Using Mini-Bucket Heuristics for Max-CSP *

Kalev Kask and Rina Dechter

Department of Information and Computer Science
University of California, Irvine, CA 92697-3425
{kkask, dechter}@ics.uci.edu

June 30, 2000

Abstract

This paper evaluates the power of a new scheme that generates search heuristics mechanically. This approach was presented and evaluated first in the context of optimization in belief networks. In this paper we extend this work to Max-CSP. The approach involves extracting heuristics from a parameterized approximation scheme called Mini-Bucket elimination that allows controlled trade-off between computation and accuracy. The heuristics are used to guide Branch-and-Bound and Best-First search, whose performance are compared on a number of constraint problems. Our results demonstrate that both search schemes exploit the heuristics effectively, permitting controlled trade-off between preprocessing (for heuristic generation) and search.

1 Introduction

In this paper we will present a general scheme of mechanically generating search heuristics for solving combinatorial optimization problems, using either Branch and Bound or Best First search. Within this scheme, the trade-off between the quality of the heuristic function and its computational complexity is quantified by an input parameter.

The scheme is based on the Mini-Bucket technique; a class of parameterized approximation algorithms for optimization tasks based on the bucket-elimination framework [2]. The mini-bucket approximation uses a controlling parameter which allows adjustable levels

*This work was supported by NSF grant IIS-9610015 and by Rockwell Micro grant #99-030.

of accuracy and efficiency [5]. It was presented and analyzed for deterministic and probabilistic tasks such as finding the most probable explanation (MPE), belief updating, and finding the maximum a posteriori hypothesis. Encouraging empirical results were reported on a variety of classes of optimization domains, including medical-diagnosis networks and coding problems [7]. However, as evident by the error bound produced by these algorithms, in some cases the approximation is seriously suboptimal even when using the highest feasible accuracy level. In such cases, augmenting the Mini-Bucket approximation with search could be cost-effective.

Recently, we demonstrated how the mini-bucket scheme can be extended and used for mechanically generating heuristics search algorithms that solve optimization tasks, using the task of finding the Most Probable Explanation in a Bayesian network. We showed that the functions produced by the Mini-Bucket method can serve as the basis for creating heuristic evaluation functions for search [9, 8]. These heuristics provide an upper bound on the cost of the best extension of a given partial assignment. Since the Mini-Bucket's accuracy is controlled by a bounding parameter, it allows heuristics having varying degrees of accuracy and results in a spectrum of search algorithms that can trade-off heuristic computation and search.

In this paper we extend this approach to Max-CSP. Max-CSP is an optimization version of Constraint Satisfaction. Instead of finding an assignment that satisfies all constraints, a Max-CSP solution satisfies a maximum number of constraints. We will use the Mini-Bucket approximation to generate a heuristic function that computes a lower bound on the minimum number of constraints that are violated in the best extension of any partial assignment. We evaluate the power of the generated heuristic within both *Branch-and-Bound* and *Best-First* search on a variety of randomly generated constraint problems.

Branch-and-Bound searches the space of partial assignments in a depth-first manner. It will expand a partial assignment only if its lower-bounding heuristic function is smaller than the current best upper bound solution. The virtue of Branch-and-Bound is that it requires a limited amount of memory and can be used as an anytime scheme; whenever interrupted, Branch-and-Bound outputs the best solution found so far. Best-First explores the search space in uniform frontiers of partial instantiations, each having the same value for the evaluation functions, while progressing in waves of increasing values. Since, as shown, the generated heuristics are admissible and monotonic, their use within Best-First search yields A* type algorithms whose properties are well understood. When the algorithm finds a solution, it is guaranteed to be optimal. When provided with more accurate heuristics, it explores a smaller search space, but otherwise it requires substantial memory. It is also known that Best-First algorithms are optimal performance wise. Namely, when given the same heuristic information, Best-First search is the most efficient algorithm in terms of the size of the search space it explores [4]. In particular, Branch-and-Bound will expand any node that is expanded by Best-First (up to some tie breaking conditions), and in many cases it explores a larger space. Still, Best-First may occasionally fail because of its memory requirements. Hybrid approaches similar to those presented for A* in the search community in the past decade are clearly of potential here as well [11].

In this paper we extend our studies of Mini-Bucket search heuristics to the Max-CSP class. Specifically, we evaluate a Best-First algorithm with Mini-Bucket heuristics (BFMB) and a Branch-and-Bound algorithm with Mini-Bucket heuristics (BBMB), and compared empirically against the full bucket elimination and its Mini-Bucket approximation over randomly generated constraint satisfaction problems for solving the Max-CSP problem. For comparison we also ran a number of state of the art algorithms such as PFC-MPRDAC [12] and a variant of Stochastic Local Search.

We show that both BBMB and BFMB exploit heuristics' strength in a similar manner: on all problem classes, the optimal trade-off point between heuristic generation and search lies in an intermediate range of the heuristics' strength. As problems become larger and harder, this optimal point gradually increases towards the more computationally demanding heuristics. We show that BBMB/BFMB outperform both SLS and PFC-MRDAC on some of the problems, while on others SLS and PFC-MRDAC are better. Unlike our results in [8, 9] here Branch-and-Bound clearly dominates Best-First search.

Section 2 provides preliminaries and background on the Mini-Bucket algorithms. Section 3 describes the main idea of the heuristic function which is built on top of the Mini-Bucket algorithm, proves its properties, and embeds the heuristic within Best-First and Branch-and-Bound search. Sections 4 and 5 present empirical evaluations, while Section 6 provides conclusions.

1.1 Related work

Our approach applies the paradigm that heuristics can be generated by consulting relaxed models, suggested in [14]. It can be viewed as a generalization of Branch-and-Bound algorithms for integer programming that are restricted to linear objective functions and constraints, and use relaxation to linear programming, assuming the integer restrictions on the domains are removed [?, 19]. The Mini-Bucket heuristics can also be viewed as an extension of bounded constraint propagation algorithms that were investigated in the constraint community in the last decade [1]. However, rather than applying this idea to the constraints only, we extend it to the objective function as well.

2 Background

2.1 Notation and Definitions

Constraint Satisfaction is a framework for formulating real-world problems as a set of constraints between variables. They are graphically represented by nodes corresponding to variables and edges corresponding to constraints between variables.

DEFINITION **2.1 (Graph Concepts)** *An* undirected graph *is a pair,* $G = \{V, E\}$, *where* $V = \{X_1, ..., X_n\}$ *is a set of variables, and* $E = \{(X_i, X_j) | X_i, X_j \in V\}$ *is the set of edges. The degree of a variable is the number of edges incident to it.*

DEFINITION **2.2 (Constraint Satisfaction Problem (CSP))** *A* Constraint Satisfaction Problem (CSP) *is defined by 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* $R_i \subseteq D_{i1} \times ... \times D_{ik}$ *defined on a subset of variables* $S_i = \{X_{i1}, ..., X_{ik}\}$ *called the scope of* C_i, *consisting of all tuples of values for* $\{X_{i1}, ..., X_{ik}\}$ *which are compatible with each other. For the max-CSP problem, we express the relation 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. A constraint network can be represented by a constraint graph that contains 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. A problem with a solution is termed* satisfiable *or* consistent. *A binary CSP is a one where each constraint involves at most two variables.*

Many real-world problems are often over-constrained are don't have a solution. In such cases, it is often desirable to find an assignment that satisfies a maximum number of constraints.

DEFINITION **2.3 (Max-CSP)** *Given a CSP, the Max-CSP task is to find an assignment that satisfies the most constraints.*

Although a Max-CSP problem is defined as a maximization problem, it can be implemented as a minimization problem. Instead of maximizing the number of constraints that are satisfied, we minimize the number of constraints that are violated.

