Bucket Elimination: a Unifying Framework for Processing Hard and Soft Constraints

Rina Dechter
Department of Information and Computer Science
University of California, Irvine
dechter@ics.uci.edu

January 8, 1998

1 Introduction

The Constraint Satisfaction framework is quite restricted. Nevertheless, it is this restrictiveness that allowed the developments of very useful concepts such as constraint propagation (also know as "consistency enforcement"), through which various tractable subclasses had emerged and by which general purpose algorithms such as backtracking were improved [8,6,7,2]. However, real life problems frequently call for extending the basic model to allow nondeterminism as the representation of preferences among solutions. Such extensions relate the CSP model to known models for combinatorial optimization developed in the *Operation Research* community as well as to more recent frameworks such as *Probabilistic Networks* [9].

In this note I argue that extending the CSP model to a richer set of tasks can be done elegantly using a unifying framework which I call "bucket elimination". I believe that this framework will allow hybrids of two fundamental problem solving paradigms: elimination and conditioning, will address computational issues such as time-space tradeoffs, and will allow developing approximation algorithms, all within this general, and therefore, widely applicable framework. In the rest of this note I outline the basic framework.

2 Bucket elimination

Bucket elimination, is a unifying algorithmic framework that generalizes dynamic programming to accommodate many complex problem solving and reasoning activities. Algorithms such as directional-resolution for propositional satisfiability, adaptive-consistency for constraint satisfaction, Fourier and Gaussian elimination, for linear equalities and inequalities, and dynamic programming for combinatorial optimization, can be all accommodated within this framework [5]. It was recently demonstrated that many algorithms for probabilistic inference, such as belief updating, finding the most probable explanation, finding the maximum posteriori hypothesis and the maximum expected utility, can also be expressed as bucket-elimination algorithms [3].

The main virtues of this framework, are *simplicity* and *generality*. A complete specification of such algorithms is feasible without introducing extensive terminology, thus making the algorithms accessible to researchers across diverse areas. More importantly, uniformity brings up understanding, cross fertilization, and technology transfer between different disciplines. Indeed, all bucket-elimination algorithms are sufficiently similar so that any improvement to a single algorithm is therefore applicable to all others in that class. For example, by expressing probabilistic inference algorithms as bucket-elimination methods, their relationship to constraint satisfaction and dynamic programming becomes perspicuous and allows the knowledge accumulated in those areas to be utilized.

Bucket elimination algorithms process variables one by one in a given order. Processing a variable means generating an equivalent representation that excludes, or eliminates that variable. Such algorithms can be viewed as *knowledge-compilation* methods since they generate not merely an answer to a query, but also an equivalent representation of the input problem from which many queries are answerable in polynomial time. For illustration we include two bucket-elimination algorithms, *directional-resolution*, a procedure for satisfiability (Figure 1) [4], and *elim-bel*, an algorithm for belief-updating in probabilistic networks (Figure 2).

Normally, an input to a bucket elimination algorithm is a knowledge-base theory and a query specified by a collection of functions or relations, over subsets of variables (e.g., clauses for propositional satisfiability, constraints, or cost functions for constraint optimization, or conditional probability matrices for belief networks). In its first step, the algorithm partitions these functions into buckets, each associated with a single variable. Given a variable ordering, the bucket of a particular variable contains the functions defined on that variable, provided the function is not defined on variables higher in that ordering. Subsequently, buckets are processed from top to bottom. When the bucket of variable X is processed, an "elimination procedure" is performed over the functions in its bucket, yielding a new function that is defined over all the variables mentioned in the bucket, excluding X. This function summarizes the "effect" of X on the remainder of the problem. The new function is placed in a lower bucket.

An important property of variable elimination algorithms is that their performance can be bounded in advance using a graph parameter, called *induced width* (or tree-width), $w*$. In general, a given theory and its query can be associated with an *interaction graph* describing various dependencies between variables. The complexity of bucket-elimination is *time and space* exponential in the induced width of the problem's interaction graph. The size of the induced width varies with various variable orderings, and leads to different performance

directional resolution
Input: A cnf theory φ, an ordering $d = Q_1, ..., Q_n$,
Output: A decision of whether φ is satisfiable. If it is, a theory $E_d(\varphi)$, equivalent to φ; else, an empty directional extension.
1. **Initialize:** Generate an ordered partition of the clauses, $bucket_1, ..., bucket_n$, where $bucket_i$ contains all the clauses whose highest literal is Q_i.
2. **Backwards** For $p = n$ to 1 do:
• **If** $bucket_p$ contains a unit clause, perform only unit resolution. Put each resolvent in the appropriate bucket.
• **else,** resolve each pair $\{(\alpha \vee Q_p), (\beta \vee \neg Q_p)\} \subseteq bucket_p$. If $\gamma = \alpha \vee \beta$ is empty, return $E_d(\varphi) = \emptyset$, the theory is not satisfiable; else, determine the index of γ and add it to the appropriate bucket.
3. **Return** $E_d(\varphi) \Longleftarrow \bigcup_i bucket_i$.

