Encodings of meet-semilattices using bit-vectors

1 Introduction

We have a meet-semilattice X, with a partial order operation $\sqsubseteq$ and a least upper bound operation $\sqcup$. Our task is to encode the nodes of the lattice using bit-vectors while preserving the partial order, success or failure of nodes to join, and joins of nodes, as well as keeping the dimension of the bit-vectors as small as possible. We can rephrase the problem as follows. Let Y be the power set of some finite set $Z = \{a_1, \ldots, a_m\}$. Then each element y of Y corresponds to a bit-vector, with the i-th entry being 1 if $a_i \in y$ and 0 otherwise. Thus a map $f : X \to Y$ gives a bit-vector encoding of X, and we would like to make $|Z| = m$ small. We are considering only encodings satisfying the following (for some nonnegative integer λ):

1. (partial order preservation) for $u, v \in X$, $u \sqsubseteq v$ iff $f(u) \supseteq f(v)$;
2. (success and failure preservation) for $u, v \in X$, $u \sqcup v$ exists iff $|f(u) \cap f(v)| > \lambda$ ($f(u)$ and $f(v)$ overlap in more than λ elements);
3. (join preservation) for $u, v, w \in X$, $u \sqcup v = w$ iff $f(u) \cap f(v) = f(w)$.

2 Is an optimum encoding possible?

2.1 A useful tool: loose block designs

Definition 1. A **loose block design (LBD)** with parameters n, b, λ; $r_1, \ldots, r_n$ (denoted as LBD$(n, b, \lambda; r_1, \ldots, r_n)$) is a collection of n sets $w_1, \ldots, w_n$, where

- each w_i is a subset of some set $C = \{c_1, \ldots, c_b\}$

- $|w_i| = r_i \ \forall i$,
- $|w_l \cap w_m| \leq \lambda \ \forall l \neq m$,

If each set is viewed as a bit-vector, the LBD can be considered as a 0-1 matrix, with n rows and b columns. For each j, we denote $k_j = |\{w_i | c_j \in w_i\}|$, so k_j is the number of 1's in the j-th column of the matrix. Thus, constructing such an LBD requires finding sets of required sizes and with pairwise overlap of no more than λ elements.

This is motivated by the well-known concept of a **balanced incomplete block design (BIBD)**. A BIBD with parameters n, b, r, k, λ (denoted as BIBD(n, b, r, k, λ)) can be viewed as a collection of sets $w_1, \ldots, w_n$, where $|w_i| = r \ \forall i$, $|\{i | a \in w_i\}| = k \ \forall a$, and $|w_l \cap w_m| = \lambda \ \forall l \neq m$ (so the overlap between any two sets has size *exactly* λ). A BIBD is thus a special case of the LBD.

What makes LBD's very useful for our problem is that any subset of X whose elements are pairwise nonjoinable has an image under f that forms an LBD, since by the failure preservation condition the elements of the image have pairwise overlap of size at most λ. We will prove some lower bounds on b for an LBD, which will allow us to find a bound for $|Z|$.

Theorem 1. An LBD(n, b, λ; $r_1, r_2, \ldots, r_n$) with column sizes $k_1, \ldots, k_b$ and $s = \sum_i r_i$ satisfies the following inequalities:

$$\lambda \binom{n}{2} \geq \sum_j \binom{k_j}{2} \geq \frac{1}{2} \lfloor \tfrac{s}{b} \rfloor [(s \mod b) + s - b] \tag{1}$$

$$\binom{b}{\lambda + 1} \geq \sum_i \binom{r_i}{\lambda + 1} \tag{2}$$

Proof. 1. Since the total overlap between all pairs (w_l, w_m) is at most $\lambda \binom{n}{2}$, we have

$$\begin{aligned} \lambda \binom{n}{2} &\geq \sum_{l<m} |w_l \cap w_m| \\ &= \sum_j |\{(w_l, w_m) | c_j \in w_l \cap w_m\}| \\ &= \sum_j \binom{k_j}{2}. \end{aligned}$$

Now consider column sizes $k'_1, \ldots, k'_b$ that are as close to equal as possible and in decreasing order, with $\sum_j k'_j = \sum_j k_j = s$. This is achieved when $k'_j = \lceil \frac{s}{b} \rceil$ for $j \leq (s \bmod b)$ and $k'_j = \lfloor \frac{s}{b} \rfloor$ for $j > (s \bmod b)$. Then $(k_1, \ldots, k_b)$ majorizes $(k'_1, \ldots, k'_b)$. Since $h(x) = \binom{x}{2}$ is a convex function, by Karamata's inequality we have