DEFINITION **2.4 (Induced-width)** *An* ordered graph *is a pair* (G, d) *where* G *is an undirected graph, and* $d = X_1, ..., X_n$ *is an ordering of the nodes. The* width of a node *in an ordered graph is the number of its earlier neighbors. The* width of an ordering d, $w(d)$, *is the maximum width over all nodes. The* induced width of an ordered graph, $w^*(d)$, *is the width of the induced ordered graph obtained by processing the nodes recursively, from last to first; when node* X *is processed, all its earlier neighbors are connected.*

DEFINITION **2.5** *Given a function* h *defined over a subset of variables* S, *where* $X \in S$, *functions* $(\min_X h)$, $(\max_X h)$, *and* $(\sum_X h)$ *are defined over* $U = S - \{X\}$ *as follows: For every* $U = u$, *and denoting by* (u, x) *the extension of tuple* u *by assignment* $X = x$,

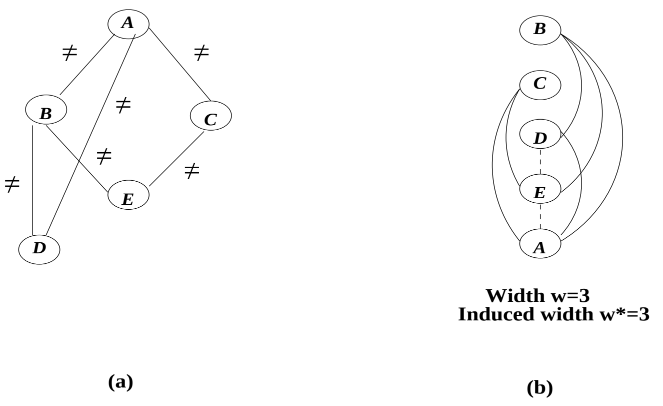


Figure 1: a) Constraint graph of a graph coloring problem, b) an ordered graph along $d = (A, E, D, C, B)$.

$(\min_X h)(u) = \min_x h(u,x)$, $(\max_X h)(u) = \max_x h(u,x)$, *and* $(\sum_X h)(u) = \sum_x h(u,x)$. *Given a set of functions* $h_1, ..., h_j$ *defined over the subsets* $S_1, ..., S_j$, *the product function* $(\Pi_j h_j)$ *and* $\sum_J h_j$ *are defined over* $U = \cup_j S_j$. *For every* $U = u$, $(\Pi_j h_j)(u) = \Pi_j h_j(u_{S_j})$, *and* $(\sum_j h_j)(u) = \sum_j h_j(u_{S_j})$.

Example 2.1 *A graph coloring problem is a typical example of a CSP problem. It is defined as a set of nodes and arcs between the nodes. The task is to assign a color to each node such that adjacent nodes have different colors. An example of a constraint graph of a graph coloring problem containing variables A, B, C, D, and E, with each variable having 2 values (colors) is in Figure 1. The fact that adjacent variables must have different colors is represented by an inequality constraint. The problem in Figure 1 is inconsistent. When formulated as a Max-CSP problem, its solution satisfies all but one constraint. Given the ordering* $d = A, E, D, C, B$ *of the graph, the width and induced-width of the ordered graph is 3.*

2.2 Bucket and Mini-Bucket Elimination Algorithms

Bucket elimination is a unifying algorithmic framework for dynamic-programming algorithms applicable to probabilistic and deterministic reasoning [3]. In the following we will present its adaptation to solving the Max-CSP problem.

The input to a bucket-elimination algorithm consists of a collection of functions or relations (e.g., clauses for propositional satisfiability, constraints, or conditional probability matrices for belief networks). Given a variable ordering, the algorithm partitions the functions into buckets, each associated with a single variable. A function is placed in the bucket of its argument that appears latest in the ordering. The algorithm has two phases.

Algorithm Elim-Max-CSP
Input: A constraint network $P(X, D, C)$; an ordering of the variables d. Each constraint is represented as a function $C_i(X_{i1} = x_{i1}, ..., X_{ik} = x_{ik}) = 0$ if $(x_{i1}, ..., x_{ik}) \in R_i$, and 1 otherwise.
Output: An assignment satisfying the most constraints.
1. **Initialize:** Partition P into $bucket_1, \ldots, bucket_n$, where $bucket_i$ contains all constraints whose highest variable is X_i. Let $S_1, ..., S_j$ be the scopes of functions (new or old) in the processed bucket.
2. **Backward:** For $p \leftarrow n$ down-to 1, do
for $h_1, h_2, ..., h_j$ in $bucket_p$, do
• **If** $bucket_p$ contains an instantiation $X_p = x_p$, assign $X_p = x_p$ to each h_i and put each resulting function into its appropriate bucket.
• **Else,** generate the function h^p: $h^p = min_{X_p} \sum_{i=1}^{j} h_i$. Add h^p to the bucket of the largest-index variable in $U_p \leftarrow \bigcup_{i=1}^{j} S_i - \{X_p\}$.
3. **Forward:** Assign values in the ordering d s.t. the sum of cost functions in each bucket is minimized.

Figure 2: Algorithm *Elim-Max-CSP*

During the first, top-down phase, it processes each bucket, from the last variable to the first. Each bucket is processed by a variable elimination procedure that computes a new function which is placed in a lower bucket. For Max-CSP, this procedure computes the sum of all constraint matrices and minimizes over the bucket's variable. During the second, bottom-up phase, the algorithm constructs a solution by assigning a value to each variable along the ordering, consulting the functions created during the top-down phase. Figure 2 shows *Elim-Max-CSP*, the bucket-elimination algorithm for computing Max-CSP. It can be shown that

THEOREM **2.2** *[2] The time and space complexity of Elim-Max-CSP applied along order d, are exponential in the induced width $w^*(d)$ of the network's ordered moral graph along the ordering d.* □

The main drawback of bucket elimination algorithms is that they require too much time and, especially, too much space for storing intermediate functions. *Mini-Bucket elimination* is an approximation scheme designed to avoid this space and time complexity of full bucket elimination [5] by partitioning large buckets into smaller subsets called mini-buckets which are processed independently. Here is the rationale. Let $h_1, ..., h_j$ be the functions in $bucket_p$. When *Elim-Max-CSP* processes $bucket_p$, it computes the function h^p: $h^p = min_{X_p} \sum_{i=1}^{j} h_i$. Instead, the Mini-Bucket algorithm creates a partitioning $Q' = \{Q_1, ..., Q_r\}$ where the mini-bucket Q_l contains the functions $h_{l_1}, ..., h_{l_k}$ and it processes each mini-bucket (by taking the sum and minimizing) separately. It therefore computes $g^p = \sum_{l=1}^{r} min_{X_p} \sum_{l_i} h_{l_i}$. Clearly, $h^p \geq g^p$. Therefore, the lower bound g^p computed in each bucket yields an overall lower bound on the number of constraints violated by the output assignment.

The quality of the lower bound depends on the degree of the partitioning into mini-buckets. Given a bounding parameter i, the algorithm creates an i-partitioning, where each mini-bucket includes no more than i variables. Algorithm *MB-Max-CSP(i)*, described in Figure 3, is parameterized by this i-bound. The algorithm outputs not only a lower bound on the Max-CSP value (namely, on the minimum number of violated constraints) and an assignment whose number of violated constraints is an upper bound, but also the collection of augmented buckets. By comparing the lower bound to the upper bound, we can always have a bound on the error for the given instance.

