New Look-Ahead Schemes for Constraint Satisfaction

Kalev Kask, Rina Dechter and Vibhav Gogate
Department of Information and Computer Science
University of California, Irvine, CA 92697-3425

Abstract

This paper presents new look-ahead schemes for backtracking search when solving constraint satisfaction problems. The look-ahead schemes compute a heuristic for value ordering and domain pruning, which influences variable orderings at each node in the search space.

As a basis for a heuristic, we investigate two tasks, both harder than the CSP task. The first is finding the solution with min-number of conflicts. The second is counting solutions. Clearly each of these tasks also finds a solution to the CSP problem, if one exists, or decides that the problem is inconsistent. Our plan is to use approximations of these more complex tasks as heuristics for guiding search for a solution of a CSP task. In particular, we investigate two recent partition-based strategies that approximate variable elimination algorithms, Mini-Bucket-Tree Elimination and Iterative Join-Graph Propagation (ijgp). The latter belong to the class of belief propagation algorithm that attracted substantial interest due to their surprising success for probabilistic inference. Our preliminary empirical evaluation is very encouraging, demonstrating that the counting-based heuristic approximated by by IJGP yields a very focused search even for hard problems.

Introduction

We investigate two cost functions for modelling constraint satisfaction problems. One as an optimization, min-conflict (MC) and the other as solution-counting (SC). When approximating variable elimination algorithms on each of these formulations, we obtain heuristic functions that can be used to guide backtracking search algorithms. For the min-conflict formulation, each constraint is modelled by a cost function that assigns 0 to each allowed tuple and 1 to unallowed tuples and the overall cost is the sum of all cost functions. An assignment is a solution when its cost is 0.

Within the solution-count formulation, each constraint is modelled by a function that assigns 1 to allowed tuples and 0 otherwise. The overall cost is the product of all individual functions. An assignment is a solution when its cost is 1. The target is to find the number of solutions, namely the number of assignments having cost 1.

Both of these tasks can be solved exactly by variable elimination algorithms. However the complexity of these algorithm are two high to be practical and approximations are necessary. In order to approximate variable elimination, we apply two basic approximation schemes, Mini-Bucket-Tree Elimination (MBTE) (Dechter, Kask, & Larrosa 2001) and Iterative Join-Graph Propagation (IJGP) (Dechter, Kask, & Mateescu 2002) to each of the two formalisms, yielding heuristic functions. The heuristics derived using MBTE can provide a lower-bound on the min-conflict function and an upper-bound in the case of solution counting[1]. The heuristics generated using IJGP provide no guarantee, but were shown to work very well sometimes for probabilistic inference tasks.

We incorporate these these heuristics, computed by MBTE/IJGP on MC/SC tasks within backtracking search and compare the resulting backtracking algorithms against MAC (Sabin & Freuder 1997), one of the most powerful lookahead methods, and against SLS (Stochastic Local Search).

Our results are very promising. We show that, on hard random problem instances on the phase transition, the SC model yields overall stronger heuristics than the MC model. In particular, IJGP computing SC yields a very focused search with relatively few deadends. We show that this new algorithm, backtracking with solution count heuristic computed by IJGP (IJGP-SC), is more scalable than MAC/SLS - it is inferior to MAC/SLS on small problems, but as the problem size grows, the performance of IJGP-SC improves relative to MAC/SLS, and on the largest problems we tried IJGP-SC outperforms both MAC and SLS. These results are significant because our base algorithm is naive backtracking that is not enhanced by either backjumping or learning, but equipped with a single look-ahead heuristics. Finally, we believe that our implementation can be optimized to yield at least an order of magnitude speed-

[1] If we normalize messages computed by MBTE-SC, we get approximations of fractions of solutions, instead of upper bounds on solution counts.

up.

Preliminaries

DEFINITION **0.1 (constraint satisfaction problem)** *A* Constraint Network (CN) *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 each constraint is satisfied, namely $\forall\, C_i \in C\ x_{S_i} \in R_i$. 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.*

As noted a constraint satisfaction problem can be solved through a minimization problem, minimizing the number of conflict cost function, through the task of finding all solutions. Both of these tasks can be solved exactly by inference algorithms defined over a tree-decomposition of the problem specification, developed for constraint network (Dechter & Pearl 1989) and for belief networks (Lauritzen & Spiegelhalter 1988). Intuitively, a tree-decomposition takes a collection of functions and partition them into a tree of clusters. The cluster tree is often called a *join-tree* or a *junction tree*. We overview formally the notion of a tree-decomposition and a message-passing algorithm over the tree. The combined description is similar to the Shafer-Shenoy variant of junction-tree algorithm (Shafer & Shenoy 1990).

We use a recent formalization of tree-decomposition given by (Gottlob, Leone, & Scarello 1999).