Figure 1: Algorithm directional resolution

guarantees. In summary, bucket-elimination algorithms exploit the problem's structure; they are tractable for problems having a small w^*.

3 Conditioning and elimination

When a problem having a high induced-width is encountered, bucket-elimination may be unsuitable, primarily because of its extensive memory demand. To alleviate space complexity, another universal method for problem solving, called *conditioning*, can be incorporated.

Conditioning is a generic name for algorithms that search the space of partial value assignments, or partial conditionings. Conditioning means splitting a problem into subproblems based on a certain condition. In general, a subset of variables, conditioning variables, will be instantiated, thus generating a subproblem that can be solved by any means; if the resulting subproblem is unsolvable, or if more solutions are needed, the algorithm can try different assignments to the conditioning set. Algorithms such as *backtracking* and *branch and bound* may be viewed as conditioning algorithms, while *cutset-conditioning* applies conditioning to a subset of variables that form a cycle-cutset of the interaction graph, and solves the resulting subproblem by bucket-elimination [1,9].

The complexity of conditioning algorithms is exponential in the conditioning set, which is normally larger than the induced-width and which frequently includes all variables. However, the space complexity of conditioning is only linear. Moreover, empirical studies show that its average performance is often

Algorithm elim-bel
Input: A belief network $BN = \{P_1, ..., P_n\}$, and an ordering of the variables, $o = X_1, ..., X_n$.
Output: the belief of X_1 given evidence e.
1. **Initialize:** generate an ordered partition of the conditional probability matrices into buckets. $bucket_i$ contains all matrices whose highest variable is X_i. Put each observation in its bucket. Let $S_1, ..., S_j$ be the subset of variables in the processed bucket on which matrices (new or old) are defined.
2. **Backwards:** for $p \leftarrow n$ downto 1 do
for all the matrices $\lambda_1, \lambda_2, ..., \lambda_j$ in $bucket_p$ do
• **If** (bucket with observed variable) $X_p = x_p$ appear in bucket, **then** substitute $X_p = x_p$ in each matrix λ_i and put each in appropriate bucket.
• **else**, $U_p \leftarrow \bigcup_{i=1}^{j} S_i - \{X_p\}$ For all $U_p = u$, $\lambda_p(u) = \sum_{x_p} \Pi_{i=1}^{j} \lambda_i(x_p, u_{S_i})$.
Add λ_p to the largest index variable in U_p.
3. **Return** $Bel(x_1) = \alpha P(x_1) \cdot \Pi_i \lambda_i(x_1)$
(where the λ_i are in $bucket_1$, α is a normalizing constant.)

Figure 2: Algorithm elim-bel

far superior to that of bucket-elimination. This suggests that combining elimination with conditioning may be the key to improving reasoning. Particularly, tailoring the balance of elimination and conditioning to the problem instance may better utilize the benefits in each scheme on a case by case basis; we may have better performance guarantees, better space complexity, and better overall average performance.

Conditioning can be easily incorporated into the bucket elimination framework. Whenever a variable is processed, it can either be *eliminated* or *conditioned upon*, a decision that can be made statically or dynamically during run-time. In summary, we believe that the key to efficient reasoning across many areas, is tailoring the balance between conditioning and elimination to the problem instance by consulting its graph.

4 Approximations

Sometimes the interaction graph may suggest that the problem at hand is too difficult, no matter what hybrid algorithm we use. In such cases *approximation algorithms* should be attempted. We could elegantly generate approximation algorithms using the framework of bucket elimination by approximating the conditioning part, the elimination part, or both. Randomized greedy algorithms such as GSAT, that are currently popular for propositional satisfiability and constraint satisfaction, approximate conditioning; rather than systematically searching the space of all "conditionings", they search a subset in the hope that it is adequate to determine the solution. On the other hand, constraint propagation algorithms, like arc-, path-, and k-consistency, approximate the full bucket elimination algorithm (i.e., adaptive-consistency) for constraint satisfaction. These approaches, approximating conditioning and elimination, as well as their hybrids in the area of constraint processing, start to show promise. Taking advantage of the generality and uniformity of the bucket-elimination framework will allow, I believe, the extensions of conditioning and elimination principles and their approximation, to areas such as constraint optimization and probabilistic inference.

References

[1] R. Dechter. Enhancement schemes for constraint processing: Backjumping, learning and cutset decomposition. *Artificial Intelligence*, 41:273–312, 1990.

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

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

[4] R. Dechter and I. Rish. Directional resolution: The davis-putnam procedure, revisited. In *Principles of Knowledge Representation and Reasoning (KR-94)*, pages 134–145, 1994.

[5] R. Dechter and P. van Beek. Local and global relational consistency. In *Principles and Practice of Constraint programming (CP-95)*, pages 240–257, 1995.

[6] E. C. Freuder. A sufficient condition for backtrack-free search. *Journal of the ACM*, 29(1):24–32, 1982.

[7] A. K. Mackworth. Constraint satisfaction. *In Encyclopedia of Artificial Intelligence*, pages 285–293, 1992.

[8] U. Montanari. Networks of constraints: Fundamental properties and applications to picture processing. *Information Sciences*, 7(66):95–132, 1974.

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