The algorithm's complexity is time and space $O(exp(i))$ where $i \leq n$. When the bound, i, is large enough (i.e. when $i \geq w^*$), the Mini-Bucket algorithm coincides with the full bucket elimination. In summary,

THEOREM **2.3** *Algorithm MB-Max-CSP(i) generates a lower bound on the exact Max-CSP value, and its time and space complexity is exponential in its bound i.* $\Box$

Example 2.4 *Figure 4 illustrates how algorithms* Elim-Max-CSP *and* MB-Max-CSP(i) *for $i = 3$ process the network in Figure 1a along the ordering (A, E, D, C, B). Algorithm* Elim-Max-CSP *records new functions $h^B(a,d,e)$, $h^C(a,e)$, $h^D(a,e)$, and $h^E(a)$. Then, in the bucket of A, $min_a h^E(a)$ equals the minimum number of constraints that are violated. Subsequently, an assignment is computed for each variable from A to B by selecting a value that minimizes the sum of functions in the corresponding bucket, conditioned on the previously assigned values. On the other hand, the approximation* MB-Max-CSP(3) *splits bucket B into two mini-buckets, each containing no more than 3 variables, and generates $h^B(e)$ and $h^B(d,a)$. A lower bound on the Max-CSP value is computed by $L = min_a(h^E(a) + h^D(a))$. Then, a suboptimal tuple is computed similarly to the Max-CSP tuple by assigning a value to each variable that minimizes the sum of functions in the corresponding bucket.*

3 Heuristic Search with Mini-Bucket Heuristics

3.1 The Heuristic Function

In the following, we will assume that a Mini-Bucket algorithm was applied to a constraint network using a given variable ordering $d = X_1, ..., X_n$, and that the algorithm outputs an ordered set of augmented buckets $bucket_1$,...,$bucket_p$,...,$bucket_n$, containing both the input constraints and the newly generated functions. Relative to such an ordered set of augmented buckets, we use the following convention:

Algorithm MB-Max-CSP(i)

Input: A constraint network $P(X, D, C)$; an ordering of the variables d. Each constraint is represented as a function $C_i(X_{i1} = x_{i1}, ..., X_{ik} = x_{ik}) = 0$ if $(x_{i1}, ..., x_{ik}) \in R_i$, and 1 otherwise.

Output: An upper bound on the Max-CSP, an assignment, and the set of ordered augmented buckets.

1. **Initialize:** Partition constraints into buckets. Let $S_1, ..., S_j$ be the scopes of constraints in $bucket_p$.

2. **Backward** For $p \leftarrow n$ down-to 1, do

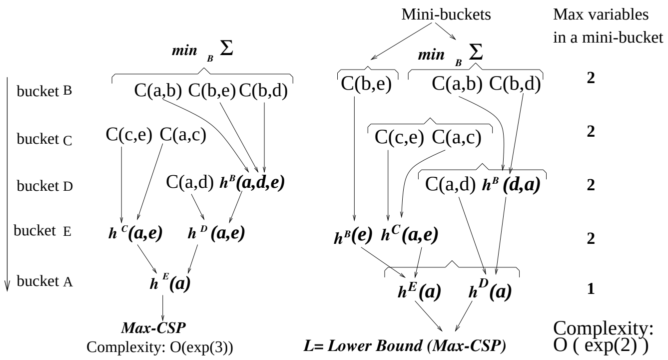
• **If** $bucket_p$ contains an instantiation $X_p = x_p$, assign $X_p = x_p$ to each h_i and put each in appropriate bucket.

• **Else,** for $h_1, h_2, ..., h_j$ in $bucket_p$, generate an (i)-partitioning, $Q' = \{Q_1, ..., Q_r\}$. For each $Q_l \in Q'$ containing $h_{l_1}, ...h_{l_t}$ generate function h^l, $h^l = min_{X_p} \sum_{i=1}^{t} h_{l_i}$. Add h^l to the bucket of the largest-index variable in $U_l \leftarrow \bigcup_{i=1}^{j} S(h_{l_i}) - \{X_p\}$.

3. **Forward** For $i = 1$ to n do, given $x_1, ..., x_{p-1}$ choose a value x_p of X_p that minimizes the sum of all the cost functions in X_p's bucket.

4. Output the ordered set of augmented buckets, an upper bound and a lower bound assignment.

Figure 3: Algorithm *MB-Max-CSP(i)*



(a) A trace of *Elim-Max-CSP* (b) A trace of *MB-Max-CSP(2)*

Figure 4: Execution of *Elim-Max-CSP* and *MB-Max-CSP(i)*

- C_{p_j} denotes an input constraint matrix in $bucket_p$ (namely, one whose highest-ordered variable is X_p), enumerated by j.
- h_{p_j} denotes a function residing in $bucket_p$ that was generated by the Mini-Bucket algorithm, enumerated by j.
- h_j^p stands for a function created by processing the j-th mini-bucket in $bucket_p$.
- λ_{p_j} stands for an arbitrary function in $bucket_p$, enumerated by j. Notice that $\{\lambda_{p_j}\} = \{C_{p_j}\} \cup \{h_{p_j}\}$.

We denote by $buckets(1..p)$ the union of all functions in the bucket of X_1 through the bucket of X_p. $S(f)$ denotes the scope of function f.

We will now show that the functions recorded by the Mini-Bucket algorithm can be used to lower bound the number of constraints violated by the best extension of any partial assignment, and therefore can serve as heuristic evaluation functions in a *Best-First* or *Branch-and-Bound* search.

DEFINITION **3.1 (Exact Evaluation Function)** *Given a variable ordering $d = X_1, ..., X_n$, let $\bar{x}^p = (x_1, ..., x_p)$ be an assignment to the first p variables in d. The number of constraints violated by the best extension of $\bar{x}^p$, denoted $f^*(\bar{x}^p)$ is defined by*

$$f^*(\bar{x}^p) = min_{x_{p+1},...,x_n} \sum_{k=1}^{n} C_k$$

The above sum defining f^* can be divided into two sums expressed by the functions in the ordered augmented buckets. In the first sum all the arguments are instantiated (belong to buckets $1, ..., p$), and therefore the minimization operation is applied to the second product only. Denoting

$$g(\bar{x}^p) = \left(\sum_{C_i \in \; buckets(1..p)} C_i \right) (\bar{x}^p)$$

and

$$h^*(\bar{x}^p) = min_{(x_{p+1},...,x_n)} \left(\sum_{C_i \in buckets(p+1..n)} C_i \right) (\bar{x}^p, x_{p+1}, ..., x_n)$$

we get

$$f^*(\bar{x}^p) = g(\bar{x}^p) + h^*(\bar{x}^p).$$

During search, the g function can be evaluated over the partial assignment $\bar{x}^p$, while h^* can be estimated by a heuristic function h, derived from the functions recorded by the Mini-Bucket algorithm, as defined next:

DEFINITION **3.2** *Given an ordered set of augmented buckets generated by the Mini-Bucket algorithm, the heuristic function $h(\bar{x}^p)$ is defined as the sum of all the h_j^k functions that satisfy the following two properties: 1) They are generated in buckets $p+1$ through n, and 2) They reside in buckets 1 through p. Namely, $h(\bar{x}^p) = \sum_{i=1}^{p} \sum_{h_j^k \in bucket_i} h_j^k$, where $k > p$, (i.e. h_j^k is generated by a bucket processed before $bucket_p$.)*

The following proposition shows how $g(\bar{x}^{p+1})$ and $h(\bar{x}^{p+1})$ can be updated recursively.