DEFINITION **0.2 (cluster-tree decompositions)** *Let $CSP =< X, D, C >$ be a constraint satisfaction problem. A* cluster-tree decomposition *for CSP is a triple $D =< T, \chi, \psi >$, where $T = (V, E)$ is a tree, and χ and ψ are labelling functions which associate with each vertex $v \in V$ two sets, variable label $\chi(v) \subseteq X$ and function label $\psi(v) \subseteq C$.*

1. *For each function $C_i \in C$, there is* exactly *one vertex $v \in V$ such that $C_i \in \psi(v)$, and $S_i \subseteq \chi(v)$.*
2. *For each variable $X_i \in X$, the set $\{v \in V | X_i \in \chi(v)\}$ induces a connected subtree of T. The connectedness requirement is also called the running intersection property.*

Let (u, v) be an edge of a tree-decomposition, the separator *of u and v is defined as $sep(u, v) = \chi(u) \cap \chi(v)$; the* eliminator *of u and v is defined as $elim(u, v) = \chi(u) - sep(u, v)$.*

DEFINITION **0.3 (tree-width, induced-width)** *The tree-width of a tree-decomposition is $tw = \max_{v \in V} |\chi(v)| - 1$, and its maximum separator size is $s = \max_{(u,v) \in E} |sep(u, v)|$. The tree-width of a graph is the minimum tree-width over all its tree-decompositions and it can be shown to be identical to the graph's induced-width (Dechter & Pearl 1989).*

Cluster-tree elimination (CTE) (Dechter, Kask, & Larrosa 2001) is a message-passing algorithm on a tree-decomposition, the nodes of which are called clusters, each associated with variable and function subsets (their labels). 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 min-conflict task is as follows: The cluster sums all its own functions (in its label), with all the messages received from its neighbors excluding v and then minimize the resulting function relative to the eliminators between u and v. This yields a message defined on the separator between u and v. For solution counts the messages are the same except summation is replaced with a product and minimization with summation.

The complexity of CTE is time exponential in the maximum size of variable subsets and space exponential in the maximum size of the separators. Join-tree and junction-tree algorithms for constraint and belief networks are instances of CTE. CTE is an exact algorithm for computing either the min conflict solution or the solution counts for the whole problem. Moreover the algorithm can output for each value and each variable the min number of conflicts, or the number of solutions extending this variable-value pair, for the respective tasks. For more details see (Dechter, Kask, & Larrosa 2001; Dechter, Kask, & Mateescu 2002). Bucket-tree-elimination algorithm, BTE, is the special case when the underlying tree-decomposition is a bucket-tree. A bucket-tree is the structure associated with bucket-elimination algorithms.

Mini-cluster-tree elimination (MCTE(i)) (Dechter, Kask, & Larrosa 2001) approximates CTE by using the partitioning idea - when computing a message from cluster u to cluster v, cluster u is partitioned into mini-clusters whose function scopes has at most i variables, each of which is processed separately by the same cluster computation described above, resulting in a set of messages which are sent to cluster v. MCTE(i) computes a bound on the exact value (a lower bound in case of a minimization problem, an upper bound in case of a maximization problem) and allows a trade-off between accuracy and complexity controlled by i. Its space and time complexity is exponential in the input parameter i that also bounds the scope of the messages.

Iterative Join-Graph Propagation (IJGP) (Dechter, Kask, & Mateescu 2002) can be perceived as an iterative version of the cluster-tree elimination algorithm that applies the message-passing to *join-graphs* rather than *join-trees*. A join-graph is a decomposition of functions into clusters that interact in a graph manner, rather than a tree-manner. namely conditions 1 and 2 should be satisfied by the clusters in the graph. Therefore there are two major differences between join-trees and join-graphs:

1. Unlike a join-tree decomposition, which is defined on a tree T, a join-graph is defined on a graph $G = (V, E)$.
2. Join-graph satisfies the same two conditions 0.2 as join-tree. In addition, each edge $e \in E$ of the join-graph has a labelling $\phi(e) \subseteq sep(e)$, such that, for any variable X_i, the set of edges $\{e|X_i \in \phi(e)\}$ is a tree.

Join-graph propagation algorithm sends the same CTE messages to their neighbors, except that their messages are computed on the edge labelling ϕ, instead of separators. This class of algorithms generalizes loopy belief propagation that demonstrated a useful approximation method for various belief networks, especially for probabilistic decoding (Dechter, Kask, & Mateescu 2002).

An important difference between MCTE and IJGP is that IJGP is iterative and can improve its performance with additional iterations. Similarly to MCTE(i), IJGP can be parameterized by i which controls the cluster size in the join-graph, yielding a class of algorithms (IJGP(i)) that allow a trade-off between accuracy and complexity. As i increase accuracy generally increases. When i is big enough to allow a tree-structure IJGP(i) coincide with CTE and become exact.

Min-Cost vs. Solution-Count

As noted before, we can express the relation R_i as a cost function $C_i(X_{i1} = x_{i1}, ..., X_{ik} = x_{ik}) = 0$ if $(x_{i1}, ..., x_{ik}) \in R_i$, and 1 otherwise. We call this a *Min-Conflict* (MC) model of the CSP. The objective function is the sum of all cost functions. The CSP problem is to find an assignment for which the cost function is 0. We will focus on a related task - given a partial assignment E, compute, for each value a of each uninstantiated variable X_i, the number of constraints violated in the extension of the assignment $E \cup \{X_i = a\}$ that has a least number of conflicts. As an optimization (minimization) task, it is NP-hard. However, an approximation of this can serve as a heuristic function guiding the Branch-and-Bound search for finding a solution.

Alternatively, we may also express the relation R_i as a cost function $C_i(X_{i1} = x_{i1}, ..., X_{ik} = x_{ik}) = 1$ if $(x_{i1}, ..., x_{ik}) \in R_i$, and 0 otherwise. We call this a *Solution Counting* (SC) model of the CSP. The objective function is the product of all cost functions. The CSP problem is to find an assignment for which the objective function is 1. However, we will focus on a harder task within this representation - given a partial assignment E, compute, for each value a of each uninstantiated variable X_i, the number of solutions that agrees with $E \cup \{X_i = a\}$. As a counting problem, it is #P-complete. However, an approximation of this task can serve as a heuristic function guiding a backtracking search algorithms for finding a solution.

