New Search Heuristics for Max-CSP. *

Kalev Kask

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

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. These algorithms are compared with a state of the art complete algorithm as well as with the stochastic local search anytime approach, demonstrating superiority in some problem cases.

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. This heuristic generation scheme is based on the Mini-Bucket technique; a class of parameterized approximation algorithms for optimization tasks based on the bucket-elimination framework [1]. The mini-bucket approximation uses a controlling parameter which allows adjustable levels of accuracy and efficiency [3]. 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 [5]. 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 heuristic search algorithms that solve optimization tasks, using the task of finding the Most Probable Explanation in a Bayesian

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

network. We showed that the functions produced by the Mini-Bucket method can serve as heuristic evaluation functions for search [7], [6]. 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, all controlled by an input parameter.

In this paper we extend this approach to Max-CSP; 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. 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. We also compare the algorithms to two state of the art algorithms - PFC-MPRDAC [10] 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 [6], [7] 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 generation 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 4-6 present empirical evaluations, while Section 7 provides conclusions.

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 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 (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 that has 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 and don't have a solution. In such cases, it is desirable to find an assignment that satisfies a maximum number of constraints.

Definition 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 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.*

Example 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.

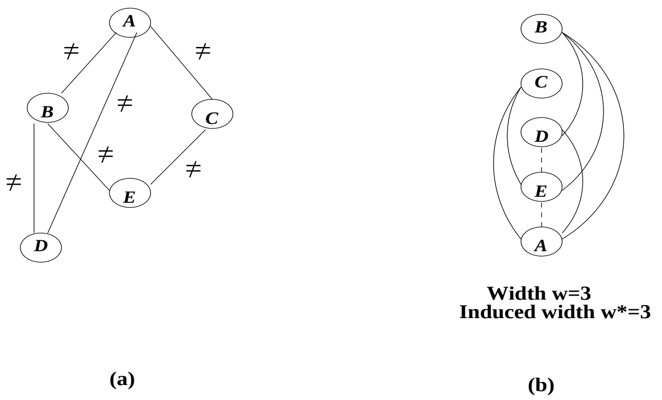


Fig. 1. a) Constraint graph of a graph coloring problem, b) an ordered graph along $d = (A, E, D, C, B)$.

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 [1]. 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. 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 (for more details see [8]). It can be shown that

Theorem 1. *[1] 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 [3] 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 $Q\prime = \{Q_1, ..., Q_r\}$ 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 2, is parameterized by this i-bound. The algorithm outputs not only a lower bound on the Max-CSP value 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.

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 $p = 1$ 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.

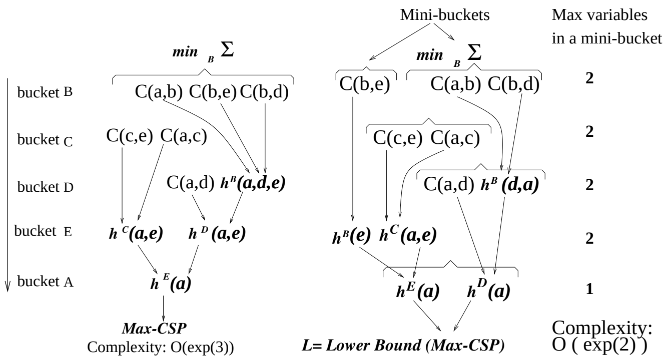
Fig. 2. Algorithm *MB-Max-CSP(i)*

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

Example 2. Figure 3 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.



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

Fig. 3. Execution of *Elim-Max-CSP* and *MB-Max-CSP(i)*

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:

- h_j^k stands for a function created by processing the j-th mini-bucket in $bucket_k$.
- $buckets(1..p)$ the union of all functions in the bucket of X_1 through the bucket of X_p.

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 5 (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 6. *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 $k > p$.*

Theorem 3 (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. 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.

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 space of partial assignments 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. 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.

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 in the search space 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 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 [2]. 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 binary CSPs. Each problem in this class is characterized by four parameters: $< N, K, C, T >$, where 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. Each problem is generated by randomly picking C constraints out of $\binom{N}{2}$ total possible constraints, and picking T nogoods out of K^2 maximum possible for each constraint.