Proposition 1 *Given a partial assignment $\bar{x}^p = (x_1, \ldots x_p)$, both $g(\bar{x}^p)$ and $h(\bar{x}^p)$ can be computed recursively by*

$$g(\bar{x}^p) = g(\bar{x}^{p-1}) + \sum_j C_{p_j}(\bar{x}^p) \quad (1)$$

$$h(\bar{x}^p) = h(\bar{x}^{p-1}) + (\sum_k h_{p_k}(\bar{x}^p) - \sum_j h_j^p(\bar{x}^p)) \quad (2)$$

Proof. A straightforward derivation from the definition. □

THEOREM **3.1 (Mini-Bucket Heuristic)** *For every partial assignment $\bar{x}^p = (x_1, ..., x_p)$, of the first p variables, the evaluation function $f(\bar{x}^p) = g(\bar{x}^p) + h(\bar{x}^p)$ is: 1) Admissible - it never overestimates the number of constraints violated by the best extension of $\bar{x}^p$, and 2) Monotonic - namely $f(\bar{x}^{p+1})/f(\bar{x}^p) \geq 1$.*

Notice that monotonicity means better accuracy at deeper nodes in the search tree.

Proof. To prove monotonicity we will use the recursive equations (1) and (2) in Proposition 1. For any $\bar{x}^p$ and any value v in the domain of X_{p+1}, we have

$$\frac{f(\bar{x}^p, v)}{f(\bar{x}^p)} = \frac{g(\bar{x}^p, v) + h(\bar{x}^p, v)}{g(\bar{x}^p) + h(\bar{x}^p)}$$

$$= \frac{(g(\bar{x}^p) + \sum_j C_{(p+1)_j}(\bar{x}^p, v)) + (h(\bar{x}^p) + (\sum_k h_{(p+1)_k}(\bar{x}^p, v) - \sum_j h_j^{p+1}(\bar{x}^p)))}{g(\bar{x}^p) + h(\bar{x}^p)}$$

$$= 1 + \frac{\sum_j C_{(p+1)_j}(\bar{x}^p, v) + (\sum_k h_{(p+1)_k}(\bar{x}^p, v) - \sum_j h_j^{p+1}(\bar{x}^p))}{g(\bar{x}^p) + h(\bar{x}^p)}$$

$$= 1 + \frac{\sum_i \lambda_{(p+1)_i}(\bar{x}^p, v) - \sum_j h_j^{p+1}(\bar{x}^p)}{g(\bar{x}^p) + h(\bar{x}^p)}$$

Since $h_j^{p+1}(\bar{x}^p)$ is computed for the j-th mini-bucket in bucket $(p+1)$ by minimizing over variable X_{p+1}, (eliminating variable X_{p+1}), we get

$$\sum_i \lambda_{(p+1)_i}(\bar{x}^p, v) \geq \sum_j h_j^{p+1}(\bar{x}^p)$$

Thus, $f(\bar{x}^p, v) \geq f(\bar{x}^p)$, concluding the proof of monotonicity.

The proof of admissibility follows from monotonicity. It is well known that if a heuristic function is monotone and if it is exact for a full solution (which is our case), then it is also admissible [14]. □

In the extreme case when each bucket p contains exactly one mini-bucket, the heuristic function h equals h^*, and the full evaluation function f computes the exact number of constraints violated by the best extension of the current partial assignment.

3.2 Search with Mini-Bucket Heuristics

The tightness of the lower bound generated by the Mini-Bucket approximation depends on its i-bound. Larger values of i generally yield better lower-bounds, but require more computation. Since the Mini-Bucket algorithm is parameterized by i, we get an entire class of Branch-and-Bound search and Best-First search algorithms that are parameterized by i and which allow a controllable trade-off between preprocessing and search, or between heuristic strength and its overhead. Figures 5 and 6 present algorithms BBMB(i) and BFMB(i).

Both algorithms (BBMB(i) and BFMB(i)) are initialized by running the Mini-Bucket algorithm that produces a set of ordered augmented buckets. Branch-and-Bound with Mini-Bucket heuristics (BBMB(i)) traverses the search space in a depth-first manner, instantiating variables from first to last, along ordering d. Throughout the search, the algorithm maintains an upper bound on the value of the Max-CSP assignment, which corresponds to the number of constraints violated by the best full variable instantiation found thus far. When the algorithm processes variable X_p, all the variables preceding X_p in the ordering are already instantiated, so it can compute $f(\bar{x}^{p-1}, X_p = v) = g(\bar{x}^{p-1}, v) + h(\bar{x}^p, v)$ for each extension $X_p = v$. The algorithm prunes all values v whose heuristic estimate (lower bound) $f(\bar{x}^p, X_p = v)$ is greater than or equal to the current best upper bound, because such a partial assignment $(x_1, \ldots x_{p-1}, v)$ cannot be extended to an improved full assignment. The algorithm assigns the best value v to variable X_p and proceeds to variable X_{p+1}, and when variable X_p has no values left, it backtracks to variable X_{p-1}. Search terminates when it reaches a time-bound or when the first variable has no values left. In the latter case, the algorithm has found an optimal solution.

Algorithm Best-First with Mini-Bucket heuristics (BFMB(i)) starts by adding a dummy node x_0 to the list of open nodes. Each node corresponds to a partial assignment $\bar{x}^p$ and has an associated heuristic value $f(\bar{x}^p)$. Initially $f(x_0) = 0$. The basic step of the algorithm consists of selecting an assignment $\bar{x}^p$ from the list of open nodes having the smallest heuristic value $f(\bar{x}^p)$, expanding it by computing all partial assignments $(\bar{x}^p, v)$ for all values v of X_{p+1}, and adding them to the list of open nodes.

Since, as shown, the generated heuristics are admissible and monotonic, their use within Best-First search yields A* type algorithms whose properties are well understood. The

Algorithm BBMB(i)
Input: A constraint network $P(X, D, C)$; ordering d; time bound t.
Each constraint is represented as a function $C_i(X_{i1} = x_{i1}, ..., X_{ik} = x_{ik}) = 0$ if $(x_{i1}, ..., x_{ik}) \in R_i$, and 1 otherwise.
Output: A Max-CSP assignment, or an upper bound and a lower bound on the Max-CSP.
1. **Initialize:** Run MB(i) algorithm which generates a set of ordered augmented buckets and a lower bound on Max-CSP. Set upper bound U to ∞. Set p to 0.
2. **Search:** Execute the following procedure until variable X_1 has no legal values left or until out of time, in which case output the current best solution.
• **Expand:** Given a partial instantiation $\bar{x}^p$, compute all partial assignments $\bar{x}^{p+1} = (\bar{x}^p, v)$ for each value v of X_{p+1}. For each node $\bar{x}^{p+1}$ compute its heuristic value $f(\bar{x}^{p+1}) = g(\bar{x}^{p+1}) + h(\bar{x}^{p+1})$ using
$g(\bar{x}^{p+1}) = g(\bar{x}^p) + \sum_j C_{(p+1)_j}$ and
$h(\bar{x}^{p+1}) = h(\bar{x}^p) + (\sum_k h_{(p+1)_k} - \sum_j h_j^{(p+1)})$.
Discard those assignments $\bar{x}^{p+1}$ for which $f(\bar{x}^{p+1})$ is not smaller than the upper bound U. Add remaining assignments to the search tree as children of $\bar{x}^p$.
• **Forward:** If X_{p+1} has no legal values left, goto Backtrack. Otherwise let $\bar{x}^{p+1} = (\bar{x}^p, v)$ be the best extension to $\bar{x}^p$ according to f. If $p + 1 = n$, then set $L = f(\bar{x}^n)$ and goto Backtrack. Otherwise remove v from the list of legal values. Set $p = p + 1$ and goto Expand.
• **Backtrack:** If p = 1, Exit. Otherwise set $p = p - 1$ and repeat the Forward step.