$$\begin{aligned}
\sum_j \binom{k_j}{2} &= \sum_j h(k_j) \geq \sum_j h(k'_j) \\
&= \sum_{j \leq (s \bmod b)} \binom{\lceil \frac{s}{b} \rceil}{2} + \sum_{j > (s \bmod b)} \binom{\lfloor \frac{s}{b} \rfloor}{2} \\
&= (s \bmod b)\binom{\lceil \frac{s}{b} \rceil}{2} + (b - s \bmod b)\binom{\lfloor \frac{s}{b} \rfloor}{2} \\
&= (s \bmod b)\binom{\lfloor \frac{s}{b} \rfloor + 1}{2} + (b - s \bmod b)\binom{\lfloor \frac{s}{b} \rfloor}{2} \\
&\quad [\text{since either } s \bmod b = 0 \text{ or } \lceil \tfrac{s}{b} \rceil = \lfloor \tfrac{s}{b} \rfloor + 1] \\
&= (s \bmod b)\left[\binom{\lfloor \frac{s}{b} \rfloor}{2} + \lfloor \tfrac{s}{b} \rfloor\right] + (b - s \bmod b)\binom{\lfloor \frac{s}{b} \rfloor}{2} \\
&= (s \mod b)\lfloor \tfrac{s}{b} \rfloor + b\binom{\lfloor \frac{s}{b} \rfloor}{2} \\
&= \frac{1}{2}\lfloor \tfrac{s}{b} \rfloor [2(s \mod b) + b\lfloor \tfrac{s}{b} \rfloor - b] \\
&= \frac{1}{2}\lfloor \tfrac{s}{b} \rfloor [2(s \mod b) + s - (s \bmod b) - b] \\
&= \frac{1}{2}\lfloor \tfrac{s}{b} \rfloor [(s \mod b) + s - b]
\end{aligned}$$

Thus,

$$\lambda\binom{n}{2} \geq \frac{1}{2}\lfloor \tfrac{s}{b} \rfloor [(s \mod b) + s - b]$$

There is in fact a simpler result given by Jensen's inequality, which simplifies to $b \geq s^2/(\lambda n(n-1) + s)$. However, in practice it turns out to be significantly weaker than the one given by Karamata's inequality.

2. Since $w_1, \ldots, w_b$ have pairwise intersections of size at most λ, a set $S \subseteq C$ of size $\lambda + 1$

will be a subset of at most one of the w_i. Thus, the number of sets of size $\lambda+1$ which are contained in at least one of the w_i is

$$\sum_i \binom{r_i}{\lambda+1} \leq |\{S|S \subset C, |S| = \lambda+1\}| = \binom{b}{\lambda+1}$$

since the total number of sets of size $\lambda+1$ is $\binom{b}{\lambda+1}$.

□

Theorem 2. Some useful properties of the LBD inequalities are:

1. If (1) is satisfied by $(n, b, \lambda; r_1, r_2, \ldots, r_n)$, then it is satisfied by $(n, b+1, \lambda; r_1, r_2, \ldots, r_n)$, in other words (1) is a lower bound on b.
2. If (1) is satisfied by $(n, b, \lambda; r_1, r_2, \ldots, r_n)$ with $\sum_i r_i = s > 0$, then it is satisfied by $(n, b, \lambda; r'_1, r'_2, \ldots, r'_n)$ with $\sum_i r'_i = s - 1$.
3. (2) is also a lower bound on b. If (2) is satisfied by $(n, b, \lambda; r_1, r_2, \ldots, r_n)$, then it is satisfied by $(n', b, \lambda; r_1, r_2, \ldots, r_{n'})$ for any $n' < n$.

We will refer to (1) as the *column bound*, and (2) as the *row bound*.

Proof. Let $h(x) = \binom{x}{2}$ as usual.

1. For each b, let $k'_1(b), \ldots, k'_b(b)$ be the column sizes that are as equal as possible and in decreasing order, and sum to s. Suppose (1) holds for $b = B$. Then,

$$\lambda\binom{n}{2} \geq \sum_{j=1}^{B} h(k'_j(B)) = h(k'_1(B)) + \cdots + h(k'_B(B)) + 0 \geq \sum_{j=1}^{B+1} h(k'_j(B+1))$$

by Karamata's inequality, since $(k'_1(B), \ldots, k'_B(B), 0)$ majorizes $(k'_1(B+1), \ldots, k'_{B+1}(B+1))$. Thus, (1) holds for $b = B+1$. (In other words, if we add an empty column, the inequality continues to hold.)

2. For each s, let $k'_1(s), \ldots, k'_b(s)$ be the column sizes that are as equal as possible and in decreasing order, and sum to s. Suppose (1) holds for $s = S$. Then,

$$(k'_1(S-1), \ldots, k'_b(S-1)) = (k'_1(S), \ldots, k'_{p-1}(S), k'_p(S) - 1, k'_{p+1}(S), \ldots, k'_b(S))$$

for some p. In other words, all the column sizes for S and $S-1$ are the same except that $k'_p(S) = k'_p(S-1)+1$ for some p. Then,

$$\lambda\binom{n}{2} \geq \sum_{j=1}^{b} h(k'_j(S)) \geq \sum_{j=1}^{b} h(k'_j(S-1))$$

since $h(x)$ is an increasing function. Thus, (1) holds for $s = S-1$.