Approximating by MBTE the Min-Conflict and Solution-Count

Mini-Bucket-Tree Elimination (Dechter, Kask, & Larrosa 2001), or more generally, the Mini-Cluster-Tree Elimination, can be applied to approximate both the Min-Cost and Solution-Count heuristics in a straightforward manner. MBTE applied to the Min-Conflict model (MBTE(i)-MC) computes, given a partial assignment E, for each value a of each uninstantiated variable X, a *lower bound* on the cost of the best extension of the given partial assignment $E \cup \{X = a\}$. When MBTE is applied to the Solution-Count model (MBTE(i)-SC) it computes, given a partial assignment E, for each value a of each uninstantiated variable X_i, an approximation on the number of solutions that agree with $E \cup \{X_i = a\}$. Note that the i-bound can be used to control the accuracy and complexity of this approximation scheme.

The respective approximated values computed by MBTE(i)-MC and MBTE(i)-SC can also be used for domain pruning. It can be shown that the SC heuristic pruning power, computed by MBTE(i), is equivalent to the MC heuristic pruning power. More precisely,

Proposition.[Equivalence of SC and MC for pruning] The pruning power of MBTE(i)-SC is equal to that of MBTE(i)-MC. Namely, if the partitioning structure used by MBTE-MC and MBTE-SC is identical, then MBTE(i)-MC(E) > 0 exactly when MBTE(i)-SC(E) = 0, where E is a set of instantiated variables. That is, MBTE(i)-MC allows pruning (> 0) iff MBTE(i)-SC allows pruning (= 0).

Overall it seems that SC is superior to MC because it allows not only pruning of domains but also value ordering. Consequently, unless MBTE-MC can be implemented more efficiently MBTE-SC is superior overall, as will be validated by our experiments.

Approximating Solution-Count by IJGP

We will also use IJGP for approximating solution counts for each singleton assignment $X_i = a$. IJGP for solution counting is technically very similar to IJGP for the computation of belief in Bayesian networks (Dechter, Kask, & Mateescu 2002). For completeness, a formal description of IJGP(i)-SC is given in Figure 1.

IJGP(i)-SC takes, as input, a join-graph and an activation schedule which specifies the order in which messages are computed. It executes a number of IJGP iterations. The algorithm sends the same messages between

neighboring clusters as CTE using the SC model. At the end of iteration j it computes the distance $\Delta(j)$ between the messages computed during iteration j and the previous iteration $j-1$. The algorithm uses this distance to decide whether IJGP(i)-SC is converging. The algorithm stops 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. Some of the more significant technical points are:

- As input, constraints are modelled by cost functions which 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 can be shown to be trivial upper bounds on the exact values. Also, in practice IJGP may suffer from double-point precision overflow. To avoid that, we will normalize all messages as they are computed. As a result, IJGP(i)-SC will compute, for each variable X_i, not solution counts but their ratios. For example, IJGP(i)-SC($X = a$)=0.4 means that in approximately 40% of the solutions, $X = a$. Therefore the approximated solution counts are no longer upper bounds. Still, when the solution count (ratio) computed by IJGP(i)-SC is 0, the true value is 0 as well, and therefore the corresponding value a of X can be pruned.
- Note, however, that since we use the solution counts only to create a variable and value ordering, we don't need to know the counts precisely. All we want is that the approximated solution counts be accurate enough to yield a value ordering as close as possible to that induced by the exact solution counts.

As we commented earlier, it is easy to see that IJGP-SC zero values are sound.

THEOREM **0.1 (Correctness of IJGP-SC for 0's)** *(Dechter & Mateescu 2003) Whenever IJGP(i)-SC($X_i = a$)=0, the exact solution count SC($X_i = a$)=0 as well.*

Approximating Min-Cost by IJGP

We do not use IJGP for approximating the Min-Conflict heuristic for the following reason. IJGP is an iterative algorithm that runs on a join-graph, not join-tree. Therefore, as with IJGP-SC, IJGP-MC diverges (messages computed by IJGP-MC would get arbitrarily large). However, unlike the case with IJGP-SC, this means that the values computed by IJGP(i)-MC are not bounds (in case of the MC task we need lower bounds), and therefore their usefulness as a heuristic is questionable. We could normalize the messages, like we did with IJGP(i)-SC, but unlike in case of IJGP(i)-SC, heuristic value 0 cannot be used for pruning, and heuristic values greater than 0 are not guaranteed to be lower bounds, so they also cannot be used for pruning as well.

Backtracking algorithm with guiding heuristics

Backtracking with MBTE-MC

BB-MBTE(i)-MC is a simple Branch-and-Bound algorithm for constraint satisfaction that uses approximated min-conflict computed by MBTE(i)-MC as a heuristic function. At each point in the search space it applies MBTE(i)-MC and prunes domains of variables that have a min-cost greater than 0. When choosing the next variable to instantiate, it chooses a variable with the smallest domain. Unfortunately, MBTE(i)-MC does not allow dynamic value ordering because all values are either pruned or have a heuristic value 0.

Backtracking with IJGP-SC/MBTE-SC

BB-IJGP(i)-SC uses approximated solution counting computed by IJGP(i)-SC as a heuristic function for guiding backtracking search. At each node in the search space it computes IJGP(i)-SC and prunes domains of variables that have a solution count 0 (theorem 0.1). When choosing the next variable to instantiate, it chooses a variable with the smallest domain computed at this point using by IJGP(i)-SC, breaking ties by choosing a variable with the largest single solution count. The strength of IJGP(i)-SC is in value ordering. It chooses a value with the largest approximated solution count (fraction). BB-MBTE(i)-SC works like BB-IJGP(i)-SC except that the heuristic function is computed by MBTE(i)-SC.

