# Counting-Based Look-Ahead Schemes for Constraint Satisfaction

Kalev Kask, Rina Dechter, and Vibhav Gogate

Donald Bren School of Information and Computer Science
University of California, Irvine, CA 92967
{kkask, dechter, vgogate}@ics.uci.edu

**Abstract.** The paper presents a new look-ahead scheme for backtracking search for solving constraint satisfaction problems. This look-ahead scheme computes a heuristic for value ordering and domain pruning. The heuristic is based on approximating the number of solutions extending each partial solution. In particular, we investigate a recent partition-based approximation of tree-clustering algorithms, Iterative Join-Graph Propagation (IJGP), which belongs to the class of belief propagation algorithms that attracted substantial interest due to their success for probabilistic inference. Our empirical evaluation demonstrates that the counting-based heuristic approximated by IJGP yields a scalable, focused search.

## 1 Introduction

We investigate the power of solution counting as a heuristic for guiding backtracking search for finding a single solution of a constraint problem. Specifically, given a set of instantiated variables (i.e., a partial solution), the task is to compute, for each value $a$ of each uninstantiated variable $X$, the number of solutions that agree with the given instantiated variables as well as the candidate assignment $X = a$. Clearly, if we could solve this task exactly, it would make solving the CSP task trivial - we can generate a solution in a backtrack free manner by instantiating variables, one at a time, by choosing a value that has a solution count greater than 0, with respect to the previously instantiated variables. Since the counting problem is a #P-complete problem, the complexity of exact algorithms, such as variable elimination algorithms, is too high to be practical and approximations are necessary. Our approach is to approximate solution counting and use it as a heuristic function for guiding a (backtracking) search for solving the CSP problems.

Our approximation idea is motivated by the success of the class of generalized belief propagation, such as Join-Graph Propagation (IJGP) [5, 10] in solving the belief updating problem in Bayesian networks. IJGP(i) is a parameterized iterative propagation scheme controlled by its i-bound. As $i$ grows the algorithm is more accurate but also requires more time and space. When i equals the tree-width of the graph, the algorithm is exact. It was shown that the IJGP(i) scheme is powerful, and superior (provides a superior time-accuracy tradeoff)

to competing algorithms on many classes of belief networks. It was shown, in particular, that even the least accurate version of $i = 2$, is often quite good. We therefore adapt the IJGP scheme for the task of solution counting (SC) and use it as a heuristic to guide backtracking search. We compare the resulting backtracking algorithm called IJGP-SC against MAC [11], one of the most powerful lookahead backtracking methods, against Stochastic Local Search (SLS), and against backtracking algorithms equipped with alternative heuristics computed by *mini-clustering/mini-bucket* (a partition-based scheme of tree-decomposition algorithms). We compare the performance of these algorithms on random CSPs, graph coloring problems and Quasi-group completion problems.

Our results are very promising. We show that IJGP-SC yields a very focused search with relatively few deadends, especially when the problems become larger and harder. We compare the algorithms in terms of scalability - how does the relative complexity of an algorithm change as the problem size grows and what is the largest problem size that each of the algorithms can solve in a range of hardness. Our results show that over the problems tried the lowest bound of i=2 was by far the most cost-effective (the time overhead for larger i-bounds did not pay off). We therefore focused much of the subsequent empirical testing on IJGP(i=2)-SC. Our results on random CSPs show that IJGP-SC is more scalable than MAC and SLS. Specifically, while MAC/SLS are better at small problem sizes ($N = 100 - 300$), IJGP-SC improves as $N$ grows, and when $N = 500$, IJGP-SC outperforms other algorithms; at $N = 1000$ IJGP-SC can solve more problem instances faster than competing algorithms and by $N = 2000$ (which is the largest problem size we tried), none of the competing algorithms can solve any problems, while IJGP-SC can still solve about 25% of the solvable problems (given a time-bound of 5 hours). We also compare the performance on hard 3-coloring and Quasi-group completion problems. Our preliminary results show that MAC is the best algorithm on 3-coloring problems (with $N$ up to 200) while IJGP-SC is superior to MAC on Quasi-group completion problems.

We also investigate the impact of the number of iterations, used in IJGP-SC, on the heuristic quality. It can be shown that when IJGP-SC converges, it performs arc-consistency, yielding pruning of domains of variables similar to MAC. However, waiting for convergence (relative to zero counts; general convergence is not guaranteed) may not be cost effective. IJGP-SC with fewer iterations may produce a value ordering that is sufficiently accurate to result in a few deadends. We contrast this with MAC which relies heavily on full arc consistency (domain pruning).

Our results are significant because 1) our base algorithm in IJGP-SC is naive backtracking that is not enhanced by either backjumping or learning 2) our implementation can be further optimized to yield significant speed-up and most significantly, it cannot be matched to the level of hacking invested in MAC, 3) we believe the strength of MAC (pruning domains by arc-consistency) and the strength of IJGP in value ordering are complementary and can be combined in a single algorithm. We plan to investigate the combined approach in the future.

The paper is organized as follows. In Section 2 we provide background information. In Sections 3 we introduce the solution-count heuristic function. The corresponding search algorithm and other competing algorithms are presented in Section 4. In Section 5 we discuss experimental data and finally in Section 6 we provide discussion of related work and concluding remarks.

