Learning Belief Functions from Data: Experimental Supplement

Václav Kratochvíl, Radim Jiroušek, Milan Daniel

prosinec 10, 2025

Contents

Introduction

This notebook provides a compact computational companion to the paper
"Learning Belief Functions from Data via Polyhedral Methods" (Kybernetika, 2025).

Its purpose is to **reproduce and document all worked examples from the paper**:

- construction of **Jeffreys-based pseudo-belief functions** g from count data,
- application of the **Upper Approximation Procedure (UAP)** to these functions (Examples 1–3, Tables 1, 4–6),
- and the **polyhedral LP approach** used to select dominating belief functions (Examples 4–5, Tables 7–9).

In particular, the notebook:

1. Shows how Jeffreys' intervals turn simple frequency tables into a normalized, monotone set function g on $\mathcal{P}(\Omega)$.
2. Applies the Upper Approximation Procedure to g and its modifications, exactly reproducing Tables 1, 4, 5, and 6.
3. Constructs the dominance polytope $\mathcal{M}_g$ and uses several linear objectives (L1, HD, CW) to obtain LP-based upper approximations, reproducing Tables 7–9.
4. For the **quaternary Example 5**, compares the LP solutions with the outcome of UAP in a single summary table.

The notebook therefore serves as an executable “log file” for all numerical tables in the paper and as a reference implementation of the UAP and LP-based upper approximation methods.

```r
# Required packages
library(lpSolve)
library(knitr)


# ===========================
# Auxiliary functions
# ===========================

# Generate all subsets
powerset <- function(Omega) {
  n <- length(Omega)
  unlist(lapply(0:n, function(i) combn(Omega, i, simplify = FALSE)), recursive = FALSE)
}

subset_name <- function(S) {
  if (length(S)==0) return("{}")
  paste0("{", paste(S, collapse=","), "}")
}
```

Jeffreys Lower Bounds

To construct the statistical lower approximation $g(A)$ on the Boolean algebra $\mathcal{P}(\Omega)$, we follow the Jeffreys–based procedure described in Section 2 of the paper.

Assume that observations are recorded on subsets $B \subseteq \Omega$, possibly ambiguous (e.g. $\{w_1, w_2\}$). Let n_B denote the number of observations associated with B, and let

$$N = \sum_{B \subseteq \Omega} n_B$$

be the total number of observed cases.

An observation assigned to B supports the event A only when $B \subseteq A$. The effective number of “successes’ ’ for event A is therefore

$$s(A) = \sum_{B \subseteq A} n_B.$$

Under Jeffreys’ prior Beta$(1/2, 1/2)$, the posterior distribution of the binomial parameter $p(A)$ is

$$p(A) \mid \text{data} \sim \text{Beta}\big(s(A) + \tfrac{1}{2},\ N - s(A) + \tfrac{1}{2}\big)\,,$$

and the corresponding lower $(1-\alpha)$-credible bound is

$$g(A) = L(A) = \text{qbeta}\big(\tfrac{\alpha}{2},\ s(A) + \tfrac{1}{2},\ N - s(A) + \tfrac{1}{2}\big)\,.$$

Following the article, we additionally impose the normalization condition $g(\Omega) = 1$, which ensures that the resulting set function behaves as a *pseudo-belief function* compatible with the dominance relation used later in the Upper Approximation Procedure.

All computations of $g(A)$ in this notebook use the function `jeffreys_pseudo_belief_function()` defined below, which implements the mapping $\{n_B\} \mapsto g(\cdot)$ exactly as described above.