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 for solving Max-CSP : PFC-MPRDAC as defined in [10] and a Stochastic Local Search (SLS) algorithm we developed for CSPs [9].

PFC-MPRDAC [10] 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.

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.

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

Here we evaluate performance of algorithms as complete ones. Tables 1-3 report results of experiments with three classes of over-constrained binary CSPs with domain sizes : K=10, K=5 and K=3. Tables 1 and 2 contain three blocks, each

T	MB BBMB BFMB i=2 #/time	MB BBMB BFMB i=3 #/time	MB BBMB BFMB i=4 #/time	MB BBMB BFMB i=5 #/time	MB BBMB BFMB i=6 #/time	PFC-MRDAC #/time
N=10, K=10, C=45. Time bound 180 sec. dense network.						
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
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
N=15, K=10, C=50. Time bound 180 sec. medium density.						
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
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
N=25, K=10, C=37. Time bound 180 sec. sparse network.						
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
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

Table 1. Search completion times for problems with 10 values. 100 instances.

corresponding to a set of CSPs with a fixed number of variables and constraints. Within each block, there are two 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 values of i-bound. In the last column we have results for PFC-MRDAC. Each entry in the table gives a percentage of the problems that were solved exactly within our time bound (fall in the 0.95 range), and the average CPU time for these problems.

For example, looking at the second block of the middle 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=2 (column 2) solved only only 1% of the problems exactly in 0.02 seconds of CPU 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

T	MB BBMB BFMB i=2 #/time	MB BBMB BFMB i=3 #/time	MB BBMB BFMB i=4 #/time	MB BBMB BFMB i=5 #/time	MB BBMB BFMB i=6 #/time	MB BBMB BFMB i=7 #/time	MB BBMB BFMB i=8 #/time	PFC- MRDAC #/time
N=15, K=5, C=105. Time bound 180 sec. dense network.								
18	0/-	0/-	0/-	12/0.56	13/2.27	31/11.7	34/49.7	
	10/180	32/180	64/148	96/81.4	100/33.4	**100/21.9**	100/52.5	100/9.61
	0/-	0/-	0/-	13/111	59/64.5	88/47.4	100/58.1	
19	0/-	0/-	0/-	0/-	5/2.78	12/14.6	40/60.3	
	16/180	40/180	77/155	100/76.8	100/29.7	**100/22.8**	100/60.9	100/7.69
	0/-	0/-	2/188	3/182	42/54.0	88/39.2	100/61.9	
N=20, K=5, C=100. Time bound 180 sec. medium density.								
18	0/-	0/-	7/0.17	10/0.71	11/3.12	23/14.4	29/68.7	
	5/180	15/180	38/170	71/132	86/82.3	95/57.4	96/90.6	100/18.7
	0/-	0/-	1/183	2/60.0	9/76.9	33/81.5	59/98.9	
19	0/-	0/-	0/-	4/0.70	4/3.21	4/14.9	4/70.7	
	4/180	24/180	56/160	64/121	84/97.5	96/85.4	92/90.0	100/17.4
	0/-	0/-	0/-	12/89.5	12/76.5	32/77.3	52/96.8	
N=40, K=5, C=55. Time bound 180 sec. sparse network.								
18	0/-	12/0.02	36/0.07	54/0.19	88/0.53	100/1.03	100/1.14	
	44/87.7	100/4.41	**100/0.21**	100/0.23	100/0.56	100/1.04	100/1.15	100/4.94
	3/4.56	92/14.9	100/0.45	**100/0.27**	100/0.57	100/1.04	100/1.16	
19	0/-	7/0.03	25/0.07	55/0.20	79/0.56	96/1.29	100/1.89	
	38/104	99/8.35	100/0.34	**100/0.25**	100/0.61	100/1.35	100/1.90	100/8.04
	1/25.4	83/14.4	100/1.28	**100/0.30**	100/0.63	100/1.36	100/1.90	

Table 2. Search completion times for problems with 5 values. 100 instances.