Competing algorithms

Stochastic Local Search

For comparison, we also compare against one of the most successful greedy local search schemes for CSP. The stochastic local search algorithm we use is a basic greedy search algorithm that uses three heuristics to improve its performance:

1. Constraint weighting ((Morris 1993)). When the algorithm gets stuck is a local minima, it re-weights constraints, which has the effect of changing the search space, eliminating the local minima.
2. Dynamic restarts ((Kask & Dechter August 1995). When one try is executed, the program automatically determines when to quit the try and restart.
3. Tie-braking according to historic information ((Gent & Walsh 1993). When more than one flip yield the same change in the objective function, choose the one that was used the longest ago.

SLS algorithms, while incomplete, have been successfully applied to wide range of automated reasoning

Algorithm IJGP(i)-SC
Input: A graph decomposition $< JG, \chi, \psi >, JG = (V, E)$ for $CSP =< X, D, C >$. Each constraint $C(S_k)$ is represented by a cost function $f(S_k) = 1$ iff $S_k \in R_k$ and 0 otherwise. Evidence 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. $\delta_{h_{(u,v)}}(j) = \sum_{sep(u,v)}(h_{(u,v)}(j) - h_{(u,v)}(j-1))/|h_{(u,v)}(j)|$, $\Delta(j) = \sum_{d_l \in d}(\delta_{h_{d_l}}(j))/2 * |E|$.

1. **Process observed variables**:
   Assign relevant evidence 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 $h_{(u_i,v_i)} = \alpha \sum_{elim(u_i,v_i)} \prod_{f \in cluster_{v_i}(u_i)} f$
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 $SC(X_i) = \alpha \sum_{\chi(u)-\{X_i\}} (\prod_{f \in cluster(u)} f)$.

Figure 1: Algorithm IJGP(i)-SC

problems. They have been more scalable than systematic complete methods, especially on random problems, and thus are the main competing algorithm.

The MAC algorithm

Maintaining arc consistency or the MAC algorithm (Sabin & Freuder 1997) is one of the best performing algorithm for random binary CSPs that uses arc-consistency look-ahead. It differs from chronological backtracking in the following three aspects:

1. The constraint network is initially made arc-consistent.
2. Every time a variable X is instantiated to a value v, the effects are propagated in the constraint network by treating the domain of X as $\{v\}$.
3. Every time an instantiation of a variable X to a value v is refuted, the network is made arc-consistent by removing v from the domain of X.

The performance of the basic MAC algorithm can be improved by using variable and value ordering heuristics during search. In our implementation[2], we have used the dom/deg heuristic for variable ordering while the min-conflicts or the MC heuristic for value ordering. This combination was shown to perform the best on random binary CSPs (Bessiere & Regin 1996). 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. Our implementation uses the AC-7 algorithm (Christian Bessière 1999) for maintaining arc-consistency.

[2]The implementation is based on Tudor's Hulubei's implementation available at *http://www.hulubei.net/tudor/csp*. We have augmented this implementation to include dom/deg and MC heuristics

Experimental Results

In this section, we report on experiments that examined the effect of adding our new lookahead schemes on the performance of the chronological backtracking algorithm. Because all the schemes are solution driven, we focused more on the soluble CSPs while our results on the insoluble instances are incomplete. We compare the performance of our algorithms by using parameters like the the cpu time and the number of backtracks. We also compare our algorithms with our implementations of the MAC algorithm and the stochastic local search algorithm described in previous sections.

Problem Sets

So far we have experimented with randomly generated binary CSPs using ModelB (MacIntyre *et al.*

1998). ModelB can be defined by a standard 4-tuple $< N, K, C, T >$ where N is the number of variables, K is the domain size of each variable, C is the number of pairs of variables that are involved in a constraint and T is the number of pairs of values that are inconsistent for each constraint. When constructing a constraint graph using this model, we select C constraints uniformly at random from the possible $N(N-1)/2$ constraints. Then, for each constraint, we select uniformly at random T pairs of values as nogoods from the possible K^2 pairs.

We generated four sets of random problem instances in the phase transition region having 100, 200, 500 and 1000 variables respectively using ModelB. Note that researchers have indicated (Cheeseman, Kanefsky, & Taylor 1991; Smith 1994) that the hardest CSP instances appear in the phase transition region and also that ModelB generates harder problem instances than other models at the phase transition for binary CSPs (Smith 1994). The domain size for all instances was 4 and the constraint tightness was 4. These instances are typical of graph coloring problems although they are not as random. For each set, we systematically located the phase transition region by varying the number of constraints in increments of 5. We then selected four points in this range that correspond to a particular constraint density value and generated a number of random problem instances for each selected point.

Results and Discussion

We will refer to the chronological backtracking algorithm that uses the look-ahead schemes IJGP(i)-SC, MBTE(i)-SC and MBTE(i)-MC as IJGP(i)-SC, MBTE(i)-SC and MBTE(i)-MC respectively where i is the i-bound used. Note that processing each search node is typically exponential in the i-bound used and so we experimented only with i-bounds of 2, 3 and 4. Also, note that we have fixed the maximum number of iterations to 10 (see step 3 Figure 1). This choice was rather arbitrary. All the experiments reported in this paper use a cpu time bound i.e. 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 while those instances that were proved to be insoluble by at least one algorithm within the time bound are considered as insoluble instances.