Figure 5: Algorithm *Branch-and-Bound with MB(i)*

Algorithm BFMB(i)
Input: A constraint network $P(X, D, C)$; ordering d; time bound t.
Each constraint is represented as a function $C_i(X_{i1} = x_{i1}, ..., X_{ik} = x_{ik}) = 0$ if $(x_{i1}, ..., x_{ik}) \in R_i$, and 1 otherwise.
Output: A Max-CSP assignment or just a lower bound and an upper bound (produced by Mini-Bucket).
1. **Initialize:** Run MB(i) algorithm which generates a set of augmented buckets, a lower bound and an upper bound assignment. Insert a dummy node $\bar{x}_0$ in the set L of open nodes. Set $f(\bar{x}_0)$ to 0.
2. **Search:**
• If out of time, output Mini-Bucket assignment.
• Select and remove a node $\bar{x}^p$ with the smallest heuristic value $f(\bar{x}^p)$ from the set of open nodes L.
• If $p = n$ then $\bar{x}^p$ is an optimal solution. Exit.
• Expand $\bar{x}^p$ by computing all child nodes $(\bar{x}^p, v)$ for each value v in the domain of X_{p+1}. For each node $\bar{x}^{p+1}$ compute its heuristic value $f(\bar{x}^{p+1}) = g(\bar{x}^{p+1}) + h(\bar{x}^{p+1})$, where
$g(\bar{x}^{p+1}) = g(\bar{x}^p) + \sum_j C_{(p+1)_j}$ and
$h(\bar{x}^{p+1}) = h(\bar{x}^p) + (\sum_k h_{(p+1)_k} - \sum_j h_j^{(p+1)})$.

• Add all nodes $(\bar{x}^p, v)$ to L and goto Search.

Figure 6: Algorithm *Best-First search with MB(i)*

algorithm is guaranteed to terminate with an optimal solution. When provided with more powerful heuristics it explores a smaller search space, but otherwise it requires substantial space. It is known that Best-First algorithms are optimal. Namely, when given the same heuristic information, Best-First search is the most efficient algorithm in terms of the size of the search space it explores [4]. In particular, Branch-and-Bound will expand any node that is expanded by Best-First (up to tie breaking conditions), and in many cases it explores a larger space. Still, Best-First may occasionally fail because of its memory requirements, because it has to maintain a large subset of open nodes during search, and because of tie breaking rules at the last frontier of nodes having evaluation function value that equals the optimal solution. As we will indeed observe in our experiments, Branch-and-Bound and Best-First search have complementary properties, and both can be strengthen by the Mini-Bucket heuristics.

4 Experimental Methodology

We tested the performance of BBMB(i) and BFMB(i) on set of random CSPs. Each problem in this class is characterized by five parameters: $< A, N, K, C, T >$, where A is the arity of the constraint, N is the number of variables, K is the domain size, C is the number of constraints, and T is the tightness of each constraint, defined as the number of tuples not allowed. In our experiments we used A=2 (binary) and A=3. Each problem is generated by randomly picking C constraints out of $\binom{N}{A}$ total possible constraints, and picking T nogoods out of K^A maximum possible for each constraint.

We ran experiments with the following classes of CSPs :

1. $< 2, 10, 10, 45, T >$
2. $< 2, 15, 10, 50, T >$
3. $< 2, 25, 10, 37, T >$
4. $< 2, 15, 5, 105, T >$
5. $< 2, 20, 5, 100, T >$
6. $< 2, 40, 5, 55, T >$
7. $< 3, 50, 3, 75, T >$

The first 6 are binary CSPs and all are over-constrained. Problem classes 1 and 4 are complete graphs, while problems 2 and 5 have medium density, and problems 3 and 6 are sparse. Problems in set 7 have arity 3 and contain both solvable and unsolvable problems.

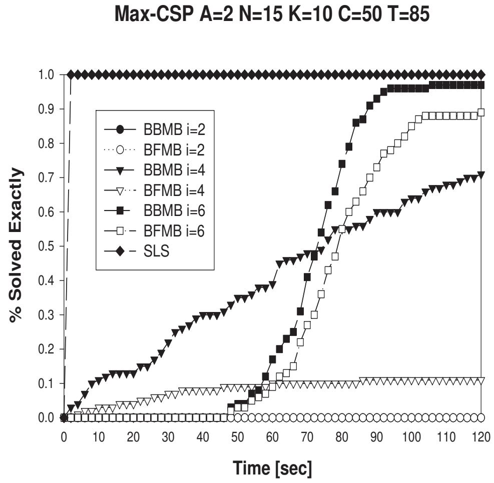


Figure 7: Max-CSP.

We used the min-degree heuristic for computing the ordering of variables. It places a variable with the smallest degree at the end of the ordering, connects all of its neighbors, removes the variable from the graph and repeats the whole procedure.

In addition to MB(i), BBMB(i) and BFMB(i) we ran, for comparison, two state of the art algorithms : PFC-MPRDAC as defined in [12] and a Stochastic Local Search (SLS) algorithm we developed for CSPs ([10]).

PFC-MPRDAC [12] is a Branch-and-Bound search algorithm. It uses a forward checking step based on a partitioning of unassigned variables into disjoint subsets of variables. This partitioning is used for computing a heuristic evaluation function that is used for determining variable and value ordering.

Stochastic Local Search (SLS) algorithms, such as GSAT [15, 18], starts from a randomly chosen complete instantiation of all the variables, and moves from one complete instantiation to the next. It is guided by a cost function that is the number of unsatisfied constraints in the current assignment. At each step, the value of the variable that leads to the greatest reduction of the cost function is changed. The algorithm stops when either the cost is zero (a *global minimum*), in which case the problem is solved, or when there is no way to improve the current assignment by changing just one variable (a *local minimum*). A number of heuristics have been reported in the literature, designed to overcome the problem of local minima, that greatly improve the performance of the basic scheme [13, 17, 16, 6]. In our implementation of SLS we use the basic greedy scheme combined with the constraint reweighting as introduced in [13]. In this algorithm, each constraint has a

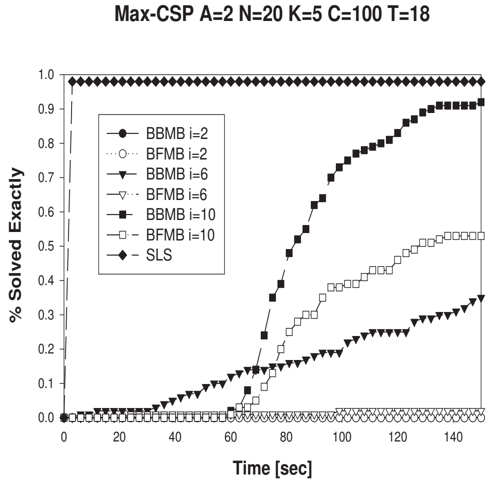


Figure 8: Max-CSP.

weight and the cost function is the weighted sum of unsatisfied constraints. Whenever the algorithm reaches a local minimum, it will increase the weights of unsatisfied constraints so that the current assignment will not be a local minimum of the new cost function.

SLS algorithms have become popular recently because they were shown to work well in practice for solving constraint satisfaction and satisfiability problems. They can sometimes solve problems larger than any complete algorithm can solve. They are naturally suitable for solving the Max-CSP problem since they use a cost function that tries to minimize the number of constraints satisfied.