## 2 Preliminaries

**Definition 1 (constraint satisfaction problem).** *A* Constraint Network (CN) *[4] is defined by a triplet* $(X, D, C)$ *where* $X$ *is a set of variables* $X = \{X_1, ..., X_n\}$*, associated with a set of discrete-valued domains,* $D = \{D_1, ..., D_n\}$*, and a set of constraints* $C = \{C_1, ..., C_m\}$*. Each constraint* $C_i$ *is a pair* $(S_i, R_i)$*, where* $R_i$ *is a relation* $R_i \subseteq D_{S_i}$ *defined on a subset of variables* $S_i \subseteq X$ *called the scope of* $C_i$*. The relation denotes all compatible tuples of* $D_{S_i}$ *allowed by the constraint. The primal graph of a constraint network, called a* constraint graph*, has a node for each variable, and an arc between two nodes iff the corresponding variables participate in the same constraint. A* solution *is an assignment of values to variables* $x = (x_1, ..., x_n)$*,* $x_i \in D_i$*, such that all constraint are satisfied. The Constraint Satisfaction Problem (CSP) is to determine if a constraint network has a solution, and if so, to find a solution. A binary CSP is one where each constraint involves at most two variables. Solution-counting is the task of counting the number of solutions.*

The task of solution counting can be solved when formulating each constraint by a cost function that assigns 1 for allowed tuples and 0 for unallowed tuples. The cost of an assignment is the product of all cost functions and the task is to count the number of assignments with cost 1. This task can be solved exactly by inference algorithms defined over a tree-decomposition of the problem specification. Intuitively, a tree-decomposition takes a collection of functions and partitions them into a tree of clusters. The cluster tree is often called a *join-tree* or a *junction tree*. The clusters in the tree-decomposition have to satisfy the running-intersection property [4]. The subset of variables in the intersection of any two adjacent clusters in the tree is called a separator. The *tree-width* of a tree-decomposition is the maximum number of variables in a cluster minus 1 and the separator width is the maximum number of variables in any separator. The tree-width of a graph is the minimum tree-width over all its tree-decompositions and is identical to another graph parameter called induced-width [3].

*Cluster-tree elimination* (CTE) [2] is a message-passing algorithm over the clusters of a tree-decomposition, each associated with variable and function subsets. CTE computes two messages for each edge (one in each direction), from each node to its neighbors, in two passes, from the leaves to the root and from the root to the leaves. The message that cluster $u$ sends to cluster $v$, for the solution-counting task is as follows: The cluster multiplies all its own functions, with all the messages received from its neighbors excluding $v$ and then eliminates, by summation, all variables not in the separator between $u$ and $v$. The

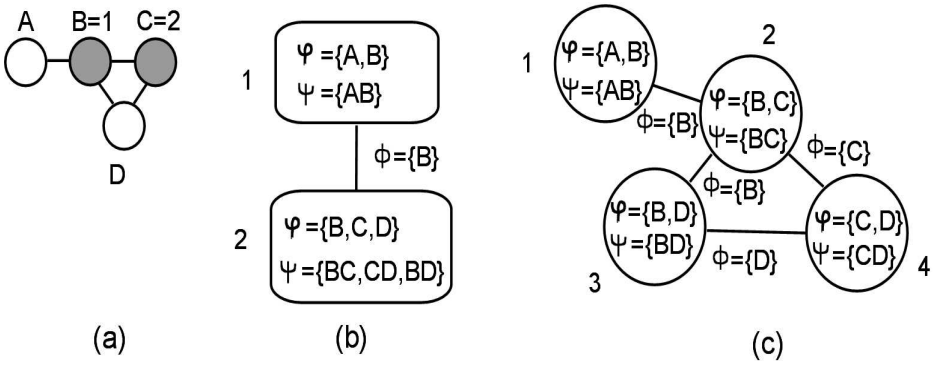


**Fig. 1.** (a) A graph coloring problem with B=1, C=2; (b) and (c) join-graph decompositions of the problem; (b) is also a tree-decomposition.

complexity of CTE is time exponential in the tree-width and space exponential in the maximum size of the separator width. CTE can output for each value and each variable the number of solutions extending this variable-value pair. For more details see [4, 2, 5].

*Iterative Join-Graph Propagation* (IJGP) [5] can be perceived as an iterative version of CTE. It applies the same message-passing to *join-graphs* rather than *join-trees*, iteratively. A join-graph is a decomposition of functions into a graph of clusters (rather than a tree) that satisfies the running intersection property. The IJGP class of algorithms generalizes loopy belief propagation. These algorithms are not guaranteed to converge, nor have bounded accuracies, however they have been demonstrated to be useful approximation methods for various belief networks, especially for probabilistic decoding [5]. While CTE is exact, and requires only 2 iterations, IJGP tends to improve its performance with additional iterations.

The size of clusters in a join-graph can be far smaller than the tree-width and is only restricted by the function's scopes. IJGP can be parameterized by $i$ which controls the cluster size in the join-graph, yielding a class of algorithms (IJGP($i$)) whose complexity is exponential in $i$, that allow a trade-off between accuracy and complexity. As $i$ increases, accuracy generally increases. When $i$ is big enough to allow a tree-structure, IJGP($i$) coincides with CTE and becomes exact.

*Example 1.* Figure 1(a) shows a graph coloring problem with 4 nodes $\{A, B, C, D\}$, each with a domain $\{1, 2, 3\}$, such that $B = 1$ and $C = 2$. In Figures 1(b) and 1(c) we have two join-graph decompositions of this problem, of which Figure 1(b) is also a tree-decomposition. This problem has 2 solutions, $\{A = 2, B = 1, C = 2, D = 3\}$ and $\{A = 3, B = 1, C = 2, D = 3\}$. Therefore, the (exact) solution counts are as follows: $SC(A = 2) = 1$, $SC(A = 3) = 1$, $SC(D = 3) = 2$. In order to compute the solution count for $SC(D = 3)$ based on the join-graph in Fig-

**Algorithm IJGP($i$)-SC**
**Input:** A join-graph decomposition $< JG, \chi, \psi, \phi >$, $JG = (V, E)$ is a join-graph for $CSP =< X, D, C >$. For every node $v \in V$ and edge $e \in E$, $\chi(v)$ is its set of variables, $\psi(v)$ is its set of functions and $\phi(e)$ is a set of edge labels. Each constraint $C(S_k)$ is represented by a cost function $f(S_k) = 1$ iff $S_k \in R_k$ and 0 otherwise. Instantiated variables $I$. Activation schedule $d = (u_1, v_1), \ldots, (u_{2*|E|}, v_{2*|E|})$.
**Output:** A solution count approximation for each singleton assignment $X = a$.

