilter to give

Do for $(f\ i) \in F_k \times \{\ 1$ to $m_{kf}\ \}$ in some order

returning "D_f wipe-out" as soon as one occurs. However for the pure binary clp, $m_{kf} = \binom{k-1}{r-2} = 1$ and only the order of selection of f from F_k is an issue. This is the case studied in [14] where we use the theory to suggest global and local Consistency-check Ordering (CO) heuristics for ranking the f of F_k. However when $m_{kf} \neq 1$ the question becomes how to rank the pairs (f i). If the statistical model were refined enough one might even study the advantages of merging the loop at line 4 as well with those at lines 2 and 3. However under the present simple statistical model 0, the nesting of loops shown is optimal, and any residual indeterminism is irrelevant (with respect to average complexity) in either algorithm. In particular, the instantiation ordering X is irrelevant on average under model 0. For individual problems though a good ordering can lead to significant savings, and our more refined model 1 and 2 analyses of [14] capture this effect for the pure binary case. These theories are then used there to suggest theory-based global and local Instantiation Order (IO) heuristics that are found to be quite effective in reducing the complexity of problem solving.

IV ANALYTIC RESULTS

Under statistical model 0 it is easy to determine the probability P(clp) of a given problem clp − analogously to the model 1 result for the r = 2 case given in [14]. In terms of P(clp) we can define the expected total number of nodes in a search tree (for a given algorithm) as $\bar{N} = \Sigma_{clp}$ N(clp) P(clp) where N(clp) is the actual total number of nodes generated for problem clp. Similarly, the expected total number of consistency-checks performed in the search tree (for a given algorithm) is by definition $\bar{C} = \Sigma_{clp}$ C(clp) P(clp) where C(clp) is the actual total number of checks for problem clp. These expressions are not useful as they stand since N(clp) and C(clp) are not known analytically. However they can be tranformed into useable expressions. We can use

$$N(clp) = \Sigma_{1\le k\le n} N(k\ clp) \quad \text{and} \quad C(clp) = \Sigma_{1\le k\le n} C(k\ clp)$$

where N(k clp) and C(k clp) are respectively the actual number of nodes generated and checks preformed at the k-th level in problem clp. In terms of these we can define the expected number of nodes and checks at the k-th level of the search tree, given by

$$\bar{N}(k) = \Sigma_{clp} N(k\ clp) P(clp) \quad \text{and} \quad \bar{C}(k) = \Sigma_{clp} C(k\ clp) P(clp)$$

The expected totals are then expressible as the expectation at a level summed over all levels

$$\bar{N} = \Sigma_{1\le k\le n} \bar{N}(k) \quad \text{and} \quad \bar{C} = \Sigma_{1\le k\le n} \bar{C}(k)$$

By successive transformations of this kind, the following expected-value expressions are obtained in [16]. As mentioned, for notational simplicity we present results for the pure r-ary CLP on n variables each variable having equal domain size M. We expect to present in [16] the fully general result under more refined statistical models 1 and 2.

Algorithm gBT:

$$\overline{N}(k) = M^k p^{\binom{k-1}{r}} \tag{1}$$

$$\overline{C}(k) = \overline{N}(k)\,\bar{c}(k) \tag{2}$$

$$\bar{c}(k) = [\ 1 - p^{\binom{k-1}{r-1}}\ ] \ / \ [\ 1 - p\ ] \tag{3}$$

Algorithm gFC:

$$\overline{N}(k) = M^k p^{\binom{k}{r}} [\ 1 - (\ 1 - p^{\binom{k-1}{r-1}}\ )^M\ ]^{n-k} \tag{4}$$

$$\overline{C}(k) = \overline{N}(k)\,\bar{c}(k) \tag{5}$$

For either algorithm, $\bar{c}(k)$ is the expected number of checks performed at a node generated at level k. An expression for $\bar{c}(k)$ of gFC still remains to be determined. We can use the above results to compare gBT with gFC in terms of the relative number of nodes the algorithms generate. Figure IV-1 shows the ratio $\overline{N}_{gFC}$ / $\overline{N}_{gBT}$ of the expected total number of nodes generated by the two algorithms. The problems solved are characterized by parameters p, n, M and r. We consider the case that M = n at two "reasonable" values of p: 0.5 and 0.75. In both cases three families of curves are shown, corresponding to three ways in which n varies with the independent variable r.

The n = c family: Curves for n = 5 and n = 10 are shown. As expected these curves generally increase with r since when r > n no constraint can be tested before termination of either algorithm. Both gBT and gFC then generate the same full search tree of all $\sum_{1 \leq k \leq n} M^k$ nodes so that $\overline{N}_{gFC}$ / $\overline{N}_{gBT}$ = 1.