Experiments on the 100-variable-set All experiments on the 100-variable-set were run on a Pentium-2400 MHz machine with a 1000 MB RAM running version 9.0 of the red-hat Linux operating system. The time bound was 500 seconds. We generated 200 instances each with 420, 430, 440 and 450 constraints and ran all the algorithms on these instances. We observed that the instances with 420 and 430 constraints lie in the under-constrained region of the phase transition while the instances with 440 and 450 constraints lie in the over-constrained region of the phase transition.

We observed that the large variation in cpu time and the number of backtracks was very dependent on whether the instance was soluble or not. So we have decomposed our results into two subsets, the first consisting of only the soluble instances while the other consisting of only insoluble instances. Tables 1 shows the results for the soluble 100 variable instances. 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. While it is not reflected in the time measure, the number of backtracks required by our algorithms is significantly lower than MAC(table 1).

Table 1 shows the results for insoluble instances. From this table, we can see that for insoluble instances in the 100-variable-set, IJGP(2)-SC performs better than MBTE(2)-SC which in turn is better than MBTE(2)-MC both in terms of cpu time and the number of backtracks. MAC is the best performing algorithm on insoluble CSPs both in terms of cpu time and the number of backtracks. Dechter and Mateescu (Dechter & Mateescu 2003) proved that IJGP(2) is identical to arc-consistency when run until convergence. However, we run IJGP(2)-SC for only 10 iterations and thus IJGP(2)-SC prunes less values as compared to the MAC algorithm. Moreover, MAC maintains arc-consistency when a variable is instantiated and also when a value is refuted while we run IJGP(2) only when a variable is instantiated. Less pruning means more nodes explored and more backtracks which is why the number of backtracks by IJGP(2)-SC is more than MAC on insoluble CSPs. We believe that the poor performance of algorithms with higher i-bounds is also due to the fact that we do not run them until convergence at each iteration.

Experiments on the 200-variable-set All experiments on the 200-variable-set were run on a Pentium-1700 MHz machine with a 256 MB RAM running the version 9.0 of the red-hat Linux operating system. The time-out used was 1800 seconds. We ran all the algorithms with i-bound 2 on 100 instances each when the number of constraints was 840, 850, 860 and 870 respectively. Algorithms with higher i-bound were found to be infeasible because of the higher cost both in terms of time and space that is required to process each node. We observed that instances with 840 and 850 constraints lie in the under-constrained region of the phase transition while the instances with 860 and 870 constraints lie in the over-constrained region of the phase transition. We analyze the results on only soluble CSPs for the 200-variable-set because as mentioned earlier, our look-ahead schemes are inherently designed for soluble CSPs. The results obtained here are similar to the 100-variable-set and are summarized in Table 2.