Denote by $h_{(u,v)}$ the message from vertex $u$ to $v$ in $JG$; $cluster(u) = \psi(u) \cup \{h_{(v,u)}|(v,u) \in E\}$; $cluster_v(u) = cluster(u)$, excluding message from $v$ to $u$. Let $h_{(u,v)}(j)$ be $h_{(u,v)}$ computed during the j-th iteration of IJGP. $\delta_{h_{(u,v)}}(j) = \sum_{\phi(u,v)}(h_{(u,v)}(j) - h_{(u,v)}(j-1))/|h_{(u,v)}(j)|$, $\Delta(j) = \sum_{d_l \in d}(\delta_{h_{d_l}}(j))/2 * |E|$.

1. **Process instantiated variables**:
   Assign relevant instantiated values to all $R_k \in \psi(u)$, $\chi(u) := \chi(u) - I, \ \forall u \in V$.
2. **Repeat iterations of IJGP** :
   - Along d, for each edge $(u_i, v_i)$ in the ordering,
   - compute $h_{(u_i,v_i)} = \alpha \sum_{elim(u_i,v_i)} \prod_{f \in cluster_{v_i}(u_i)} f$, where $\alpha$ is a normalization constant.
3. **until**:
   - Max number of iterations is exceeded, or
   - Distance $\Delta(j)$ is less than 0.1,
   - $\Delta(j) > \Delta(j-1)$.
4. **Compute solution counts**:
   For every $X_i \in X$ let $u$ be a vertex in $JG$ such that $X_i \in \chi(u)$. Compute $SC(X_i) = \alpha \sum_{\chi(u)-\{X_i\}}(\prod_{f \in cluster(u)} f)$, where $\alpha$ is a normalization constant.

**Fig. 2.** Algorithm IJGP($i$)-SC

ure 1(b), IJGP will compute (using node 2) $\sum_{B,C}(C_{BC} * C_{CD} * C_{BD} * h_{(1,2)}(B))$ where $C_{BC}$, $C_{CD}$, $C_{BD}$ are functions stored in node 2 of the join-tree and $h_{(1,2)}(B)$ is a message that node 1 sends to node 2. Note that this value of $SC(D = 3)$ since Figure 1(b) is a tree-decomposition. In order to compute the solution count for $SC(D = 3)$ based on the join-graph in Figure 1(c), IJGP will compute (using node 3) $\sum_B(C_{BD} * h_{(2,3)}(B) * h_{(4,3)}(D))$ where $C_{BD}$ is a function stored in node 3 of the join-graph and $h_{(2,3)}(B)$, $h_{(4,3)}(D)$ are messages that nodes 2 and 4 send to node 3. Note that this value is an approximation of the exact value. A formal description of the IJGP-SC algorithm is given next.

## 3 Approximating SC by IJGP

Our application of IJGP for solution counting is technically very similar to the application of IJGP for belief updating in Bayesian networks [5]. We will discuss some technical points here. As input, constraints are modelled by cost functions

that assign 1 to combinations of values that are allowed, and 0 to nogoods. IJGP-SC diverges (solution count values computed by IJGP-SC may get arbitrarily large) and thus the solution count values computed by IJGP would be trivial upper bounds on the exact values. To avoid double-point precision overflow we normalize all messages as they are computed. As a result, IJGP($i$)-SC will output, for each variable $X_i$, not solution count absolute values, but their ratios. For example, IJGP($i$)-SC($X = a$)=0.4 means that in approximately 40% of the solutions, $X = a$. When the solution count ratio computed by IJGP($i$)-SC is 0, the absolute value is 0 as well, and therefore the corresponding value $a$ of $X$ can be pruned. Note, however, that we don't need to know the solution counts very precisely because we use the counts only to create a variable and value ordering during backtracking search. All we want is that the approximated solution counts be sufficiently accurate as to yield a good value-ordering heuristic.

IJGP($i$)-SC is presented in Figure 2. As input it takes a join-graph and an activation schedule which specifies the order in which messages are computed. It executes a number of iterations. At the end of iteration $j$ we compute the distance $\Delta(j)$ between messages computed during iteration $j$ and the previous iteration $j-1$. We use this distance to decide whether IJGP($i$)-SC is converging. We stop IJGP($i$)-SC when either a predefined maximum number of iterations is exceeded (indicating that IJGP($i$)-SC is not converging), the distance $\Delta(j)$ is not decreasing (IJGP($i$)-SC is diverging), or $\Delta(j)$ is less than some predefined value (0.1) indicating that IJGP($i$)-SC has reached a fixed-point.

It is easy to see that solution count values 0 computed by IJGP($i$)-SC are correct [6], i.e. whenever IJGP($i$)-SC($X = a$)=0, the true solution count SC($X = a$)=0 as well. Consequently, when running IJGP(i) until convergence the algorithm also accomplishes $i$-consistency.

## 4 Algorithms

### 4.1 Backtracking with IJGP-SC

A formal description of a backtracking algorithm using solution counts computed by IJGP-SC(i) as a heuristic function is given in Figure 3. At each node in the search space it runs IJGP($i$)-SC and prunes domains of future variables whose approximated solution count is 0. The algorithm selects as next the variable with the smallest domain, breaking ties in favor of a variable having the largest single (approximated) solution count. The strength of IJGP($i$)-SC however is in guiding value ordering. The algorithm chooses a value having the largest approximated solution count (fraction).