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

T	MB BBMB BFMB i=2 #/time	MB BBMB BFMB i=4 #/time	MB BBMB BFMB i=6 #/time	MB BBMB BFMB i=8 #/time	MB BBMB BFMB i=10 #/time	PFC-MRDAC #/time
N=100, K=3, C=200. Time bound 180 sec. sparse network.						
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
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
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
N=100, K=3, C=200. Time bound 600 sec. sparse network.						
4	0/- 0/- 0/-	0/- 0/- 0/-	0/- 60/431 0/-	0/- 80/236 20/243	0/- **100/165** 20/165	100/7320

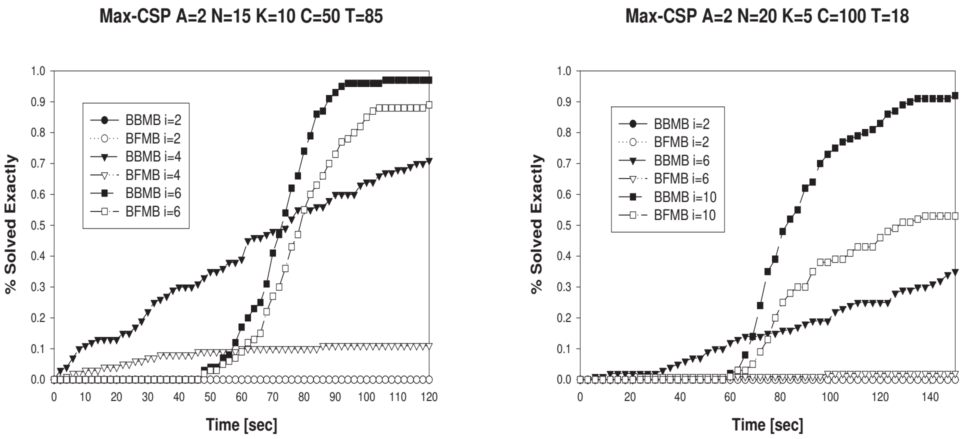
Table 3. Search completion times for problems with 3 values. 10 instances.

finds an optimal solution. In our experiments, if BFMB did not finish within the preset time bound, it returned the MB 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 weaker. These results are in contrast to the behavior we observed when using this scheme for optimization in belief networks [7]. We speculate that this is because, for Max-CSP, there are large numbers of frontier nodes with the same heuristic value.

Tables 1-3 also report results of PFC-MRDAC. When the constraint graph is dense PFC-MRDAC is up to 2-3 times faster than the best performing BBMB. When the constraint graph is sparse the best BBMB is up to two orders of magnitude faster than PFC-MRDAC. The superiority of our approach is most notable for larger problems (Table 3).

In Figure 4 we provide an alternative view of the performance of BBMB(i) and BFMB(i). 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 4 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 4a BBMB(4) completely dominates BBMB(2) (here BBMB(2) and BFMB(2) overlap). When $F_{BBMB(i)}(t)$ and $F_{BBMB(j)}(t)$ intersect, they display a trade-off as a function of time. For



(a) (b)

Fig. 4. Distribution of search completion.

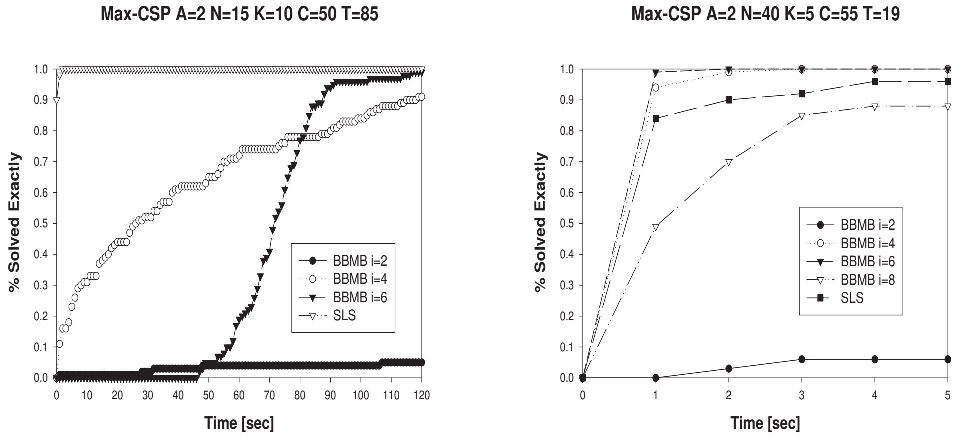
example, if we have less than 70 seconds, BBMB(4) is better than BBMB(6). However, when sufficient time is allowed, BBMB(6) is superior.