		IJGP(i)-SC			MBTE(i)-SC			MBTE(i)-MC				
C	Quartiles	i=2	i=3	i=4	i=2	i=3	i=4	i=2	i=3	i=4	SLS	MAC
		Time for soluble instances										
	1st quartile	1.2	1.8	3.5	3.7	4.6	8.7	2.3	3.0	5.2	0.1	0.1
420.0(163)	median	1.2	1.9	3.9	4.1	5.6	9.9	4.7	4.5	5.9	0.2	0.2
	3rd quartile	1.7	3.2	6.0	53.4	45.8	65.1	14.9	20.3	22.8	0.3	0.3
	1st quartile	1.2	1.9	3.6	3.9	5.0	9.0	5.3	3.2	5.0	0.1	0.1
430.0(109)	median	1.3	2.1	4.2	14.4	15.8	16.1	13.0	7.1	15.8	0.2	0.2
	3rd quartile	5.1	4.8	7.0	87.7	113.7	80.2	41.4	29.6	48.7	0.5	0.4
	1st quartile	1.3	2.0	3.6	4.5	10.4	8.7	4.7	4.1	6.9	0.1	0.2
440.0(85)	median	1.4	2.1	7.0	29.0	30.2	20.9	9.9	20.4	20.7	0.3	0.4
	3rd quartile	2.4	20.9	23.1	88.7	68.9	144.6	150.2	69.0	57.1	1.1	0.6
	1st quartile	1.3	2.0	4.6	4.2	6.5	10.0	53.4	53.4	84.1	0.0	0.3
450.0(43)	median	1.4	2.2	11.2	9.3	38.9	287.2	134.9	57.8	89.5	0.7	0.6
	3rd quartile	9.5	8.8	32.7	356.4	209.7	436.3	290.7	217.3	218.1	1.8	0.7
		Backtracks for soluble instances										
	1st quartile	0.0	0.0	0.0	100.0	100.0	100.8	1.8	0.0	0.0		38.8
420.0(163)	median	0.0	0.0	0.0	111.0	103.0	115.0	83.0	29.0	10.0		44.0
	3rd quartile	21.0	51.0	42.0	871.0	481.0	455.0	338.0	287.0	126.0		54.0
	1st quartile	0.0	0.0	0.0	100.8	100.0	100.0	55.3	3.3	2.3		32.3
430.0(109)	median	2.0	3.0	2.0	261.0	267.0	155.0	243.0	62.0	92.0		40.0
	3rd quartile	82.0	68.5	60.0	1200.5	1258.0	485.5	838.5	334.5	415.0		58.5
	1st quartile	0.0	0.0	0.8	117.5	171.0	102.3	58.8	15.3	31.5		42.0
440.0(85)	median	2.0	39.0	132.0	485.5	331.0	334.5	415.0	48.5	140.0		73.0
	3rd quartile	69.0	75.8	228.0	1334.0	849.0	565.0	1085.0	386.0	362.3		109.0
	1st quartile	0.0	0.0	17.0	102.0	109.0	116.0	687.0	484.0	503.0		39.0
450.0(43)	median	0.0	0.0	69.0	178.0	400.0	1550.0	1673.0	565.0	531.0		58.0
	3rd quartile	162.8	89.0	291.5	4213.8	2033.8	2278.5	5337.8	1823.3	1379.5		75.8
		Time for insoluble instances										
	1st quartile	46.0	70.6	107.7	237.5	183.3	290.3	274.1	192.9	202.1		0.3
420.0(37)	median	76.6	139.9	168.7	392.6	327.9	398.2	398.4	223.2	276.4		0.4
	3rd quartile	99.1	143.5	261.2	475.2	486.2	500.0	500.0	331.0	363.2		0.5
	1st quartile	24.0	32.8	86.9	158.3	167.1	242.5	104.9	88.7	116.3		0.6
430.0(91)	median	44.0	62.7	107.9	276.6	231.7	306.7	148.0	134.1	171.1		0.7
	3rd quartile	59.6	71.5	154.2	415.5	430.2	500.0	246.8	187.6	298.6		0.8
	1st quartile	19.4	30.2	56.2	106.4	108.1	154.0	97.8	92.7	105.9		0.1
440.0(115)	median	26.2	43.4	80.1	211.5	232.6	311.8	164.1	141.4	161.3		0.4
	3rd quartile	42.0	62.0	143.6	371.8	382.0	500.0	223.5	174.7	263.0		0.7
	1st quartile	15.9	23.2	46.9	144.4	155.4	216.2	80.5	79.4	102.3		0.2
450.0(157)	median	30.2	38.2	90.4	230.6	231.6	327.3	132.0	124.4	156.5		0.4
	3rd quartile	43.5	68.2	113.0	438.0	428.4	495.5	299.5	256.6	292.4		0.6
		Backtracks for insoluble instances										
	1st quartile	886.0	1000.5	895.0	1814.0	1402.5	1194.3	3647.3	2266.0	1418.3		57.8
420.0(37)	median	1501.0	1718.5	1415.0	4870.0	2870.0	2211.0	4463.5	2577.0	1764.0		92.5
	3rd quartile	2152.0	2067.0	2064.0	7253.0	5312.0	3054.0	34533.0	3044.0	2598.0		106.0
	1st quartile	606.5	422.0	630.8	1565.8	1358.5	1242.5	1537.8	1001.8	856.8		53.3
430.0(91)	median	944.0	762.5	792.5	2818.5	1829.5	1415.0	2038.0	1599.0	1350.0		61.0
	3rd quartile	1394.0	1083.0	1398.5	4664.5	3434.5	2512.5	3782.0	2116.5	2215.5		68.5
	1st quartile	395.0	354.0	358.0	991.0	732.0	652.0	1149.0	920.0	704.0		34.0
440.0(115)	median	565.0	549.0	617.0	2332.0	1861.0	1549.0	2068.0	1460.0	1048.0		67.0
	3rd quartile	910.8	793.3	881.0	3795.8	2894.5	2181.3	2790.3	1828.3	1695.5		89.5
	1st quartile	293.0	245.5	257.0	1210.5	953.5	760.5	920.0	711.0	550.0		27.5
450.0(157)	median	573.0	420.0	481.0	2090.0	1622.0	1429.0	1756.0	1131.0	918.0		52.0
	3rd quartile	1003.3	845.5	706.8	3698.5	2863.8	1868.5	3483.8	2578.3	1791.5		79.5

Table 1: Table showing time in seconds and number of backtracks taken by various algorithms for *100 variable problems with K=4, T=4.* C:number of constraints and i is the i-bound used. The quantity in the bracket alongside each constraint indicates the number of instances on which the results are based on.

		IJGP(2)-SC		MBTE(2)-SC		MBTE(2)-MC		MAC		SLS
C	Quartiles	T	B	T	B	T	B	T	B	T
	1st quartile	17.0	0.0	58.6	6.5	73.7	201.5	0.5	102.5	2.7
840.0(72)	median	18.2	3.0	214.6	643.0	272.6	489.0	1.3	303.0	6.9
	3rd quartile	159.0	898.0	1800.0	3334.0	1800.0	2811.0	2.9	729.0	13.3
	1st quartile	18.1	1.5	356.9	266.5	531.9	1473.0	0.3	135.0	2.1
850.0(59)	median	24.9	48.0	1800.0	3097.0	1800.0	2301.0	0.8	265.0	12.5
	3rd quartile	182.2	1589.0	1800.0	4542.0	1800.0	2814.0	3.3	1044.8	53.5
	1st quartile	27.6	32.5	1024.9	1416.8	689.4	1495.5	0.6	226.5	0.7
860.0(37)	median	135.7	656.5	1800.0	2844.5	1800.0	2068.5	0.9	312.0	7.3
	3rd quartile	973.4	4373.0	1800.0	3604.0	1800.0	2543.0	1.9	508.8	22.1
	1st quartile	35.7	3.75	1800.0	2466	643.6	1249.25	1.2	443	34.8
870.0(22)	median	83.8	358	1800.0	3451	1800.0	1657.5	2.7	887.5	50.0
	3rd quartile	549.2	2341	1800.0	3730	1800.0	2077	3.7	1924	114.9

Table 2: Table showing the time in seconds and the number of backtracks made by various algorithms for *200 variable soluble problems with K=4, T=4.* C:number of constraints, B: the number of backtracks, T: time in seconds and i is the i-bound used. The quantity in the bracket alongside each constraint indicates the number of instances on which the results are based on.