We will refer to BB-IJGP-SC as IJGP-SC. In the course of carrying out experiments, we have implemented two versions of BB-IJGP-SC. The first version (called IJGP($i$)-SC, where $i = 2, 3, ...$), is a simple, general implementation that can be run on a CSP without restrictions on the constraint scope and can use any legal join-graph. $i$ denotes the bound on the cluster size used to generate the

**Procedure BB-IJGP($i$)-SC($\mathcal{G}$,$i$,$I$)**
**Input:** Join-graph $\mathcal{G}$, parameter $i$,set of instantiated variables $I$.
**Output:** A solution, or proof of inconsistency, or timeout.
**1. If** $I = X$, return 1/0 depending on whether $I$ satisfies all constraints.
**2. Run** IJGP-SC($i$); let $\{scX_j\}$ be the set of heuristic values computed by IJGP-SC($i$) for each variable $X_j \in X - I$.
**3. Prune** domains of uninstantiated variables, by removing values $x \in D(X_l)$ for which $scX_l(x) = 0$.
**4. Backtrack** If $D(X_l) = \emptyset$ for some variable $X_l$, return 0.
**5. Otherwise** let $X_j$ be the uninstantiated variable with the smallest domain: $X_j = argmin_{X_k \in X-I}|D(X_k)|$.
**6. Repeat** while $D(X_j) \neq \emptyset$
*i*. Let $x_k$ be the value of $X_j$ with the largest heuristic estimate: $x_k = argmax_{x_j \in D(X)} scX_j(x_j)$.
*ii*. Set $D(X) = D(X) - x_k$.
*iii*. Compute $csp = BB - IJGP - SC(\mathcal{G}, i, I \cup \{X_j = x_k\})$.
*iv*. If $csp$=1, return 1.
**7. Return** 0.

**Fig. 3.** Branch-and-Bound with IJGP($i$)-SC.

join-graph (thus $i$ controls the complexity as well as accuracy of IJGP($i$)-SC). Note that processing each node in the join-graph is exponential in $i$.

We also have a more efficient implementation (called IJGP-SC*) for the special case of $i = 2$. This version assumes binary constraints and uses the problem's dual-graph as its join-graph. The reason for developing this more specialized implementation is to be more competitive with MAC which is restricted to binary constraints, and because IJGP(i=2)-SC (the general version) was superior to higher values of $i$ time-wise.

### 4.2 The MAC algorithm

Maintaining arc consistency or the MAC algorithm [11] is one of the best performing algorithms for random binary CSPs that uses arc-consistency look-ahead. The performance of the basic MAC algorithm can be improved by using variable and value ordering heuristics during search. In our implementation[1], we have used the *dom/deg* heuristic for variable ordering while the min-conflicts ($MC$) heuristic for value ordering. This combination was shown to perform the best on random binary CSPs [1]. The *dom/deg* heuristic selects the next variable to be instantiated as the variable that has the smallest ratio between the size of the remaining domain and the degree of the variable. The MC heuristic chooses the value that removes the smallest number of values from the domains of the future variables.

[1] The implementation is based on Tudor's Hulubei's implementation available at *http://www.hulubei.net/tudor/csp*

### 4.3 Stochastic Local Search

We also compare against Stochastic Local Search (SLS), which, while incomplete, has been successfully applied to a wide range of automated reasoning problems. The SLS algorithm we use is a basic greedy search algorithm that uses a number of heuristics to improve its performance (see [7] for more details).

### 4.4 MBTE-SC/MC

On 100 variable random problems, we will compare IJGP($i$)-SC against MBTE($i$)-SC and MBTE($i$)-MC, using various $i$-bounds. MBTE($i$)-SC and MBTE($i$)-MC are backtracking algorithms that use a heuristic computed by the mini-cluster-tree elimination algorithm [2]. In case of MBTE($i$)-SC, the heuristic is Solution Counting, whereas in case of MBTE($i$)-MC, the heuristic is Min-Conflicts.

## 5 Experimental Results

We have performed three sets of experiments: (1) Comparison of scalability of IJGP-SC and MAC, (2) The effect of using different $i$-bounds on the performance of IJGP($i$)-SC, and (3) The effect of using different number of iterations.

All experiments use a cpu time bound and if a solution is not found within this time bound, we record a time-out. Note that only those instances that were solved by at least one algorithm within the time bound are considered as soluble instances. All experiments were carried out on a Pentium-2400 MHz PC with 2 GB of RAM running version 9.0 of the red-hat Linux operating system.

### 5.1 Problem Sets and Terminology

**Random CSPs** Random binary CSPs were generated using the parametric model $(N, K, C, T)$ called ModelB [12]. In this model, for a given $N$ and $K$, we select $C$ constraints uniformly at random from the available $N(N-1)/2$ binary constraints and then for each constraint we select exactly $T$ tuples (called as constraint tightness) as no-goods from the available $K^2$ tuples. The number of variables was varied between 100 and 2000. The domain size $K$ for all instances was 4 and the constraint tightness $T$ was also 4. This tightness is the same as in $not - equal$ constraints.

We will refer to the set of random CSPs having $n$ variables as the $n$-variable-set. For each $n$-variable-set, we tried to systematically (experimentally) locate the phase transition region by varying the number of constraints. However, the phase transition region could not be located for problems sets having more than 500 variables, due to the presence of large number of instances on which the algorithms (including MAC and SLS) did not terminate (in almost 3 days of cpu time). So for such problems, we chose to extrapolate the location of the phase transition region based on the statistics on smaller number of variables. We test the performance of our algorithms in the phase-transition region because the hardest csp instances appear there [12].