For convenience, we include the empty set $\emptyset$ in the internal representation (with $g(\emptyset) = 0$, but we omit it from the printed tables.

```r
# ------------------------------------------------------------
# Jeffreys confidence interval for the binomial model
# Returns the lower and upper (1 - alpha) credible bounds.
# ------------------------------------------------------------
jeffreys_interval <- function(successes, total, alpha = 0.05,
                              a0 = 0.5, b0 = 0.5) {

  lower <- qbeta(alpha / 2, successes + a0, total - successes + b0)
  upper <- qbeta(1 - alpha / 2, successes + a0, total - successes + b0)

  list(lower = lower, upper = upper)
}



# ------------------------------------------------------------
# Normalize a named vector of counts:
# Ensures all subsets exist; missing subsets are assigned zero.
# ------------------------------------------------------------
normalize_counts <- function(counts_named, subsets) {

  subset_names <- sapply(subsets, subset_name)
  out <- setNames(rep(0, length(subsets)), subset_names)

  if (is.null(names(counts_named)))
    stop("counts_named must be a named vector, e.g. c('{w1}'=5, '{w1,w2}'=2).")

  # fill matching names
  out[names(counts_named)] <- as.numeric(counts_named)

  out
}



# ------------------------------------------------------------
# Construct Jeffreys-based pseudo-belief function g(A):
#   g(A) = Jeffreys lower bound for f(A)
#   where "successes" for A is the total number of observations
#   assigned to subsets B \subseteq A.
#
# The function automatically enforces g(Omega) = 1, consistently with the paper
# ------------------------------------------------------------
jeffreys_pseudo_belief_function <- function(Omega, counts_named, alpha = 0.05) {
  # all subsets in cardinality order
  subsets <- powerset(Omega)
  subset_names <- sapply(subsets, subset_name)

  # full count vector for all subsets
  counts_full <- normalize_counts(counts_named, subsets)
  N <- sum(counts_full)

  # compute "successes" s(A) = sum_{B \subseteq A} n_B
  successes <- sapply(subsets, function(A) {
    idx <- sapply(subsets, function(B) all(B %in% A))
    sum(counts_full[idx])
  })

  # allocate vector for g(A)
  g_values <- numeric(length(subsets))

  for (i in seq_along(subsets)) {
    A <- subsets[[i]]

    # by convention g(emptyset) = 0
    if (length(A) == 0) {
      g_values[i] <- 0
    } else {
      ji <- jeffreys_interval(successes[i], N, alpha = alpha)
      g_values[i] <- ji$lower
    }
  }

  # normalize: g(Omega) must equal 1
  g_values[length(g_values)] <- 1

  list(
    g            = g_values,
    subsets      = subsets,
    subset_names = subset_names,
    counts       = counts_full,
    N            = N
  )
}
```

Example: Ambiguous observations

This example illustrates how the Jeffreys-based construction processes **partially ambiguous measurements**.

We consider 30 observations in total:

- 25 observations are fully specified (assigned to the singletons $\{w_1\}, \{w_2\}, \{w_3\}$),
- 3 observations cannot distinguish between states w_1 and w_2 and are therefore recorded as the set $\{w_1, w_2\}$,
- 2 observations provide no discriminatory information at all, represented by the full set $\{w_1, w_2, w_3\}$.

In this setting, an ambiguous observation recorded as B contributes evidence to every event A such that $B \subseteq A$.
Consequently, the resulting Jeffreys lower bounds $g(A)$ naturally incorporate both precise and imprecise

information without forcing an artificial resolution of uncertainty.

```r
## Frame of discernment
Omega <- c("w1", "w2", "w3")

## Example: 30 observations (25 precise, 3 ambiguous {w1,w2}, 2 fully ambiguous)
counts <- c(
  "{w1}"        = 10,
  "{w2}"        = 8,
  "{w3}"        = 7,
  "{w1,w2}"     = 3,
  "{w1,w2,w3}"  = 2
)

## Compute Jeffreys pseudo-belief function g(A)
g_result <- jeffreys_pseudo_belief_function(Omega, counts, alpha = 0.05)

## Create output table (drop empty set)
df <- data.frame(
  subset = g_result$subset_names,
  g      = round(g_result$g, 3),
  row.names = NULL
)

df <- df[df$subset != "{}", ]   # hide the empty set

df
```

```
##        subset     g
## 2        {w1} 0.186
## 3        {w2} 0.135
## 4        {w3} 0.111
## 5     {w1,w2} 0.523
## 6     {w1,w3} 0.390
## 7     {w2,w3} 0.328
## 8 {w1,w2,w3} 1.000
```

Upper Approximation Procedure (UAP)

To obtain a belief function f that dominates the statistically derived pseudo-belief function g, the paper employs the **Upper Approximation Procedure** (UAP) introduced in Section 3.
The method constructs a mass assignment m and the corresponding belief function f by processing the subsets of Ω in increasing order of cardinality.

For every set A, the value of $f(A)$ is determined so that

$$f(A) \geq g(A) \quad \text{and} \quad f(A) \geq \sum_{B \subset A} m(B),$$

where the second inequality ensures the internal consistency of the belief function. The increment

$$m(A) = f(A) - \sum_{B \subset A} m(B)$$

defines the basic probability assigned to A.