We treat all algorithms as approximation algorithms. Algorithms BBMB and BFMB, if allowed to run until completion will solve all problems exactly. However, since we use a time-bound, both algorithms may return suboptimal solutions, especially for harder and larger instances. BBMB outputs its best solution, while BFMB, if interrupted, outputs the Mini-Bucket solution. Consequently BFMB is effective only as a complete algorithm.

As a measure of performance we used the accuracy ratio $opt = F_{alg}/F_{Max-CSP}$ between the value of the solution found by the test algorithm (F_{alg}) and the value of the optimal solution ($F_{Max-CSP}$), whenever $F_{Max-CSP}$ is available. We also record the running time of each algorithm.

We recorded the distribution of the accuracy measure *opt* over five predefined ranges : $opt \geq 0.95$, $opt \geq 0.5$, $opt \geq 0.2$, $opt \geq 0.01$ and $opt < 0.01$. However, we only report the number of problems that fall in the range 0.95. Problems in this range were solved optimally.

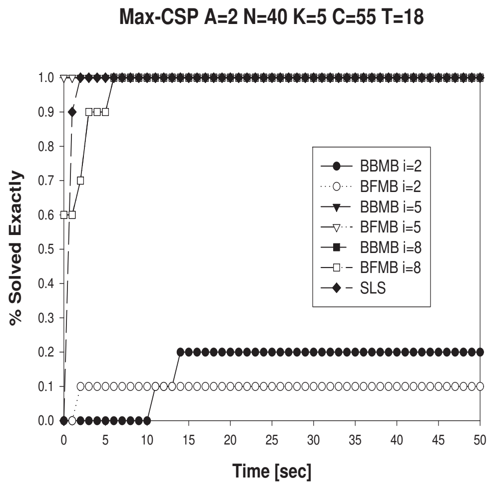


Figure 9: Max-CSP.

In addition, during the execution of both BBMB and BFMB we also stored the current upper bound U at regular time intervals. This allows reporting the accuracy of each algorithm as a function of time.

5 Results

Tables 1, 2, and 3 report results with random CSPs. Table 1 contains results with binary CSPs with domain size K=10, Table 2 contains results with binary CSPs with domain size K=5, and Table 3 contains CSPs with arity 3 with domain size K=3. Tables 1 and 2 contain three large blocks, each corresponding to a set of CSPs with a fixed number of constraints. Within each block, there are three small blocks each corresponding to a different constraint tightness, given in the first column. In columns 2 through 6 (Tables 1 and 3), and columns 2 through 7 (Table 2), we have results for MB, BBMB and BFMB (in different rows) for different i-bound. In Tables 1 and 2 we also have results for PFC-MRDAC and SLS in last two columns. Table 3 does not include results with PFC-MRDAC because it is implemented only for binary constraints. Each entry in the table gives the percentage of problems that fall in the 0.95 range and the average CPU time for these problems.

For example, looking at the middle block of the second large block in Table 1 (corresponding to binary CSPs with N=15, K=10, C=70 and T=85) we see that MB with i=bound 2 (column 2) solved only only 1% of the problems exactly in 0.02 seconds of CPU

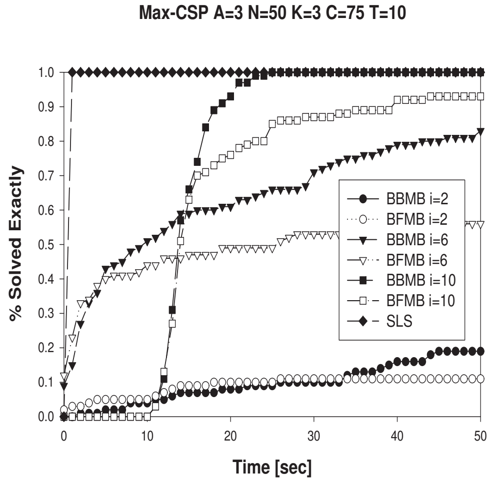


Figure 10: Max-CSP.

time. On the same set of problems BBMB, using Mini-Bucket heuristics, solved 20% of the problems optimally using 180 seconds of CPU time, while BFMB solved 1% of the problems exactly in 190 seconds. When moving to columns 3 through 6 in rows corresponding to the same set of problems, we see a gradual change caused by a higher level of Mini-Bucket heuristic (higher values of the i-bound). As expected, Mini-Bucket solves more problems, while using more time. Focusing on BBMB, we see that it solved all problems when the i-bound is 5 or 6, and its total running time as a function of time forms a U-shaped curve. At first (i=2) it is high (180), then as i-bound increases the total time decreases (when i=5 the total time is 28.7), but then as i-bound increases further the total time starts to increase again. The same behavior is shown for BFMB as well.

This demonstrates a trade-off between the amount of preprocessing performed by MB and the amount of subsequent search using the heuristic cost function generated by MB. The optimal balance between preprocessing and search corresponds to the value of i-bound at the bottom of the U-shaped curve. The added amount of search on top of MB can be estimated by $t_{search} = t_{total} - t_{MB}$. As i increases, the average search time t_{search} decreases, and the overall accuracy of the search algorithm increases (more problems fall within higher ranges of *opt*). However, as i increases, the time of MB preprocessing increases as well.

One crucial difference between BBMB and BFMB is that BBMB is an anytime algorithm - it always outputs an assignment, and as time increases, the solution improves. BFMB on the other hand only outputs a solution when it finds an optimal solution. In our experiments, if BFMB did not finish within the preset time bound, it returned the MB

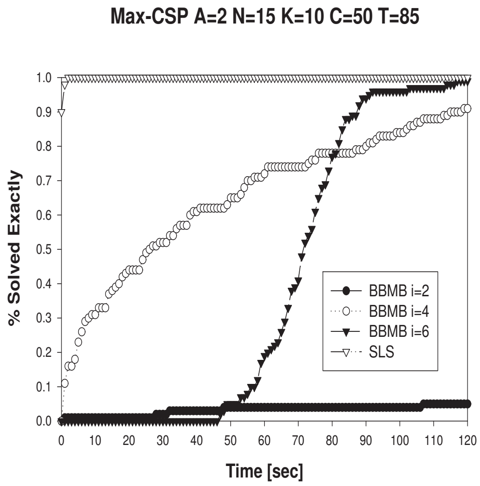


Figure 11: Max-CSP : anytime.

assignment.

From the data in the Tables we can see that the performance of BFMB is consistently worse than that of BBMB. BFMB(i) solves fewer problems than BBMB(i) and, on the average, takes longer on each problem. This is more pronounced when non-trivial amount of search is required (lower i-bound values) - when the heuristic is not exact. We speculate that this because there are large numbers of nodes on each level of the search tree with the same heuristic values. This will result in a large branching factor for BFMB(i).

Tables 1 and 2 also report results of PFC-MRDAC. When the constraint graph is dense (blocks 1 and 2) PFC-MRDAC is up to 2-3 times faster than the best performing BBMB. When the constraint graph is sparse (block 3) the best BBMB is up to an order of magnitude faster than PFC-MRDAC.

In Figures 7-10 we provide an alternative view of the performance of BBMB(i), BFMB(i) and SLS. Let $F_{BBMB(i)}(t)$ and $F_{BFMB(i)}(t)$ be the fraction of the problems solved completely by BBMB(i) and BFMB(i), respectively, by time t. Each graph in Figure 7 plots $F_{BBMB(i)}(t)$ and $F_{BFMB(i)}(t)$ for several values of i. These figures display trade-off between preprocessing and search in a clear manner. Clearly, if $F_{BBMB(i)}(t) > F_{BBMB(j)}(t)$ for all t, then BBMB(i) completely dominates BBMB(j). For example, in Figure 7 BBMB(4) completely dominates BBMB(2). When $F_{BBMB(i)}(t)$ and $F_{BBMB(j)}(t)$ intersect, they display a trade-off as a function of time. For example, if we have only few seconds, BBMB(4) is better than BBMB(6). However, when sufficient time is allowed, BBMB(6) is superior to BBMB(4).