**Random 3-coloring problems** We generated random 3-coloring problems using Joseph Culberson's flat graph coloring generator. This generator generates graph coloring problems which are guaranteed to be 3-colorable. We experimented with 3-coloring problems having 100 and 200 vertices with 0 flatness. For these problems, we used a specialized graph coloring solver[2] to locate the settings at which hard problems occur with a high probability.

**Balanced Quasi-group completion problems with holes** We also generated random balanced Quasi-group completion problems with holes (balanced QWH) using the generator developed by Henry Kautz et al. [8]. Note that the problems generated by this generator are guaranteed to be consistent. We mapped these problems into a binary csp as follows: (1) The variables in the csp are the holes in a balanced QWH problem (2) The domain of each variable is the initial number of colors minus the colors already assigned to the row and column of the hole (3) The binary constraints in our csp formulation are the $not-equal$ constraints between two variables (or holes) that lie in the same column or row. We experimented with balanced QWH problems of order 18 and 20 respectively. For these problems, we used the results provided in [8] to locate the settings at which hard problems appear with a high probability.

The tables used in this section use the following terminology. The columns T and B give the time in seconds and number of backtracks respectively. Column TB gives the time-bound used while the column #Solved gives the number of instances solved by the algorithm in the given time-bound. Column V gives the number of variables and C gives the number of constraints. Note that an asterisk $(*)$ indicates that the median time was same as the time-bound used.

### 5.2 Scalability of IJGP-SC vs. MAC

First we compare IJGP-SC* against MAC in terms of scalability. In Table 1 we have results on random CSPs. For each $N$, we generated 200 instances at the 50% solubility point. Each row reports the statistics taken over soluble instances.

We observe from Table 1 that IJGP-SC* is superior to MAC in terms of scalability. We see that MAC is better than IJGP-SC* time-wise in the range $100-300$. However, as the number of variables increases above 400, MAC is inferior to IJGP-SC*. In particular, when $N \geq 500$, IJGP-SC* can solve many more problems than MAC. Also, IJGP-SC* is by far superior to MAC in terms of number of backtracks. A more detailed comparison between MAC and IJGP-SC* in terms of time and number of backtracks on individual instances in the 500-variable-set is shown in the two scatter plots of Figure 4.

Note that the results are likely based on a subset of the soluble instances that constitute easy instances at the 50% solubility point, because when $N > 1000$ we keep the time bound constant at 5 hours. However, the main point is that: given a time-bound IJGP-SC* solves more problems and faster than MAC as the problem size increases.

[2] solver and generator are available at http://www.cs.ualberta.ca/ joe/Coloring/

**Table 1.** Performance of MAC and IJGP-SC* on random problems

| | | | IJGP-SC* | | | MAC | | | |
|---|---|---|---|---|---|---|---|---|---|
| Statistics | V | C | T | B | #Solved | T | B | #Solved | TB |
| Average | 100 | 430 | 2.35 | 119.76 | | 0.32 | 62.26 | | |
| Median | 100 | 430 | 0.63 | 0 | 117 | 0.2 | 46 | 117 | 15 min |
| Average | 200 | 850 | 104.25 | 3014.59 | | 1.55 | 950.5 | | |
| Median | 200 | 850 | 3.496 | 42 | 102 | 0.715 | 461 | 102 | 1hr |
| Average | 300 | 1280 | 188.4 | 7882.2 | | 27.54 | 12409.4 | | |
| Median | 300 | 1280 | 74.5 | 352 | 98 | 9.1 | 4428 | 102 | 2 hr |
| Average | 400 | 1710 | 218.6 | 2363.7 | | 231.2 | 87456.7 | | |
| Median | 400 | 1710 | 113.4 | 360.5 | 91 | 143.8 | 57549 | 86 | 3 hr |
| Average | 500 | 2150 | 621.7 | 704.2 | | 1065.2 | 178921.2 | | |
| Median | 500 | 2150 | 431.62 | 389 | 72 | 653.2 | 122309 | 58 | 4hr |
| Average | 700 | 2970 | 1073.2 | 141.2 | | * | * | | |
| Median | 700 | 2970 | 732.5 | 89 | 67 | * | * | 3 | 5hr |
| Average | 1000 | 4250 | 1148.2 | 87.4 | | * | * | | |
| Median | 1000 | 4250 | 1073.6 | 83 | 41 | * | * | 2 | 5hr |
| Average | 1200 | 5100 | 1297.2 | 98.1 | | * | * | | |
| Median | 1200 | 5100 | 881.2 | 71 | 23 | * | * | 4 | 5hr |
| Average | 1500 | 6370 | 712.8 | 3.4 | | * | * | | |
| Median | 1500 | 6370 | 634.6 | 0 | 16 | * | * | 0 | 5hr |
| Average | 1700 | 7250 | 1756.2 | 145.2 | | * | * | | |
| Median | 1700 | 7250 | 1212.4 | 78 | 15 | * | * | 0 | 5hr |
| Average | 2000 | 8500 | 788.2 | 1.6 | | * | * | | |
| Median | 2000 | 8500 | 765.4 | 0 | 28 | * | * | 0 | 5hr |

**Table 2.** Performance of IJGP-SC* and MAC on 3-coloring problems

| | | | IJGP-SC* | | | MAC | | | |
|---|---|---|---|---|---|---|---|---|---|
| Statistics | V | C | T | B | #Solved | T | B | #Solved | TB |
| Average | 100 | 239 | 1.38 | 279 | | 0.18 | 40 | | |
| Median | 100 | 239 | 0.74 | 129 | 100 | 0.1 | 32 | 100 | 15min. |
| Average | 200 | 479 | 267.14 | 37763 | | 0.75 | 876 | | |
| Median | 200 | 479 | 125.34 | 24273 | 92 | 0.54 | 354 | 100 | 1hr. |

In order to test the performance of IJGP(2)-SC and MAC in the underconstrained region, we ran experiments on the 1000-variable-set. The number of constraints was varied between 4000 and 4100. The scatter plots in Figure 5 show the results on individual instances in the 1000-variable-set with a time-out of 5 hrs. Once again we observe that MAC is superior both time-wise and in number of backtracks.