C	Quartiles	IJGP(2)-SC T	IJGP(2)-SC B	MAC T	MAC B	SLS T
2040.0(41)	1st quartile	126.2	0.0	23.1	7844.0	31.7
	median	276.7	404.0	45.1	14987.0	48.4
	3rd quartile	638.3	1280.5	187.4	65446.5	94.2
2060.0(30)	1st quartile	128.8	2.0	72.9	22708.3	44.8
	median	180.9	109.0	124.9	38507.5	184.5
	3rd quartile	1800.0	11870.0	480.5	129258.0	1195.9
2080.0(24)	1st quartile	128.3	2.8	171.2	56792.3	82.7
	median	326.5	479.0	497.9	131492.0	261.5
	3rd quartile	1800.0	5570.5	498.2	135240.3	376.9
2100.0(18)	1st quartile	1665.4	3735.0	13.3	3939.0	558.6
	median	1800.0	12791.0	113.2	31759.0	988.5
	3rd quartile	1800.0	17237.0	270.7	74156.3	1503.8

Table 3: Table showing the time in seconds and the number of backtracks made by various algorithms for *500 variable soluble problems with K=4, T=4*. C:number of constraints, B: the number of backtracks, T: time in seconds and i is the i-bound used. The quantity in the bracket alongside each constraint indicates the number of instances on which the results are based on.

Experiments on the 500-variable-set All experiments on the 500 variable problems were run on a Pentium-2400 MHz machine with a 1000 MB RAM running the red-hat Linux operating system. The time-out used was 7200s. On the 500-variable-set, we report results on IJGP(2)-SC, SLS and MAC. MBTE(2)-SC and MBTE(2)-MC were able to solve only 3 and 6 problems respectively out of the 400 problems considered in the stipulated time-bound. So we do not report results on these algorithms. We observed that the phase transition for the 500-variable-set occurs around when the number of constraints is in the range 2040-2100.

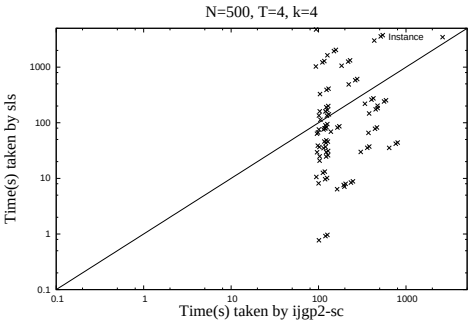

Figure 2: Results comparing IJGP(2)-SC and SLS for soluble 500 variable problems with K=4, T=4.

Figures 2 and 3 show a scatter plot of the time taken by IJGP(2)-SC vs SLS and IJGP(2)-SC vs MAC respectively while Table 3 gives a summary of results for SLS, MAC and IJGP(2)-SC. From Table 3 and Figures 3 and 2, we observe that SLS and MAC are only slightly better in terms of time as compared to IJGP(2)-SC. Once again note that the number of backtracks performed by IJGP(2)-SC is significantly less than MAC.

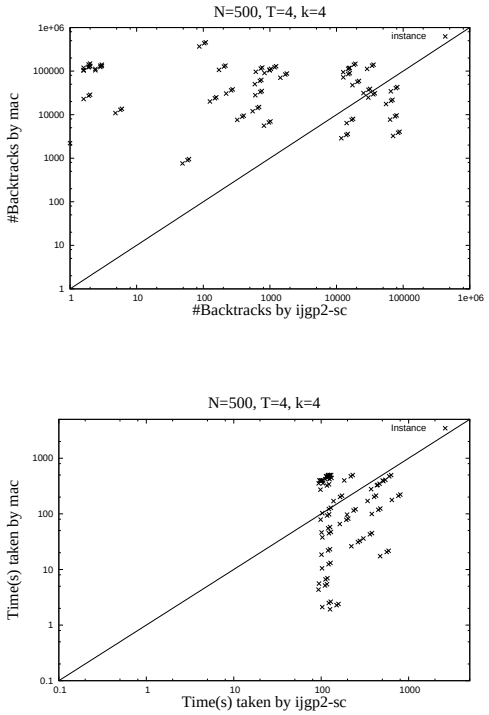

Figure 3: Results comparing IJGP(2)-SC and MAC for *soluble 500 variable problems with K=4, T=4*.

Experiments on the 1000-variable-set On the 1000-variable-set, we report results on IJGP(2)-SC and MAC. Our SLS implementation timed-out on all the problems in the 1000-variable-set. We must acknowledge that we are using a sub-optimal implementation of SLS and better results could be obtained by a better implementation of SLS. All experiments on the 1000-variable set were run on a Pentium-2400 MHz machine with a 1000 MB RAM running the red-hat Linux operating system. The time-out used was 7200s. The number of constraints was varied between 4000 and 4100 which corresponds to the phase-transition region.

Figure 4 shows a scatter plot of the time and the number of backtracks taken by IJGP(2)-SC and MAC. We can see that IJGP(2)-SC is better than MAC both in terms of cpu time and the number of backtracks. This result clearly indicates that IJGP(2)-SC scales better than MAC.