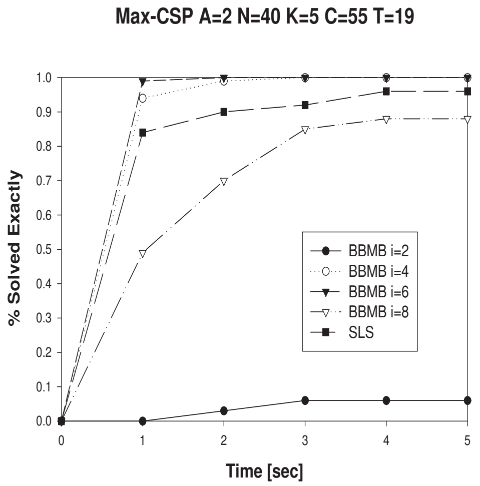


Figure 12: Max-CSP : anytime.

6 Anytime algorithms

In all Tables we also report results with SLS. On each problem, we set the SLS time bound the same as for BBMB/BFMB. At the end of its execution, SLS outputs the best assignment it found. For each set of problems, we report the number of times an SLS solution is optimal. To determine that, we used the optimal cost found by BBMB/BFMB. We also report the average time it took for the SLS algorithm to find an assignment with an optimal cost, as opposed to the completion time of SLS. Because of this, SLS time and BBMB/BFMB/PFC-MRDAC times reported in Tables 1-4 cannot be directly compared, since BBMB/BFMB/PFC-MRDAC times are completion times.

Figures 11 and 12 compare BBMB and SLS as anytime algorithms. Figure 11 (12) corresponds to one row in Table 1 (2). When the constraint graph is dense (Figure 11), SLS is substantially faster than BBMB. However, when the constraint graph is sparse (Figure 12), BBMB(4) and BBMB(6) are faster than SLS.

7 Summary and Conclusion

In this paper we evaluate the power of a scheme that generates search heuristics mechanically for solving the Max-CSP problem. The heuristics are extracted from the Mini-Bucket approximation method which allows controlled trade-off between computation and accuracy. Our experiments demonstrate the potential of this scheme in improving general search, showing that the Mini-Bucket heuristic's accuracy can be controlled to yield a trade-off between preprocessing and search. We demonstrate this property in the context of both Branch-and-Bound and Best-First search. Although the best threshold point cannot be predicted a priori, a preliminary empirical analysis can be informative when given a class of problems that is not too heterogeneous.

We show that search with Mini-Bucket heuristics can be competitive with state of the art algorithms for solving the Max-CSP problem. Although SLS was faster than BBMB/BFMB/PFC-MRDAC on many classes of problems we tried, complete methods like BBMB/BFMB/PFC-MRDAC have a number of advantages. Unlike complete methods, SLS cannot be used to prove optimality. It has been reported in the literature that there are classes of constraint satisfaction and satisfiability problems for which the performance of SLS is very poor while being easy for complete methods like backtracking. Due to the close nature between CSP and Max-CSP problems this is most likely true for Max-CSP problems as well.

References

[1] R. Dechter. Constraint networks. *Encyclopedia of Artificial Intelligence*, pages 276–285, 1992.

[2] R. Dechter. Bucket elimination: A unifying framework for probabilistic inference algorithms. In *Uncertainty in Artificial Intelligence (UAI-96)*, pages 211–219, 1996.

[3] R. Dechter. Bucket elimination: A unifying framework for reasoning. *Artificial Intelligence*, 113:41–85, 1999.

[4] R. Dechter and J. Pearl. Generalized best-first search strategies and the optimality of a*. *Journal of the ACM*, 32:506–536, 1985.

[5] R. Dechter and I. Rish. A scheme for approximating probabilistic inference. In *Proceedings of Uncertainty in Artificial Intelligence (UAI97)*, pages 132–141, 1997.

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

[7] K. Kask I. Rish and R. Dechter. Approximation algorithms for probabilistic decoding. In *Uncertainty in Artificial Intelligence (UAI-98)*, 1998.

[8] K. Kask and R. Dechter. Branch and bound with mini-bucket heuristics. *Proc. IJCAI*, 1999.

[9] K. Kask and R. Dechter. Mini-bucket heuristics for improved search. *Proc. UAI*, 1999.

[10] K. Kask and R. Dechter. Gsat and local consistency. In *International Joint Conference on Artificial Intelligence (IJCAI95)*, pages 616–622, Montreal, Canada, August 1995.

[11] R. Korf. Linear-space best-first search. In *Artificial Intelligence*, pages 41–78, 1993.

[12] J. Larossa and P. Meseguer. Partition-based lower bound for max-csp. *Proc. CP99*, 1999.

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

[14] J. Pearl. Heuristics: Intelligent search strategies. In *Addison-Wesley*, 1984.

[15] A.B. Philips S. Minton, M.D. Johnston and P. Laired. Solving large scale constraint satisfaction and scheduling problems using heuristic repair methods. In *National Conference on Artificial Intelligence (AAAI-90)*, pages 17–24, Anaheim, CA, 1990.

[16] B. Selman and H. Kautz. An empirical study of greedy local search for satisfiability testing. In *Proceedings of the Eleventh National Conference on Artificial Intelligence*, pages 46–51, 1993.

[17] B. Selman, H. Kautz, and B. Cohen. Noise strategies for local search. In *Proceedings of the Eleventh National Conference on Artificial Intelligence*, pages 337–343, 1994.

[18] B. Selman, H. Levesque, and D. Mitchell. A new method for solving hard satisfiability problems. *National Conference on Artificial Intelligence (AAAI-92)*, 1992.

[19] G. L. Nemhauser Z, Gu and M. W. P. Savelsbergh. Lifted flow covers for mized 0-1 integer programs. *Mathematical Programming*, pages 439–467, 1999.

T	MB BBMB FBMB i=2 #/time	MB BBMB FBMB i=3 #/time	MB BBMB FBMB i=4 #/time	MB BBMB FBMB i=5 #/time	MB BBMB FBMB i=6 #/time	PFC-MRDAC #/time	SLS #/time
N=10, K=10, C=45. Time bound 180 sec.							
84	2/0.02 26/180 2/189	4/0.11 98/90.7 4/184	6/0.87 100/11.7 78/65.7	10/7.25 100/10.0 98/17.9	16/56.7 100/57.6 100/59.3	100/4.00	100/0.21
85	0/- 20/180 0/-	3/0.11 100/80.1 5/124	2/0.89 100/11.6 82/54.4	8/7.45 100/9.62 100/18.7	10/57.3 100/57.3 100/58.9	100/3.95	100/0.21
86	0/- 24/180 0/-	2/0.11 100/87.1 4/154	4/0.91 100/10.5 84/56.6	10/7.12 100/9.38 98/16.1	14/55.2 100/57.2 100/58.1	100/3.84	100/0.23
N=15, K=10, C=50. Time bound 180 sec.							
84	0/- 10/180 0/-	0/- 60/161 0/-	3/0.96 90/50.1 21/70.5	6/8.77 100/26.2 65/49.8	14/78.3 100/86.2 97/89.7	100/13.5	100/0.47
85	1/0.02 20/180 1/190	2/0.13 68/164 5/184	3/0.95 98/79.0 16/82.0	7/8.12 100/28.7 63/59.6	17/71.0 100/74.9 97/82.8	100/13.2	100/0.43
86	0/- 0/- 0/-	0/- 50/173 0/-	1/0.98 100/58.2 20/74.5	3/9.05 100/32.1 60/52.3	15/81.5 100/83.6 86/86.7	100/16.8	100/0.48
N=25, K=10, C=37. Time bound 180 sec.							
84	0/- 36/114 3/56.9	7/0.10 99/4.42 94/8.67	30/0.60 100/0.77 100/1.28	84/3.41 100/3.70 100/3.77	99/9.74 100/9.93 100/9.93	100/4.16	99/0.90
85	0/- 31/88.6 9/51.1	10/0.10 100/7.55 89/17.1	34/0.60 100/0.75 100/1.34	79/3.20 100/3.31 100/3.34	99/9.36 100/9.58 100/9.59	100/7.51	100/1.04
86	1/0.02 37/122 6/90.3	9/0.09 99/4.74 91/14.8	44/0.60 100/0.73 100/1.13	88/3.09 100/3.19 100/3.23	100/8.75 100/8.75 100/8.77	100/4.10	100/1.72

Table 1: Max-CSP. A=2, K=10.