For the Quasi-group completion problems with holes (balanced QWH), we see that (Table 3) IJGP-SC* is superior to MAC both in terms of time and number of backtracks. The statistics on each row in Table 3 is based on 100 instances.

**Table 3.** Performance of IJGP-SC* and MAC on balanced QWH problems

| | | | IJGP-SC* | | | MAC | | | |
|---|---|---|---|---|---|---|---|---|---|
| Statistics | Order | Holes | T | B | #Solved | T | B | #Solved | TB |
| Average | 18 | 150 | 463.06 | 17437.8 | | 954.74 | 1254880 | | |
| Median | 18 | 150 | 10.78 | 726 | 100 | 218.64 | 269919 | 91 | 1800s |
| Average | 18 | 158 | 178.22 | 6453 | | 714.56 | 975632 | | |
| Median | 18 | 158 | 13.21 | 934 | 100 | 219.45 | 248212 | 94 | 1800s |
| Average | 20 | 176 | 1214.5 | 11712.1 | | * | * | | |
| Median | 20 | 176 | 319.1 | 3326 | 81 | * | * | 22 | 1hr. |
| Average | 20 | 187 | 1176.49 | 9773.49 | | * | * | | |
| Median | 20 | 187 | 159.33 | 768 | 78 | * | * | 19 | 1hr. |

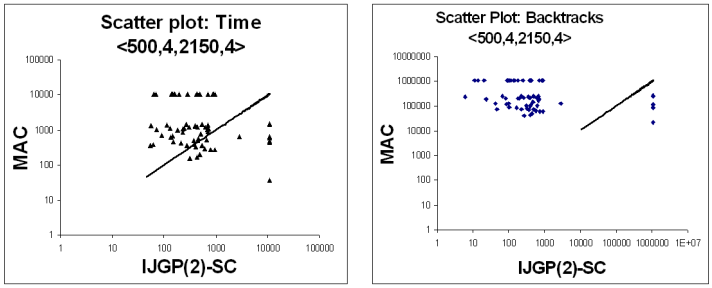


**Fig. 4.** IJGP-SC* vs. MAC for *soluble 500 variable problems with K=4, T=4, C=2150.*

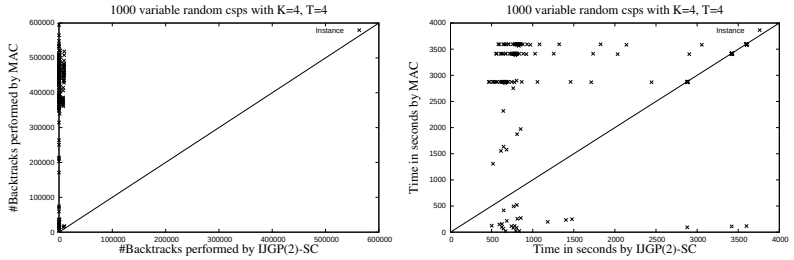


**Fig. 5.** IJGP(2)-SC vs. MAC for *soluble 1000 variable problems with K=4, T=4.*

We also observe that IJGP-SC* solves more problems than MAC within the specified time-bound.

On the other hand, our preliminary results on 3-coloring problems (see Table 2), show a contrasting picture in which MAC is superior both in terms of cpu-time and number of backtracks. We currently don't have a good explanation

for this phenomenon. We will address this question when we compare the effect of using different number of iterations on the performance of IJGP-SC*.

### 5.3 Effect of using different number of iterations

**Table 4.** The effect of number of iterations on IJGP-SC* on random csp instances

| | | | MAC | | IJGP-SC* 10 iterations | | IJGP-SC* 100 iterations | | IJGP-SC* 500 iterations | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Statistics | V | E | T | B | T | B | T | B | T | B | TB |
| Average | 100 | 430 | 0.32 | 62.26 | 2.35 | 119.76 | 5.88 | 3.23 | 26.3 | 2.5 | 15min. |
| Median | 100 | 430 | 0.2 | 46 | 0.63 | 0 | 5.91 | 0 | 28.21 | 0 | |
| Average | 200 | 850 | 1.55 | 950.5 | 104.25 | 3014.59 | 39.91 | 28.69 | 131.34 | 16.35 | 1hr. |
| Median | 200 | 850 | 0.72 | 461 | 3.5 | 42 | 26.43 | 0 | 128.59 | 0 | |

In order to determine the effect of iterations, we report results on (1) hard 100 and 200 vertex flat 3-coloring instances, (2) 100 and 200 variable random CSPs and (3) balanced Quasi-group completion problems with holes (balanced QWH) of order 18 and 20. The results are summarized in Tables 4 , 5 and 6. The statistics on each row in these tables is based on 100 random instances.

We can see that as we increase the number of iterations, the number of backtracks decreases. This is to be expected because as the number of iterations is increased the pruning power of IJGP-SC* increases (it becomes closer to full arc-consistency). We believe however that the decrease in backtracks is primarily due to a better value ordering heuristic. Our claim can be supported by comparing the performance of IJGP-SC* with MAC.

Dechter and Mateescu ([6]) proved that the pruning caused by IJGP-SC* is equivalent to arc-consistency when the algorithm is run until convergence. Thus, if we ignore the effect of value-ordering heuristic, MAC should prune better than IJGP-SC*. On the other hand, Tables 4 , 5 and 6 show that the number of backtracks performed by IJGP-SC* with 500 iterations is better than MAC. This suggests that the SC based value ordering heuristic used in IJGP-SC* is more powerful than MAC's min-conflict value-ordering heuristic.