3. It is obvious that (2) is a lower bound on b, since the left side is an increasing function of b, and the right side is constant with respect to b. If (2) is satisfied by $(n, b, \lambda; r_1, r_2, \ldots, r_n)$, then removing rows will decrease the right side while keeping the left side constant, so the inequality will still hold. (Note that for (1) this is not necessarily the case, since removing rows decreases both n and s, so both sides of the inequality decrease.)

□

The above results will be useful in showing the difficulty of finding an optimal encoding for X, and obtaining lower bounds on the encoding size.

2.2 Finding an optimal encoding is difficult

Since the image of any pairwise non-joinable subset of X forms an LBD, finding an optimal encoding for X (one with the smallest possible value of $|Z|$) is at least as hard as solving an LBD. Finding an LBD is in turn at least as difficult as solving a BIBD, which is a well-known hard problem.

Definition 2. BIBD decision problem: For given parameters n, b, r, k, λ, does there exist a BIBD(n, b, r, k, λ)?

LBD decision problem: For given parameters $n, b, \lambda; r_1, \ldots, r_n$, does there exist an LBD$(n, b', \lambda; r_1, \ldots, r_n)$, with $b' \leq b$?

Theorem 3. The BIBD decision problem reduces to the LBD decision problem.

Proof. It is well-known that the following are necessary conditions for a BIBD(n, b, r, k, λ) to exist: $b\binom{k}{2} = \lambda\binom{n}{2}$ and $nr = bk$.

Suppose there is no LBD$(n, b', \lambda;\ r_1 = r_2 = \cdots = r_n = r)$, where $b' \leq b$. Since a BIBD$(n, b,\ r, k, \lambda)$ would satisfy these constraints, it follows that there is no such BIBD.

Suppose there exists an LBD$(n, b', \lambda;\ r_1 = r_2 = \ldots = r_n = r)$ where $b' \leq b$. Let the column sizes be $k_1, \ldots, k_{b'}$ as usual. Then, by (1):

$$\lambda\binom{n}{2} \geq \sum_j \binom{k_j}{2}$$

Define $k_{b'+1} = \ldots = k_b = 0$. Then, since the function $h(x) = \binom{x}{2}$ is convex, Jensen's Inequality gives

$$\frac{\sum_{j=1}^{b} h(k_j)}{b} \geq h\left(\frac{\sum_{j=1}^{b} k_j}{b}\right) = h\left(\frac{nr}{b}\right) = h(k), \text{ so}$$

$$\sum_j \binom{k_j}{2} \geq b\binom{k}{2} \text{ with equality iff } k_1 = \cdots = k_b = k.$$

Putting the inequalities together,

$$\lambda\binom{n}{2} \geq \sum_j \binom{k_j}{2} \geq b\binom{k}{2}.$$

However, since from the BIBD conditions $b\binom{k}{2} = \lambda\binom{n}{2}$, we must have equality everywhere. Thus,

$$k_1 = \ldots = k_b = k,\ b' = b,$$
$$\lambda\binom{n}{2} = \sum_j \binom{k_j}{2}$$

(so every pair of rows has overlap exactly λ). Thus the given LBD is the required BIBD. □

We do not have a proof that the BIBD decision problem is NP-complete, but it is widely considered difficult. In particular, there is an important open problem in combinatorics that reduces to it, namely the existence of a finite projective plane of order 12, which is equivalent to the existence of a BIBD$(157, 157,\ 13, 13, 1)$[1]. The existence of a finite projective plane of order 10 (a BIBD$(111, 111,\ 11, 11, 1)$) was disproven after 2000 hours of computation on a

[1]Lindner, C. C. and Rodger, C. A. "Design Theory". Boca Raton, FL: CRC Press, 1997.

Cray computer[2]. A question related to the BIBD decision problem - whether it is possible to construct a BIBD from a given partial BIBD (with several rows, or several columns, already filled in) - has been proven NP-complete[3]. This shows that we cannot hope to find a fast optimal algorithm for assigning the bit-vectors, although there is still the possibility of finding one that performs near optimal. We see that it is permissible to use something that doesn't always come up with an optimal solution, e.g. a general constraint solver.

3 Lower bounds

3.1 An algorithm for computing a lower bound for $|Z|$

We perform a topological sort of the semilattice X, and go through it in reverse order (from maximal nodes to bottom), recursively computing, for each node u, a lower bound $l(u)$ on the size of $f(u)$. If u has no successors, we set the lower bound to be $l(u) = \lambda + 1$, since $u = u \sqcup u$ and so $|f(u)| > \lambda$. Otherwise, we consider the set $D = \{v | v \sqsupseteq u\}$. All of these nodes are after u in the topological sort, so by the time we consider u, $l(v)$ has already been computed for every $v \in D$.