The n = cr family: Curves for n = 1r, 3r/2, 3r and 5r are shown. In contrast to the n = c family, these curves generally decrease with r – in other words the relative advantage of gFC compared to gBT becomes even greater as r grows.

The n = r + c family: Curves for n = r+0, r+1, r+3, r+10, r+15 and r+30 are shown. This family shows the behaviours of both the above two families of curves. For the smaller c values these curves decrease with r. At larger c values the curves first increase with r but level off at larger r. In any case, the ratio $\overline{N}_{gFC}$ / $\overline{N}_{gBT}$ stays small and hence the relative advantage of gFC compared to gBT remains large.

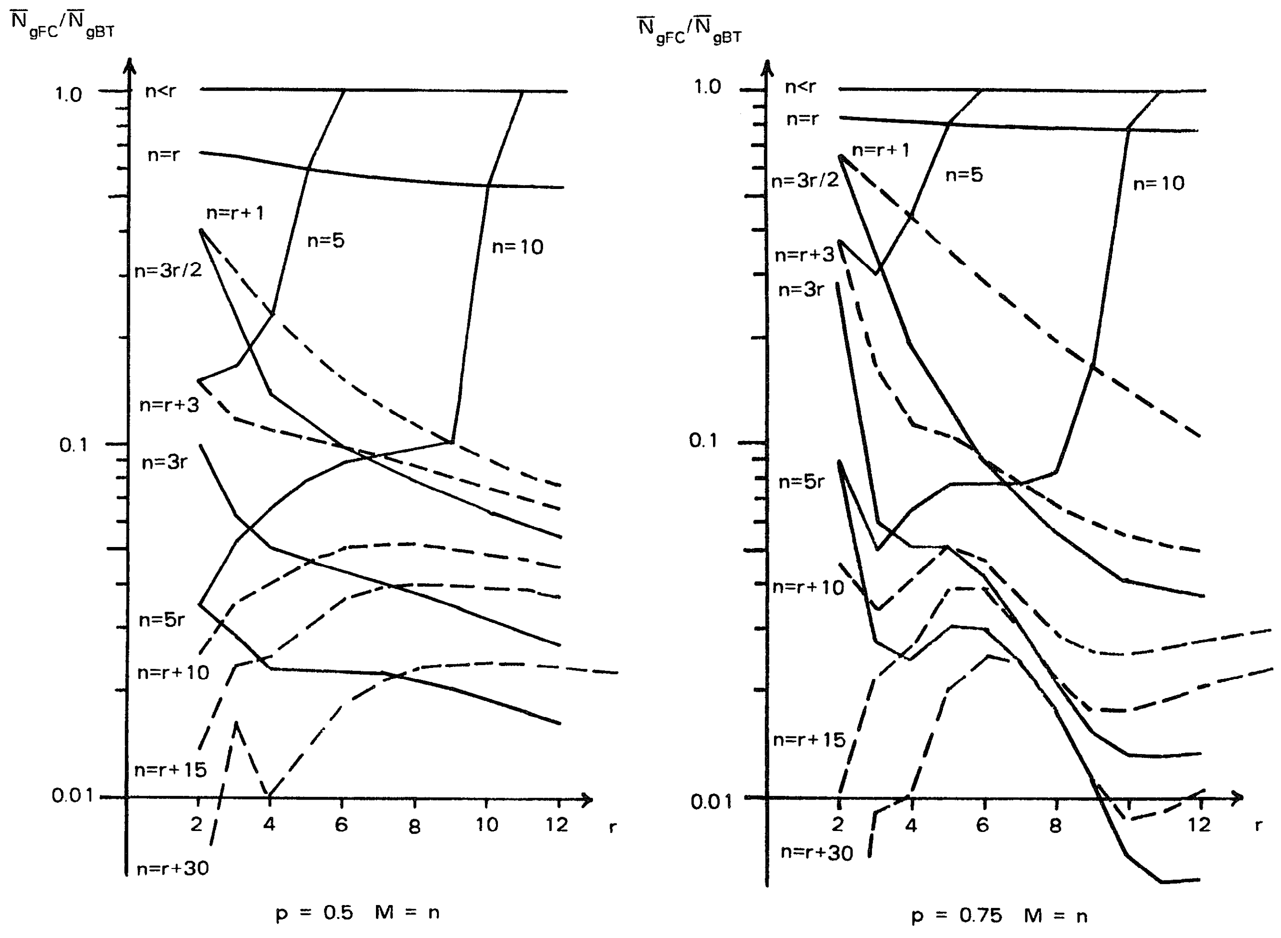

Figure IV-1 Analytic comparison of gFC and gBT for solving pure r-ary clps.