Note that the time taken by IJGP-SC* is dependent on two factors (1) the number of backtracks made and (2) the number of iterations. The tables indicate the trade-off between the two. We see (Tables 4 and 5) that as we increase the number of iterations, the median number of backtracks required by IJGP-SC* for 10, 100 and 500 iterations is almost equal. On the other hand, from Table 6 we can see that IJGP-SC* requires significantly more iterations for 3-coloring problems than random CSPs and balanced QWH problems to obtain comparable level of accuracy (backtracks). This explains the poor performance of IJGP-SC* as compared to MAC for 3-coloring problems.

**Table 5.** The effect of number of iterations on IJGP-SC* for balanced QWH

| | | | MAC | | IJGP-SC* 10-iterations | | IJGP-SC* 100-iterations | | IJGP-SC* 500-iterations | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Statistics | Order | Holes | T | B | T | B | T | B | T | B | TB |
| Average | 18 | 150 | 954.74 | 1254880 | 463.06 | 17437.8 | 432.4 | 1400 | 1189.2 | 724.91 | 1800s |
| Median | 18 | 150 | 218.64 | 269919 | 10.78 | 726 | 170.53 | 622 | 688.92 | 345 | |
| Average | 18 | 158 | 714.56 | 975632 | 178.22 | 6453 | 213.56 | 744.3 | 533.4 | 387 | 1800s |
| Median | 18 | 158 | 219.45 | 248212 | 13.21 | 934 | 145.67 | 507 | 412.56 | 218 | |
| Average | 20 | 176 | * | * | 1214.5 | 11712.1 | 2080.43 | 2168.8 | 2908.4 | 542.3 | 1 hr. |
| Median | 20 | 176 | * | * | 319.1 | 3326 | 2706 | 1176 | 1845.2 | 471 | |
| Average | 20 | 187 | * | * | 1176.49 | 9773.49 | 1682.48 | 1454.8 | 2118.38 | 311.21 | 1 hr. |
| Median | 20 | 187 | * | * | 159.33 | 768 | 1058.73 | 198 | 1998 | 47 | |

**Table 6.** The effect of number of iterations on IJGP-SC* for 3-coloring problems

| | | | MAC | | IJGP-SC* 10 iterations | | IJGP-SC* 100 iterations | | IJGP-SC* 500 iterations | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Statistics | V | E | T | B | T | B | T | B | T | B | TB |
| Average | 100 | 239 | 0.18 | 40 | 1.38 | 279 | 2.21 | 47 | 10.41 | 7 | 15 min. |
| Median | 100 | 239 | 0.1 | 32 | 0.74 | 129 | 1.39 | 31 | 7.13 | 0 | |
| Average | 200 | 479 | 0.75 | 876 | 267.14 | 37763 | 27.9 | 141 | 53.72 | 39 | 1hr. |
| Median | 200 | 479 | 0.54 | 354 | 125.34 | 24273 | 23.99 | 108 | 39.85 | 13 | |

### 5.4 Effect of using different $i$-bounds

To determine the effect of varying the $i$-bound, we experimented with $N = 100$ random problems and generated 200 instances each with 420, 430, 440 and 450 constraints. We have decomposed our results into two subsets, the first consisting of only the soluble instances while the other consisting of only insoluble instances (Table 7). It is evident that algorithms with $i$-bound 2 dominate their counterparts with higher $i$-bounds in terms of cpu time. In general, for a given $i$-bound, IJGP($i$)-SC was better than MBTE($i$)-SC which was in turn better than MBTE($i$)-MC in terms of cpu time and the number of backtracks. As expected the number of backtracks decreases as $i$-bound increases.

### 5.5 Performance of SLS

We ran SLS on random problems from $N = 100$(Table 7) until $N = 1000$. We saw that SLS was competitive with MAC and IJGP-SC at problem sizes $N = 100$-500. However at $N = 1000$ SLS failed to solve any problems. We admit that our implementation of SLS may not be competitive with the best local search algorithms.

**Table 7.** Median Time and Backtracks required by various algorithms on the 100-variable-set, $i$=$i$-bound used. The quantity in the bracket indicates the number of instances on which the results are based on. We have used use IJGP($i$)-SC, the non-optimized version.