Consider a pairwise non-joinable subset $T = \{t_1, \ldots, t_n\}$ of D. Then $\forall l, m \ |f(t_l) \cap f(t_m)| \leq \lambda$, so $f(t_1), \ldots, f(t_n)$ form an LBD$(n, b, \lambda; r_1, \ldots, r_n)$, with $r_i \geq |l(t_i)| \ \forall i$. The LBD inequalities give lower bounds on b given row sizes $l(t_1), \ldots, l(t_n)$: let $c(T)$ and $r(T)$ be the smallest values of b that satisfy the column bound and row bound respectively, and let $l(T) = \max(c(T), r(T))$. Since $\forall v \in D \ f(v) \subseteq f(u)$, $\bigcup_i f(t_i) \subseteq f(u)$, so $|f(u)| \geq l(T)$. Thus, we can set

$$l(u) = \max_{T \subseteq D} l(T),$$

and compute lower bounds for all nodes in X in this way.

In practice, it is usually not possible to go through all non-joinable subsets T of D, since there are tremendously many of them. We instead look at a smaller number of subsets that give relatively strong lower bounds.

Call a non-joinable subset *maximal* if it is not contained in any larger non-joinable subset.

[2]Lam, C. W. H. "The Search for a Finite Projective Plane of Order 10." American Mathematical Monthly 98, 305-318, 1991.

[3]Kaski, P. and Ostergard, P. R. J. "Classification Algorithms for Codes and Designs". Springer, 2006.

Then any non-joinable subset of D is contained in a maximal subset M. We define the n-th *suffix* of M as the n-element subset $S = \{s_1, \ldots, s_n\}$ of M that maximizes $\sum_i l(s_i)$. The n-th suffix is obtained by dropping the element of M with the smallest lower bound value until n elements are left. In our algorithm, we will only compute lower bounds for maximal subsets of D and their suffixes.

Let T be any non-joinable subset of D, $|T| = n$. Let M be the maximal subset T is contained in, and let S be the n-th suffix of M. By Theorem 2, the $r(T) \leq r(M)$ (by part 3), and $c(T) \leq c(S)$ (by part 2). Then,

$$\begin{aligned} l(T) &= \max(c(T), r(T)) \\ &\leq \max(c(S), r(M)) \\ &\leq \max(c(S), r(S), c(M), r(M)) \\ &= \max(l(S), l(M)) \end{aligned}$$

Thus, considering only maximal subsets and their suffixes is enough to compute the best lower bound. Unfortunately, this optimization is not sufficient to make the algorithm feasible - the number of relevant subsets is usually still too large. Thus, we decided to only consider subsets of the set of joins of successors of a, instead of using all of D. This gives a reasonable number of subsets to consider. Below we show the resulting lower bounds for the English Resource Grammar (ERG).

3.2 Lower bound results for the English Resource Grammar (ERG)

λ	0	1	2	3	4	5	6	7
bound	2039	540	429	388	366	353	342	337
λ	8	9	10	11	12	13	14	15-20
bound	333	329	326	323	319	315	313	311
λ	21-26	27-29	30-79	80	81	82	83	84
bound	310	309	308	309	310	311	312	313

4 Encoding techniques

4.1 Componentization of the lattice

Definition 3. A vertex is a *choke-vertex* if there are no paths in the Hasse diagram from any of its successors to any of its predecessors that don't pass through the vertex. (Note that by this definition, $\perp$, as well as the maximal nodes of X, are choke-vertices.) Equivalently, a vertex is a choke-vertex if there's no node with which it's joinable but not comparable.

We can use these choke-vertices to break the semilattice into edge-disjoint *components* such that:

- each component is a meet-semilattice, just like our semilattice X itself;
- within each component, any choke-vertex must be either the bottom node or a *maximal* node of the component;
- the components form a tree;
- each vertex is in exactly one component if it's not a choke-vertex, and two components (one above, whose bottom it is, and one below) if it is a choke-vertex (unless the node is $\perp$, which only has a component above);
- given λ, optimally encoding X is equivalent to optimally encoding each component, going down the tree, in such a way that for each choke-vertex, the encoding size below is at least the encoding size above.

The way to construct this is as follows: for each node, let the bottom of its component (the component below, if it's a choke-vertex) be the largest choke-vertex below it that occurs on every path from the root to this node. For each edge, then, the component is the unique one that the endpoints of the edge are both contained in. This is quite useful because now the semilattice can be encoded one component at a time. For some special types of components we have techniques that help encode them efficiently, and sometimes optimally.