If the procedure reaches the top element Ω and produces $f(\Omega) > 1$, the construction fails; otherwise the output is a valid belief function dominating g.
This behaviour matches the examples in the article, including the failure for Table 4 and the successful runs for Table 1 (modified), Table 5, and Table 6.

All computations below use the implementation `upper_approximation()`, which follows the pseudocode of the article exactly.The procedure processes subsets in increasing cardinality, ensuring the Möbius recursion is respected.

```r
# ------------------------------------------------------------
# Upper Approximation Procedure (UAP)
#
# Input:
#   Omega     ... vector of frame elements
#   g_values  ... numeric vector of g(A) in cardinality order,
#                 or named by subset_name().
#
# Output:
#   list(success, subsets, f, m)
#
# The procedure assumes that g(Omega) = 1 (enforced here).
# ------------------------------------------------------------
upper_approximation <- function(Omega, g_values) {

  # 1) Generate subsets in canonical order (cardinality-first)
  subsets <- powerset(Omega)
  subsets <- subsets[order(sapply(subsets, length))]
  subset_names <- sapply(subsets, subset_name)

  # 2) If g_values have names -> reorder; if not -> assume correct order
  if (!is.null(names(g_values))) {
    g_values <- g_values[subset_names]
  } else {
    names(g_values) <- subset_names
  }


  # 3) Enforce normalization: g(Omega) = 1
  # Find index of \Omega = full set
  full_idx <- which(subset_names == subset_name(Omega))
  g_values[full_idx] <- 1

  k <- length(subsets)
  f <- numeric(k)
  m <- numeric(k)

  # 4) Main UAP loop (exact definition from article)
  for (i in seq_len(k)) {
    A <- subsets[[i]]

    # proper subsets of A
    idx <- sapply(subsets, function(B)
      length(B) < length(A) && all(B %in% A))

    sum_prev <- sum(m[idx])
    # f(A)
    f[i] <- max(g_values[i], sum_prev)

    # m(A)
    m[i] <- f[i] - sum_prev
  }

  # 5) Check failure condition
  if (f[k] > 1 + 1e-12) {
    return(list(
      success = FALSE,
      message = "UAP failed: f(Omega) > 1",
      subsets = subset_names,
      f = f,
      m = m
    ))
  }

  # IMPORTANT:
  # f(Omega) is NOT overwritten.
  # m(Omega) stays as computed.

  list(
    success = TRUE,
    subsets = subset_names,
    f = f,
    m = m
  )
}


# ============================================
# Helper: Print UAP table exactly like in paper
# ============================================

uap_table <- function(result, g_values) {
  df <- data.frame(
    A        = result$subsets,
    g        = g_values,
    sum_prev = round(result$f - result$m, 6),
    f        = round(result$f, 6),
    m_f      = round(result$m, 6),
    row.names = NULL
  )

  # remove the first row (empty set)
  df <- df[-1, ]

  df
}
```

Table 1 (ternary, original g)

```r
Omega3 <- c("w1","w2","w3")

g_ternary <- c(
  "{}"=0,
  "{w1}"=0.1, "{w2}"=0.1, "{w3}"=0.2,
  "{w1,w2}"=0.5, "{w1,w3}"=0.4, "{w2,w3}"=0.7,
  "{w1,w2,w3}"=1.0
)

res <- upper_approximation(Omega3, g_ternary)
res$success  # expected: FALSE (as in Table 1)
```

```
## [1] FALSE
```

```r
table <- uap_table(res, g_ternary)

# print table
knitr::kable(
  table,
  caption = "Table 1 - Upper Approximation for the original pseudo-belief function g on a ternary frame
  align = "lrrrr",
  row.names = FALSE
)
```

Table 1: Table 1 — Upper Approximation for the original pseudo-belief function g on a ternary frame.

A	g	sum_prev	f	m_f
{w1}	0.1	0.0	0.1	0.1
{w2}	0.1	0.0	0.1	0.1
{w3}	0.2	0.0	0.2	0.2
{w1,w2}	0.5	0.2	0.5	0.3
{w1,w3}	0.4	0.3	0.4	0.1
{w2,w3}	0.7	0.3	0.7	0.4
{w1,w2,w3}	1.0	1.2	1.2	0.0

For the original pseudo-belief function g, the UAP fails because the lower bounds on several subsets jointly force the final value $f(\Omega)$ to exceed 1.