Note that as p goes to 1, both algorithms will be increasingly unable to find inconsistent search paths and, as above, both will generate full search trees with the maximum number of nodes at each level. In this limit therefore, $\overline{N}_{gFC} = \overline{N}_{gBT}$ and all curves collapse to be horizontal at value $\overline{N}_{gFC} / \overline{N}_{gBT} = 1$. This effect however requires p very close to 1. Even at p = 0.99 we have found that the advantage of gFC over gBT is often still significant.

It was however pointed out in [13] that $\overline{C}$, not $\overline{N}$, is the appropriate measure of complexity for these algorithms. For the pure binary case we have in [14] used $\overline{C}$ under model 0 to analytically compare algorithm π2FC against π2BT and π2FC against π2wFC. We expect that the corresponding comparison for the pure r-ary case will soon be possible, once we obtain an expression for the still outstanding $\bar{c}(k)$ of gFC.

ACKNOWLEDGEMENTS

Many thanks to my Ph.D. committee - Saul Amarel, Martin Dowd, Marvin Paull and especially my supervisor William Steiger - who strongly suggested I generalize to arbitrary arity constraints. Without their "encouragement" this paper may never have been written.

References

[1] Fikes, R. E. "REF-ARF: A system for solving problems stated as procedures." *Artificial Intelligence.* 1 (1970) 27-120.

[2] Haralick, R. M. and Shapiro, L. G. "The consistent labeling problem: Part I." *IEEE Trans. Pattern Analysis and Machine Intelligence.* PAMI-1:2 (1979) 173-184.

[3] Gaschnig, J., *Performance measurement and analysis of certain search algorithms,* PhD dissertation, Dept. Computer Science, Carnegie-Mellon U., 1979.

[4] Rosenfeld, A., Hummel, R. and Zucker, S. "Scene labeling by relaxation operations." *IEEE Trans. Systems, Man and Cybernetics.* SMC-6 (1976) 420-433.

[5] Freuder, E. C. "Synthesizing constraint expressions." *Comm. ACM.* 21 (1978) 958-966.

[6] Garey, M. R. and Johnson, D. S. *Computers and Intractability.* Freeman, San Francisco, 1979.

[7] Haralick, R. M., Davis, L. S. and Rosenfeld, A. "Reduction operations for constraint satisfaction." *Information Sciences.* 14 (1978) 199-219.

[8] Haralick, R. M. and Shapiro, L. G. "The consistent labeling problem: Part II." *IEEE Trans. Pattern Analysis and Machine Intelligence.* PAMI-2:3 (1980) 193-203.

[9] Gaschnig, J. "Experimental case studies of backtrack vs. Waltz-type vs. new algorithms for satisficing assignment problems." In *Proc. 2-nd National Conf. Canadian Soc. for Computational Studies of Intelligence.* Toronto, Ontario, 1978,

[10] Montanari, U. "Networks of constraints: fundamental properties and applications to picture processing." *Information Sciences.* 7 (1974) 95-132.

[11] Mackworth, A. K. "Consistency in Networks of Relations." *Artificial Intelligence.* 8 (1977) 99-118.

[12] McGregor, J. J. "Relational consistency algorithms and their application in finding subgraph and graph isomorphisms." *Information Sciences.* 19 (1979) 229-250.

[13] Haralick, R. M. and Elliot, G. L. "Increasing tree search efficiency for constraint satisfaction problems." *Artificial Intelligence.* 14 (1980) 263-313.

[14] Nudel, B. A. "Consistent-labeling problems and their algorithms: expected-complexities and theory-based heuristics." *Artificial Intelligence.* 21:1 and 2 March (1983) , Special issue on Search and Heuristics, in memory of John Gaschnig; This issue is also published as a seperate book: Search and Heuristics, North-Holland, Amsterdam 1983.

[15] Nudel, B. A. "Consistent-labeling problems and their algorithms." In *Proc. National Conf. Artificial Intelligence.* Pittsburg, 1982, 128-132.

[16] Nudel, B. A., *title to be decided,* PhD dissertation, Dept. Computer Science, Rutgers U., 1983, To appear

[17] Golomb, S. W. and Baumert, L. D. "Backtrack programming." *J. Assoc. Computing Machinery.* 12 (1965) 516-524.

[18] Bitner, J. R. and Reingold, M. "Backtrack programming techniques." *Comm. ACM.* 18 (1975) 651-656.