4.2 Special cases

4.2.1 Component with a top element

If the component C has a top, i.e. a unique maximal node, we can use the classical Ait-Kaci encoding for $\lambda = 0$ to achieve an optimal encoding for the component.

Definition 4. Consider the set $\{x_1, \ldots, x_m\}$ of meet-irreducible nodes in C, i.e. the nodes that have at most one successor. Then $g(x) = \{i | x_i \sqsupseteq x\}$ is an m-dimensional encoding of C, called the *Ait-Kaci encoding*.

We generalize this encoding to the case $\lambda \geq 0$ by creating λ new bits and adding them to each node in the encoding. Letting S be the set of those new bits, we now have $g(x) = \{i | x_i \sqsupseteq x\} \cup S$.

Theorem 4. The generalized Ait-Kaci encoding satisfies the encoding rules.

Proof. 1. (partial order preservation) If $u \sqsubseteq v$, then $x_i \sqsupseteq v \Rightarrow x_i \sqsupseteq u$, so

$$g(v) = \{i | x_i \sqsupseteq v\} \cup S \subseteq \{i | x_i \sqsupseteq u\} \cup S = g(u).$$

2. (success and failure preservation) If $u \sqcup v = w$ exists, then there exists a maximal node $x_j \sqsupseteq w$, so

$$|g(u) \cap g(v)| = |S \cup (\{i | x_i \sqsupseteq u\} \cap \{i | x_i \sqsupseteq v\})| \geq |S \cup \{j\}| = \lambda + 1.$$

If $u \sqcup v$ does not exist, then there is no x_j such that $x_j \sqsupseteq u$ and $x_j \sqsupseteq v$, so

$$|g(u) \cap g(v)| = |S \cup (\{i | x_i \sqsupseteq u\} \cap \{i | x_i \sqsupseteq v\})| = |S \cup \emptyset| = |S| = \lambda.$$

In particular, since $u = u \sqcup u$, $|g(u)| = |g(u) \cap g(u)| > \lambda$.

3. (join preservation) If $u \sqcup v = w$, then

$$g(u) \cap g(v) = S \cup (\{i | x_i \sqsupseteq u\} \cap \{i | x_i \sqsupseteq v\}) = S \cup \{i | x_i \sqsupseteq w\} = g(w).$$

If $g(u) \cap g(v) = g(w)$, then $u \sqcup v = w'$ exists, since otherwise we would have $|g(w)| = |g(u) \cap g(v)| \leq \lambda$. Then $\{i|x_i \sqsupseteq u\} \cap \{i|x_i \sqsupseteq v\} = \{i|x_i \sqsupseteq w'\}$, so $g(w) = g(w')$. We will prove that the Ait-Kaci encoding is injective, and thus $w = w'$.

Suppose that $\{x|x \sqsupseteq w\} \not\subseteq \{x|x \sqsupseteq w'\}$. Let z be a maximal element of $\{x|x \sqsupseteq w\} \setminus \{x|x \sqsupseteq w'\}$, in other words all successors of z are in $\{x|x \sqsupseteq w'\}$. If z is meet-irreducible, then $z = x_j$ with $j \in g(w)$ and $j \notin g(w')$, so $g(w) \neq g(w')$, contradiction. Thus, z has at least two successors z_1 and z_2. Since $z_1 \sqsupseteq w'$ and $z_2 \sqsupseteq w'$, we have $z = z_1 \cap z_2 \sqsupseteq w'$ by meet-semilattice properties. However, by assumption $z \notin \{x|x \sqsupseteq w'\}$, contradiction. Similarly, $\{x|x \sqsupseteq w'\} \subseteq \{x|x \sqsupseteq w\}$. Thus, $\{x|x \sqsupseteq w\} = \{x|x \sqsupseteq w'\}$, so $w = w' = u \sqcup v$. □

Theorem 5. The Ait-Kaci encoding is optimal for $\lambda = 0$, and the generalized Ait-Kaci encoding is optimal if C has a top element.

Proof. Consider an arbitrary encoding h of C for $\lambda = 0$ that satisfies the encoding rules. For each meet-irreducible node u, we choose a bit β_u in its encoding that is not present in the encodings of its successors. If u is maximal, then it has no successors, so any bit of u satisfies this. If u has a successor u', then $h(u) = h(u')$ violates join preservation, so there is a bit contained in $h(u)$ but not in $h(u')$. Suppose that for two distinct meet-irreducible nodes u and v, we have $\beta_u = \beta_v$. Then $|h(u) \cap h(v)| > 0$, so by the encoding rules $u \sqcup v = w$ exists, and $\beta_u \in h(w)$. Since u and v are joinable, one of them has a successor, and this successor contains β_u, contradiction. Therefore, the encoding h has at least m bits, where m is the number of meet-irreducible nodes in C. Since the Ait-Kaci encoding has m bits, it is optimal.