T	MB BBMB FBMB i=2 #/time	MB BBMB FBMB i=3 #/time	MB BBMB FBMB i=4 #/time	MB BBMB FBMB i=5 #/time	MB BBMB FBMB i=6 #/time	MB BBMB FBMB i=7 #/time	MB BBMB FBMB i=8 #/time	PFC- MRDAC #/time	SLS #/time
N=15, K=5, C=105. Time bound 180 sec.									
17	0/- 20/180 0/-	0/- 70/180 0/-	10/0.16 90/134 10/180	20/0.69 100/61.9 30/152	30/2.86 100/22.0 60/92.0	50/13.0 100/19.4 100/48.0	70/57.2 100/59.0 100/64.0	100/10.1	100/1.0
18	0/- 10/180 0/-	0/- 32/180 0/-	0/- 64/148 0/-	12/0.56 96/81.4 13/111	13/2.27 100/33.4 59/64.5	31/11.7 100/21.9 88/47.4	34/49.7 100/52.5 100/58.1	100/9.61	100/1.0
19	0/- 10/180 0/-	0/- 30/180 0/-	0/- 80/155 10/188	0/- 100/76.8 20/182	10/2.78 100/29.7 50/54.0	10/14.6 100/22.8 80/39.2	40/60.3 100/60.9 100/61.9	100/7.69	100/1.0
N=20, K=5, C=100. Time bound 180 sec.									
17	0/- 0/- 0/-	0/- 20/180 0/-	10/0.16 50/153 10/184	10/0.74 80/128 10/138	10/3.46 90/122 10/188	10/15.9 100/62.2 40/70.0	10/67.0 100/94.0 50/108	100/19.3	100/1.0
18	0/- 5/180 0/-	0/- 15/180 0/-	7/0.17 38/170 1/183	10/0.71 71/132 2/60.0	11/3.12 86/82.3 9/76.9	23/14.4 95/57.4 33/81.5	29/68.7 96/90.6 59/98.9	100/18.7	100/1.0
19	0/- 0/- 0/-	0/- 40/180 0/-	0/- 50/180 0/-	0/- 90/179 0/-	30/3.21 100/120 30/180	40/15.3 100/79.0 40/131	40/70.0 100/91.2 60/132	100/17.4	100/1.0
N=40, K=5, C=55. Time bound 180 sec.									
17	0/- 56/75.7 12/70.2	1/0.03 100/2.80 97/13.0	22/0.07 100/0.17 100/0.26	47/0.20 100/0.23 100/0.24	89/0.54 100/0.56 100/0.56	100/1.07 100/1.08 100/1.08	100/1.24 100/1.25 100/1.25	100/4.29	100/0.42
18	0/- 44/87.7 3/4.56	12/0.02 100/4.41 92/14.9	36/0.07 100/0.21 100/0.45	54/0.19 100/0.23 100/0.27	88/0.53 100/0.56 100/0.57	100/1.03 100/1.04 100/1.04	100/1.14 100/1.15 100/1.16	100/4.94	100/0.51
19	0/- 38/104 1/25.4	7/0.03 99/8.35 83/14.4	25/0.07 100/0.34 100/1.28	55/0.20 100/0.25 100/0.30	79/0.56 100/0.61 100/0.63	96/1.29 100/1.35 100/1.36	100/1.89 100/1.90 100/1.90	100/8.04	99/1.19

Table 2: Max-CSP. A=2, K=5.

T	MB BBMB FBMB i=2 #/time	MB BBMB FBMB i=4 #/time	MB BBMB FBMB i=6 #/time	MB BBMB FBMB i=8 #/time	MB BBMB FBMB i=10 #/time	SLS #/time
A=3, N=50, K=3, C=75. Time bound 180.						
10	0/- 40/62.7 17/41.7	0/- 80/60.4 50/49.5	0/- 98/26.3 67/24.6	0/- 100/9.10 97/8.83	0/- 100/15.5 97/20.3	100/0.5
14	0/- 10/141 10/83.8	0/- 20/63.6 10/2.24	0/- 80/57.4 20/30.7	0/- 100/30.7 60/23.4	0/- 100/22.8 90/31.3	100/1.5
14	0/- 0/- 0/-	0/- 0/- 0/-	0/- 40/79.7 10/182	0/- 40/30.9 20/16.4	0/- 80/52.8 50/40.2	100/3.7

Table 3: Max-CSP. A=3, N=50, K=3, C=75

T	MB BBMB FBMB i=2 #/time	MB BBMB FBMB i=4 #/time	MB BBMB FBMB i=6 #/time	MB BBMB FBMB i=8 #/time	MB BBMB FBMB i=10 #/time	PFC-MRDAC #/time	SLS #/time
N=100, K=3, C=200. Time bound 180 sec.							
1	70/0.03 90/12.5 80/0.03	90/0.06 100/0.07 100/0.07	100/0.32 100/0.33 100/0.33	100/2.15 100/2.16 100/2.15	100/15.1 100/15.1 100/15.1	100/0.08	100/0.01
2	0/- 0/- 0/-	0/- 0/- 0/-	10/0.34 40/38.0 20/0.76	10/2.03 80/19.6 70/19.8	40/15.7 100/22.6 100/33.2	100/757	100/0.02
3	0/- 0/- 0/-	0/- 0/- 0/-	0/- 60/72.4 30/39.2	0/- 70/27.7 60/28.7	10/16.2 100/24.5 90/28.9	100/2879	100/0.74
N=100, K=3, C=200. Time bound 600 sec.							
4	0/- 0/- 0/-	0/- 0/- 0/-	0/- 60/431 0/-	0/- 80/236 20/243	0/- 100/165 20/165	100/7320	100/5.32
5	0/- 0/- 0/-	0/- 0/- 0/-	0/- 10/180 0/-	0/- 60/108 10/-	0/- 70/92.9 0/-	100/7168	70/18.6
6	0/- 0/- 0/-	0/- 10/180 0/-	10/0.30 20/106 10/180	0/- 30/111 0/-	0/- 20/24.8 10/166	100/7533	30/34.6
7	0/- 0/- 0/-	0/- 0/- 0/-	10/0.33 10/180 10/183	0/- 30/101 0/-	10/16.4 40/115 0/-	100/4824	40/12.9
8	0/- 0/- 0/-	0/- 0/- 0/-	0/- 10/180 0/-	0/- 30/180 0/-	10/13.6 40/74.8 10/41.1	100/3.78	40/46.6

Table 4: Max-CSP. A=2, K=3.