6 Anytime algorithms

Next we evaluate anytime performance of relevant algorithms - SLS, that is inherently incomplete and can never guarantee an optimal solution for Max-CSP, and BBMB, that returns a (suboptimal) solution any time during search.

A Stochastic Local Search (SLS) algorithm, such as GSAT [12], [15], 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 designed to overcome the problem of local minima [11], [14], [13], [4]. In our implementation of SLS we use the basic greedy scheme combined with the constraint re-weighting as introduced in [11]. In this algorithm, each constraint has a weight and the cost function is the weighted sum of unsatisfied constraints. Whenever the algorithm reaches a local minimum, it increases the weights of unsatisfied constraints.

An SLS algorithm for CSPs can immediately be applied to a Max-CSP problem. When measuring the performance of SLS we treat it as an anytime algorithm



(a) (b)

Fig. 5. Distribution of anytime performance.

- we report the fraction of problems solved exactly by time t as a function of t. To do that, we use the optimal cost found by BBMB.

In Figure 5 we present results comparing BBMB and SLS as anytime algorithms. Figure 5a (5b) corresponds to one row in Table 1 (2). When the constraint graph is dense (Figure 5a), SLS is substantially faster than BBMB. However, when the constraint graph is sparse (Figure 5b), BBMB(4) and BBMB(6) are faster than SLS. We should note that for most of the problem instances we ran where the graph exceeds a certain sparseness threshold, SLS exhibits impressive performance, arriving at an optimal solution within a few seconds.

7 Summary and Conclusion

The paper evaluates the power of a new scheme that generates search heuristics mechanically for solving a variety of optimization tasks. The approach was presented and evaluated for optimization queries over belief networks [7]. In this paper we extend the approach to the Max-CSP task and evaluate its potential on a variety of randomly generated constraint satisfaction problems. The basic idea is to extract heuristics for search from the Mini-Bucket approximation method which allows controlled trade-off between computation and accuracy. Our experiments demonstrate again 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 this approach can be competitive with state of the art algorithms for solving the Max-CSP problem. In particular, it outperformed a complete algorithm developed specifically for MAX-CSP on sparse constraint problems and is even competitive as anytime scheme against an SLS algorithm when the problems are sparse. Although, overall SLS performance was impressive on the classes of problems we experimented with, an SLS approach cannot prove optimality and therefore its termination time is speculative.

References

1. R. Dechter. Bucket elimination: A unifying framework for reasoning. *Artificial Intelligence*, 113:41–85, 1999.
2. R. Dechter and J. Pearl. Generalized best-first search strategies and the optimality of a*. *Journal of the ACM*, 32:506–536, 1985.
3. R. Dechter and I. Rish. A scheme for approximating probabilistic inference. In *Proceedings of Uncertainty in Artificial Intelligence (UAI97)*, pages 132–141, 1997.
4. 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.
5. K. Kask I. Rish and R. Dechter. Approximation algorithms for probabilistic decoding. In *Uncertainty in Artificial Intelligence (UAI-98)*, 1998.
6. K. Kask and R. Dechter. Branch and bound with mini-bucket heuristics. In *Proc. IJCAI*, pages 426–433, 1999.
7. K. Kask and R. Dechter. Mini-bucket heuristics for improved search. In *Proc. UAI*, 1999.
8. K. Kask and R. Dechter. Using mini-bucket heuristics for max-csp. *UCI Technical report*, 2000.
9. K. Kask and R. Dechter. Gsat and local consistency. In *International Joint Conference on Artificial Intelligence (IJCAI95)*, pages 616–622, Montreal, Canada, August 1995.
10. J. Larossa and P. Meseguer. Partition-based lower bound for max-csp. *Proc. CP99*, 1999.
11. 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.
12. 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.
13. 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.
14. 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.
15. B. Selman, H. Levesque, and D. Mitchell. A new method for solving hard satisfiability problems. *National Conference on Artificial Intelligence (AAAI-92)*, 1992.