If C has a top node, the encodings of all the nodes in C must contain the bits of the top node. Removing λ of these bits from all the nodes in C yields an encoding for $\lambda = 0$. Thus, if an encoding of C for $\lambda > 0$ has less than $m + \lambda$ bits, then the encoding obtained by removing λ common bits from it will have less than m bits, and thus better than the optimum for $\lambda = 0$. Therefore, the optimal encoding for $\lambda > 0$ has $m + \lambda$ bits, and so the generalized Ait-Kaci encoding is optimal. □

When the component doesn't have a top, the Ait-Kaci encoding is not optimal, but can be used as a fallback if the constraint solver fails.

4.2.2 Component consisting of a node and its successor set $A \cup B$ where no node in A is joinable with any node in B

Suppose the optimal encoding for this component has b bits. A valid encoding of the component is given by optimally encoding A using b_A bits, and then optimally encoding B using b_B bits where λ of them have been used for A and the rest are new. This preserves the non-joinability of nodes in A and nodes in B, since only λ bits are shared. This encoding uses $b_A + b_B - \lambda$ bits, and the number of extra bits is $b_A + b_B - \lambda - b \leq b_B - \lambda$, since obviously $b_A \leq b$.

This bound is fairly tight if we choose B that has a compact encoding, i.e. with b_B small. The number of nodes in B can still be large, however. This means that even if the component is intractable, it might be possible to split it into A and B that are both encodable by a constraint solver, and still lose only a few bits.

4.3 Unary leaf removal

4.3.1 Algorithm

Definition 5. A *unary leaf (UL)* is a node with no successors and only one predecessor, i.e. a join-irreducible maximal node. The encoding of a leaf always has size $\lambda + 1$.

For any component, we can remove the ULs, encode the component without them, and then augment the encoding to fit them. This is very useful, since it is often the case that an originally intractable component can be encoded using a constraint solver once the ULs are removed. Then, we can easily add the ULs to the encoding, because a UL can be assigned any $(\lambda + 1)$-tuple of its predecessor's bits that has not yet been used for another successor of its predecessor, while satisfying the encoding rules.

The algorithm goes through the component from top to bottom, and for each node it adds in the UL successors of that node, assigning a free $(\lambda + 1)$-tuple of bits to each UL. Whenever all the possible $(\lambda + 1)$-tuples are taken, it increments the node's encoding size (by adding a bit to the encodings of all its ancestors). Once a new bit is added, any $(\lambda + 1)$-tuple containing this bit is free, so all these tuples can be used to encode ULs. We continue incrementing the encoding size of the node until all its UL successors are encoded.

4.3.2 Distance from the optimum

While we do not have bounds of the number of extra bits for an entire component, we have bounds for the number of extra bits for a single node, given the encodings of its non-UL successors (some of which might be joinable). Let u be the number of UL successors to be added to this node, and let $\tilde{b}$ be the size of the optimal encoding for the ULs only, so $\tilde{b}$ is the smallest number such that $\binom{\tilde{b}}{\lambda+1} \geq u$.

Let the encoding of the non-ULs have b' bits, and the optimal encoding of the non-ULs have b'^* bits, where $b' \leq b'^* + x$, so the non-UL encoding is at most x bits away from the optimum. Let the encoding given by our method have b bits, and the optimal encoding of all the successors have b^* bits. We are interested in the distance of our encoding from the optimal encoding for all the successors, i.e. $b - b^*$.

Theorem 6. The number of extra bits in our encoding satisfies

$$b - b^* \leq \min(\left\lceil u/\binom{b'}{\lambda} \right\rceil + x, \tilde{b} - \lambda + x)$$

Proof. 1. Let r be the number of free $(\lambda+1)$-tuples in the encoding of the non-ULs. Then,

$$r + \binom{b'}{\lambda} + \binom{b'+1}{\lambda} + \cdots + \binom{b-1}{\lambda} \geq u \tag{3}$$

If $b > b'$, then since b is the smallest number such that (3) holds, we have

$$\begin{aligned} u &> r + \binom{b'}{\lambda} + \binom{b'+1}{\lambda} + \cdots + \binom{b-2}{\lambda} \\ &\geq 0 + \sum_{l=b'}^{b-2} \binom{l}{\lambda} \\ &\geq \sum_{l=b'}^{b-2} \binom{b'}{\lambda} \\ &= (b - b' - 1)\binom{b'}{\lambda}. \end{aligned}$$

Thus, $b - b' < u/\binom{b'}{\lambda} + 1$. Since $b - b'$ is an integer, $b - b' \leq \left\lceil u/\binom{b'}{\lambda} \right\rceil$. This also trivially

holds when $b = b'$. Thus, $b - b^* \leq b - b'^* \leq b - b' + x \leq \left\lceil u/\binom{b'}{\lambda} \right\rceil + x$.