| | IJGP(i)-SC | | | MBTE(i)-SC | | | MBTE(i)-MC | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| C | i=2 | i=3 | i=4 | i=2 | i=3 | i=4 | i=2 | i=3 | i=4 | SLS | MAC |
| | **Time for soluble instances** | | | | | | | | | | |
| 420.0(163) | 1.2 | 1.9 | 3.9 | 4.1 | 5.6 | 9.9 | 4.7 | 4.5 | 5.9 | 0.2 | 0.2 |
| 430.0(109) | 1.3 | 2.1 | 4.2 | 14.4 | 15.8 | 16.1 | 13.0 | 7.1 | 15.8 | 0.2 | 0.2 |
| 440.0(85) | 1.4 | 2.1 | 7.0 | 29.0 | 30.2 | 20.9 | 9.9 | 20.4 | 20.7 | 0.3 | 0.4 |
| 450.0(43) | 1.4 | 2.2 | 11.2 | 9.3 | 38.9 | 287.2 | 134.9 | 57.8 | 89.5 | 0.7 | 0.6 |
| | **Backtracks for soluble instances** | | | | | | | | | | |
| 420.0(163) | 0.0 | 0.0 | 0.0 | 111.0 | 103.0 | 115.0 | 83.0 | 29.0 | 10.0 | | 44.0 |
| 430.0(109) | 2.0 | 3.0 | 2.0 | 261.0 | 267.0 | 155.0 | 243.0 | 62.0 | 92.0 | | 40.0 |
| 440.0(85) | 2.0 | 39.0 | 132.0 | 485.5 | 331.0 | 334.5 | 415.0 | 48.5 | 140.0 | | 73.0 |
| 450.0(43) | 0.0 | 0.0 | 69.0 | 178.0 | 400.0 | 1550.0 | 1673.0 | 565.0 | 531.0 | | 58.0 |
| | **Time for insoluble instances** | | | | | | | | | | |
| 420.0(37) | 76.6 | 139.9 | 168.7 | 392.6 | 327.9 | 398.2 | 398.4 | 223.2 | 276.4 | | 0.4 |
| 430.0(91) | 44.0 | 62.7 | 107.9 | 276.6 | 231.7 | 306.7 | 148.0 | 134.1 | 171.1 | | 0.7 |
| 440.0(115) | 26.2 | 43.4 | 80.1 | 211.5 | 232.6 | 311.8 | 164.1 | 141.4 | 161.3 | | 0.4 |
| 450.0(157) | 30.2 | 38.2 | 90.4 | 230.6 | 231.6 | 327.3 | 132.0 | 124.4 | 156.5 | | 0.4 |
| | **Backtracks for insoluble instances** | | | | | | | | | | |
| 420.0(37) | 1501.0 | 1718.5 | 1415.0 | 4870.0 | 2870.0 | 2211.0 | 4463.5 | 2577.0 | 1764.0 | | 92.5 |
| 430.0(91) | 944.0 | 762.5 | 792.5 | 2818.5 | 1829.5 | 1415.0 | 2038.0 | 1599.0 | 1350.0 | | 61.0 |
| 440.0(115) | 565.0 | 549.0 | 617.0 | 2332.0 | 1861.0 | 1549.0 | 2068.0 | 1460.0 | 1048.0 | | 67.0 |
| 450.0(157) | 573.0 | 420.0 | 481.0 | 2090.0 | 1622.0 | 1429.0 | 1756.0 | 1131.0 | 918.0 | | 52.0 |

## 6 Related Work and Conclusions

The paper presents a new look-ahead scheme based on the Iterative Join-Graph Propagation (IJGP) approximation of the solution counting task. We compare a simple backtracking algorithm using the IJGP-SC heuristic against MAC and SLS on random CSPs, graph coloring problems and Quasi-group completion problems. We show that the counting heuristic produces a highly focused search that has as much as orders of magnitude fewer backtracks than MAC. In our experiments we use IJGP-SC mostly as a value ordering heuristic, while the strength of MAC lies in domain pruning.

Horsch and Havens [9] have studied the task of computing solution probabilities. Their algorithm, called Probabilistic Arc Consistency (pAC), is a generalization of arc consistency and approximates solution counts for each singleton assignment. We believe our approach of employing IJGP for computing solution counts is much more general, allowing a tradeoff between accuracy and complexity. Our experimental work is far more extensive; [9] experiments with problems up to 70 variables and finds that while pAC greatly reduces the number of backtracks competing algorithms are often still superior time-wise; we solve problems

with up to 2000 variables, and show that often, especially for large problems, IJGP-SC is superior to some of the best competing algorithms.

The main result of the paper is in demonstrating the power of counting approximation by IJGP as a value-ordering heuristic. Specifically, we showed that backtracking with IJGP-SC is superior to MAC in terms of scalability on random CSPs and Quasi-group completion problems, while (based on preliminary results) on graph coloring MAC is superior to IJGP-SC. On random CSPs, MAC is superior to IJGP-SC when the problem size is small. However, as the problem size grows to $N = 500$, IJGP-SC is better than MAC in terms of CPU time. As $N$ grows further, IJGP-SC can solve more problems and in less CPU time.

**Acknowledgments**

This work was supported in part by the NSF grant IIS-0086529 and the MURI ONR award N00014-00-1-0617.

## References

[1] Christian Bessiere and Jean-Charles Regin. MAC and combined heuristics: Two reasons to forsake FC (and CBJ?) on hard problems. In *CP'96*, 1996.

[2] R. Dechter, K. Kask, and J. Larrosa. A general scheme for multiple lower-bound computation in constraint optimization. *CP-2001*, 2001.

[3] R. Dechter and J. Pearl. Tree clustering for constraint networks. *Artificial Intelligence*, pages 353–366, 1989.

[4] Rina Dechter. *Constraint Processing*. Morgan Kaufmann, 2003.

[5] Rina Dechter, Kalev Kask, and Robert Mateescu. Iterative join graph propagation. In *UAI '02*, pages 128–136. Morgan Kaufmann, August 2002.

[6] Rina Dechter and Robert Mateescu. A simple insight into iterative belief propagation's success. *UAI-2003*, 2003.

[7] K. Kask and R. Dechter. Gsat and local consistency. In *IJCAI-95*, 1995.

[8] Henry A. Kautz, Yongshao Ruan, Dimitris Achlioptas, Carla P. Gomes, Bart Selman, and Mark E. Stickel. Balance and filtering in structured satisfiable problems. In *IJCAI*, pages 351–358, 2001.

[9] Horsch Michael and Havens Bill. Probabilistic arc consistency: A connection between constraint reasoning and probabilistic reasoning. In *UAI-2000*, pages 282–290, 2000.

[10] J. Pearl. *Probabilistic Reasoning in Intelligent Systems*. Morgan Kaufmann, 1988.

[11] Daniel Sabin and Eugene C. Freuder. Understanding and improving the MAC algorithm. In *CP (1997)*, pages 167–181, 1997.

[12] Barbara Smith. The phase transition in constraint satisfaction problems: A CLoser look at the mushy region. In *Proceedings ECAI'94*, 1994.