To summarize, we found that in general IJGP(i)-SC shows a consistently better performance than MBTE(i)-SC while MBTE(i)-SC shows a consistently better performance than MBTE(i)-MC both in terms of the cpu time and the number of backtracks. On the other hand, we found that algorithms that use a lower i-bound outperform those that use a higher i-bound in terms of cpu time but not in terms of the number of backtracks. IJGP(2)-SC was the best performing algorithm among

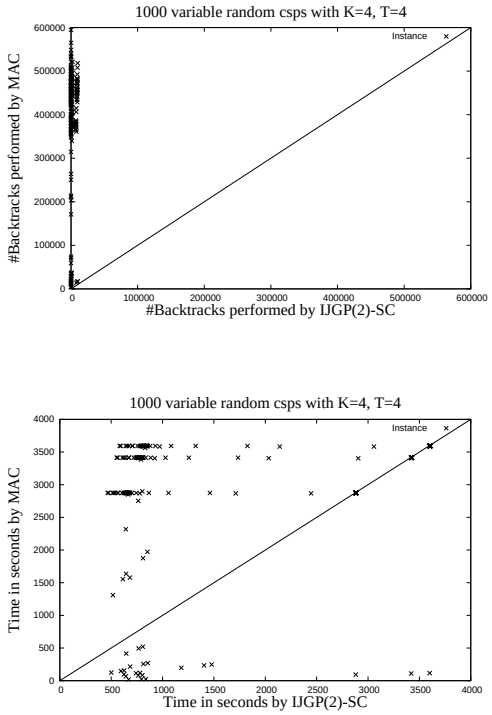

Figure 4: Results comparing IJGP(2)-SC and MAC for *soluble 1000 variable problems with K=4, T=4*.

our look-ahead schemes in terms of cpu time. More importantly, we found that IJGP(2)-SC was better than MAC in terms of the cpu time on larger problems (1000 variables) while it was inferior to SLS and MAC on smaller problems (100, 200 and 500 variables). In other words, our results show that IJGP(2)-SC is more scalable than MAC. Also since the MAC implementation is highly optimized and the implementations of our look-ahead schemes are sub-optimal, we find our results very encouraging.

Summary and Conclusions

As we claim, the solution count measure is stronger than the min-conflicts heuristic when everything else remains fixed since, using the same bound i, they have identical pruning power but SC also provides value ordering. This heuristic is designed to be effective when the problem has a solution. We demonstrated empirically that indeed for MBTE, MBTE-SC is stronger than MBTE-MC. Still, it is possible to implement MBTE-MC as a strictly propagation algorithm with relation description. This is likely to be much more efficient and therefore may present a worthwhile approach and time-accuracy trade-off that is not dominated by MBTE-SC. We plan to investigate this in the future.

The next question is whether IJGP is more cost-effective than MBTE for SC approximation. Our empirical evaluation strongly suggests that IJGP-SC is much stronger and dominates MBTE-SC, both in pruning power, as shown over inconsistent instances (see Table 1, as well as in its informativeness and guidance to the solution, as shown over consistent instances (see Table 1). However, there is no complete dominance.

We see that the "focus" power of IJGP-SC for value ordering is very strong even for i=2 (the median is low even in the phase transition). It is much stronger than MBTE-SC. See in particular the results for 500 and 1000 variables.

Finally, our experiments show that IJGP-SC has better scalability than MAC and SLS (at least relative to our implementation). Comparing IJGP-SC with MAC/SLS, we observed that MAC/SLS are superior on small problems, but as the problem size grows the relative performance of IJGP-SC improves and on 1000 variable problems, IJGP-SC outperforms MAC. This is even more significant since our implementation is far from optimal and we are using the heuristic on top of chronological backtracking without any backjumping or constraint recording.

References

Bessiere, C., and Regin, J.-C. 1996. MAC and combined heuristics: Two reasons to forsake FC (and CBJ?) on hard problems. In *Principles and Practice of Constraint Programming*, 61–75.

Cheeseman, P.; Kanefsky, B.; and Taylor, W. M. 1991. Where the Really Hard Problems Are. In *Proceedings of the Twelfth International Joint Conference on Artificial Intelligence, IJCAI-91, Sidney, Australia*, 331–337.

Christian Bessière, Eugene C. Freuder, J.-C. R. 1999. Using constraint metaknowledge to reduce arc consistency computation. *Artificial Intelligence* 125–148.

Dechter, R., and Mateescu, R. 2003. A simple insight into iterative belief propagation's success. *UAI-2003*.

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

Dechter, R.; Kask, K.; and Larrosa, J. 2001. A general scheme for multiple lower-bound computation in constraint optimization. *Principles and Practice of Constraint Programming (CP-2001)*.

Dechter, R.; Kask, K.; and Mateescu, R. 2002. Iterative join graph propagation. In *UAI '02*, 128–136. Morgan Kaufmann.

Gent, I. P., and Walsh, T. 1993. Towards an understanding of hill-climbing procedures for sat. In *Proceedings of the Eleventh National Conference on Artificial Intelligence (AAAI-93)*, 28–33.

Gottlob, G.; Leone, N.; and Scarello, F. 1999. A comparison of structural csp decomposition methods. *Ijcai-99*.

Kask, K., and Dechter, R. August 1995. Gsat and

local consistency. In *International Joint Conference on Artificial Intelligence (IJCAI-95)*, 616–622.

Lauritzen, S., and Spiegelhalter, D. 1988. Local computation with probabilities on graphical structures and their application to expert systems. *Journal of the Royal Statistical Society, Series B* 50(2):157–224.

MacIntyre, E.; Prosser, P.; Smith, B.; and Walsh, T. 1998. Random constraint satisfaction: Theory meets practice. *Lecture Notes in Computer Science* 1520:325+.

Morris, P. 1993. The breakout method for escaping from local minima. In *Proceedings of the Eleventh National Conference on Artificial Intelligence (AAAI-93)*, 40–45.

Sabin, D., and Freuder, E. C. 1997. Understanding and improving the MAC algorithm. In *Principles and Practice of Constraint Programming*, 167–181.

Shafer, G., and Shenoy, P. 1990. Probability propagation. *Annals of Math and Artificial Intelligence* 2:327–352.

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