2. Consider the optimal encoding for the u ULs by themselves, with $\tilde{b}$ bits. We combine this encoding with the b'-bit encoding of the non-ULs, with an overlap of λ bits between the two encodings. The resulting encoding of all the successors has $b' + \tilde{b} - \lambda$ bits. Since the b-bit encoding given by our method actually does no worse than this one, fitting the ULs at least as well into the non-UL encoding, $b \leq b' + \tilde{b} - \lambda$. Since $b' \leq b^* + x$, we have $b - b^* \leq \tilde{b} - \lambda + x$.

□

A special case occurs when all the non-UL successors are pairwise non-joinable, so their encodings have overlap at most λ. Let the encoding sizes be $r_1, \ldots, r_n$. Then a $(\lambda + 1)$-tuple can be contained in at most one of them, so the number of free $(\lambda + 1)$-tuples is

$$r = \binom{b'}{\lambda + 1} - \sum_i \binom{r_i}{\lambda + 1}$$

Thus, r depends only on the encoding sizes of the non-ULs, not on the encodings themselves, so r is fixed. Let b_u be the smallest value of b that satisfies (3). Then $b = \max(b', b_u)$, while $b^* \geq \max(b'^*, b_u)$. Since taking the optimal non-UL encoding with b'^* bits and adding the u ULs gives a valid encoding for all the successors with $\max(b'^*, b_u)$ bits, we must have $b^* = \max(b'^*, b_u)$. Then, $b - b^* \leq b' - b'^* = x$. Usually the distance will be much less, for example if $b' \leq b_u$ and $b'^* \leq b_u$ then $b = b^* = b_u$, so the distance is 0. Thus our method in fact improves on the original distance from the optimum.

4.3.3 Analysis

Unfortunately, it is not clear how to obtain a bound for the distance of a whole component's encoding from the optimum by using these bounds for each node in the component. This is because these bounds measure the distance of the encoding of this node's successors from the optimal encoding of this node's successors, given a particular set of encoding sizes. However, in the optimal encoding of the component, the encoding sizes of these successors might be different than those given by our method. In other words, the number of extra bits per node

depends on how well its successors have been encoded, so it is not possible to consider each node independently and simply add up the bounds to obtain the component bound.

The good news is that these bounds are quite weak, so in practice the algorithm usually does much better. Observe that neither of the bounds takes into account the free $(\lambda + 1)$-tuples available before adding more bits, and there is usually quite a substantial number of those. This is especially true if some of the node's successors have already been augmented with extra bits by the algorithm, because then there are some bits used by only one of the successors, thus generating a lot of free $(\lambda + 1)$-tuples. In practice, the algorithm loses at most 0-1 bits per component, and in fact usually decreases the original distance from the optimum.

4.4 Splitting a component

It is sometimes the case that removing the ULs is not enough to make a component encodable by a constraint solver. The following is an idea for splitting the component into two pieces, encoding them separately, and then combining the encodings. We don't have bounds on the distance of the resulting encoding from the optimum, but if this technique makes some large intractable components encodable, this will already be good progress. The technique has not yet been tested on the ERG, so its effectiveness is unknown.

Let C be our component, and let $D_x = \{z \in C | z \sqsupseteq x\}$. Let u be the bottom node of C, and let $v \in C$. Let S be the set of nodes in D_v that are successors of nodes in $D_u \setminus D_v$. Consider the case when S is part of a chain starting at v, of the following form: $(v = w_0) \rightarrow w_1 \rightarrow \cdots \rightarrow w_k$. Then it is possible to construct an encoding for D_v and an encoding for $T = \{D_u \setminus D_v\} \cup S$, and combine them to obtain an encoding for $D_u = C$.

4.4.1 Constructing the encoding

Suppose we have an encoding g for D_v that imposes a size c_i on each node $w_i \in S$ (observe that $i < j \Rightarrow c_i > c_j$). We now need an encoding h for T that satisfies $h(w_i) = c'_i \geq c_i$. Since only w_k is a maximal node in T whose size can be specified, for $w_0, \ldots, w_{k-1}$ we create a set $Q = \{q_1, \ldots, q_k\}$ of virtual nodes such that q_{i+1} is a successor of w_i. We specify the size of q_{i+1} as $d_{i+1} = c_i - c_{i+1} + \lambda$, which ensures that $c'_i \geq c_i$, as follows by induction. Since we

have enforced that $c'_k \geq c_k$, and assuming $c'_{i+1} \geq c_{i+1}$, we have that

$$c'_i \geq c'_{i+1} + d_{i+1} - \lambda = c'_{i+1} + (c_i - c_{i+1} + \lambda) - \lambda = c'_{i+1} - c_{i+1} + c_i \geq c_i.$$

In fact, we have something stronger, namely

$$c'_i - c_i \geq (c'_{i+1} - c_{i+1} + c_i) - c_i = c'_{i+1} - c_{i+1}.$$

Now we combine the encodings g and h to obtain an encoding f. Suppose that $g(w_i) = \{\beta_1, \ldots, \beta_{c_i}\}$. This is valid, since it satisfies $g(w_i) \supseteq g(w_{i+1})$ and $|g(w_i)| = c_i$. To make each w_i have size c'_i, we consider new bits $\beta'_1, \ldots, \beta'_{c'_0 - c_0}$, and add $\beta'_1, \ldots, \beta'_{c'_i - c_i}$ to $g(w_i)$. Since, for $i < k$, $c'_i - c_i \geq c'_{i+1} - c_{i+1}$, we still have $g(w_i) \supseteq g(w_{i+1})$. Now we propagate the extra bits into D_v, adding all the extra bits of w_i to every node $x \in D_v, x \sqsubseteq w_i$. The result is clearly a working encoding of D_v where w_i has c'_i bits, so these bits can be matched up to the bits used for the w_i in the encoding of T. In the end, the bits of D_v are a subset of the bits of T, since all the bits of v were matched to bits of T.

4.4.2 Why the encoding is valid

Theorem 7. The encoding f of C satisfies the encoding rules.

Proof.

1. Partial order is preserved within T because the encoding of T has not changed, and within D_v because the bits added to each w_i were propagated to its ancestors. Now let $x \in T \setminus S, y \in D_v \setminus S$. Since $x \notin D_v$, we cannot have $y \sqsubseteq x$, so assume $x \sqsubseteq y$. Then any path from x to y in the Hasse diagram must pass through an element w_i of S. Then, since the encoding is valid within T and D_v, $f(x) \supseteq f(w_i) \supseteq f(y)$.

2. Similarly to the above, success and failure are preserved within T and within D_v. Let $x \in T \setminus S, y \in D_v \setminus S$. Suppose $z = x \sqcup y$ exists. Clearly, since $y \in D_v$, we have $z \in D_v$. Thus a path from x to z must pass through some $w_i \in S$, so $w_i \sqcup y$ exists. Then, since the encoding is valid within D_v, $|f(w_i) \cap f(y)| \geq \lambda$. Since $f(x) \supseteq f(w_i)$, $|f(x) \cap f(y)| \geq \lambda$.

   Now suppose $|f(x) \cap f(y)| \geq \lambda$. Since $y \sqsupseteq v$, $f(v) \supseteq f(x) \cap f(y)$. Thus $|f(x) \cap f(v)| \geq \lambda$, and since the encoding is valid within T, it must be that $x \sqcup v$ exists. Then clearly

$x \sqcup v = w_i$ for some i, and $f(w_i) = f(x) \cap f(v) \supseteq f(x) \cap f(y)$, so $|f(w_i) \cap f(y)| \supseteq |f(x) \cap f(y)| \geq \lambda$. Thus, $w_i \sqcup y$ exists, so since $x \sqsubseteq w_i$, $x \sqcup y$ exists.

3. Similarly to the above, joins are preserved within T and within D_v. Let $x \in T \setminus S, y \in D_v \setminus S$. Suppose $z = x \sqcup y$, so $z \in D_v \setminus S$ (if $z \in S$, then $y \in S$). Consider a path from x to z passing through a node $w_i \in S$, and let $x \sqcup v = w_j \in S$. Since $w_j \sqsubseteq w_i \sqsubseteq z$, $w_j \sqcup y \sqsubseteq z$. However, since $w_j \sqcup y \sqsupseteq x \sqcup y = z$, we must have $w_j \sqcup y = z$. Then, since the encoding is valid within T and D_v, $f(x) \cap f(y) = f(x) \cap f(v) \cap f(y) = f(w_j) \cap f(y) = f(z)$.

   Now suppose $f(x) \cap f(y) = f(z)$. Then $x \sqcup y$ exists, otherwise we would have $|f(z)| = |f(x) \cap f(y)| \leq \lambda$. Thus $x \sqcup v$ exists, and clearly $x \sqcup v = w_i$ for some i. Then $f(z) = f(x) \cap f(y) = f(x) \cap f(v) \cap f(y) = f(w_i) \cap f(y)$. If $w_i \sqcup y = z'$, then $f(z) = f(w_i) \cap f(y) = f(z') \subseteq f(v)$, so $z \sqsupseteq v$, and thus $z \in D_v$. Then since the encoding is valid within D_v, $f(z) = f(w_i) \cap f(y)$ implies $w_i \sqcup y = z$. Thus, $x \sqcup y \sqsubseteq z$. A path from x to $x \sqcup y$ pases through $w_j \in S$. Since $w_i \sqsubseteq w_j \sqsubseteq x \sqcup y$, $z = w_i \sqcup y \sqsubseteq x \sqcup y$. Thus we must have $z = x \sqcup y$.

□