Table 1 (modified $\tilde{g}$)

```r
g_ternary_tilde <- c(
  "{}"=0,
  "{w1}"=0.2, "{w2}"=0.2, "{w3}"=0.3,
  "{w1,w2}"=0.5, "{w1,w3}"=0.4, "{w2,w3}"=0.7,
  "{w1,w2,w3}"=1.0
)

res_tilde <- upper_approximation(Omega3, g_ternary_tilde)
res_tilde$success  # expected: TRUE
```

```
## [1] TRUE
```

```
table <- uap_table(res_tilde, g_ternary_tilde)

# print table
knitr::kable(
  table,
  caption = "Table 1 (modified) - UAP succeeds for the adjusted pseudo-belief.",
  align = "lrrrr",
  row.names = FALSE
)
```

Table 2: Table 1 (modified) — UAP succeeds for the adjusted pseudo-belief.

A	g	sum_prev	f	m_f
{w1}	0.2	0.0	0.2	0.2
{w2}	0.2	0.0	0.2	0.2
{w3}	0.3	0.0	0.3	0.3
{w1,w2}	0.5	0.4	0.5	0.1
{w1,w3}	0.4	0.5	0.5	0.0
{w2,w3}	0.7	0.5	0.7	0.2
{w1,w2,w3}	1.0	1.0	1.0	0.0

The modified function $\tilde{g}$, which only increases singleton values, removes this conflict: the accumulated mass from proper subsets stays below 1, allowing the UAP to construct a valid dominating belief function. The example illustrates that the feasibility of UAP is strongly dependent on the internal coherence of the statistical lower bounds.

UAP Reproduction of Tables 4, 5, 6

In the quaternary case $|\Omega| = 4$, the data originate from a simple frequency table (Table 2 in the article). Applying the Jeffreys construction yields a pseudo-belief function g whose properties depend sensitively on the confidence level α.
This section reproduces the behaviour of the Upper Approximation Procedure under three different conditions corresponding to Tables 4, 5, and 6.

```
Omega4 <- c("w1","w2","w3","w4")

counts_raw <- c(
  "{w1}"=14, "{w2}"=17, "{w3}"=15, "{w4}"=6
)
```

Table 4 — Jeffreys lower bounds with $\alpha = 0.05$

For the standard 95% Jeffreys interval, many lower bounds become relatively tight, especially on two– and three–element subsets.
When fed into the UAP, the cumulative mass assigned to proper subsets grows too large, and the procedure attempts to set

$$f(\Omega) > 1,$$

which violates the normalization constraint.
Consequently, **UAP fails** for this choice of α, matching the result reported in Table 4 of the article.

This failure illustrates that the lower statistical bounds may be too optimistic for a belief function to dominate them.

```r
# Compute pseudo-belief function g(A)
g4_res  <- jeffreys_pseudo_belief_function(Omega4, counts_raw, alpha = 0.05)
g4_vals <- g4_res$g
names(g4_vals) <- g4_res$subset_names

# Apply UAP
res4 <- upper_approximation(Omega4, g4_vals)

# Expected: FALSE (matches article Table 4)
res4$success
```

```
## [1] FALSE
```

```r
# Construct table
tab4 <- uap_table(res4, g4_vals)

# print table
knitr::kable(
  tab4,
  caption = "Table 4 - UAP failure for Jeffreys lower bounds with alpha = 0.05.",
  align = "lrrrr",
  row.names = FALSE
)
```

Table 3: Table 4 — UAP failure for Jeffreys lower bounds with alpha = 0.05.

A	g	sum_prev	f	m_f
{w1}	0.1634559	0.000000	0.163456	0.163456
{w2}	0.2114491	0.000000	0.211449	0.211449
{w3}	0.1792032	0.000000	0.179203	0.179203
{w4}	0.0496248	0.000000	0.049625	0.049625
{w1,w2}	0.4605750	0.374905	0.460575	0.085670
{w1,w3}	0.4226370	0.342659	0.422637	0.079978
{w1,w4}	0.2615272	0.213081	0.261527	0.048447
{w2,w3}	0.4798459	0.390652	0.479846	0.089194
{w2,w4}	0.3134786	0.261074	0.313479	0.052405
{w3,w4}	0.2786434	0.228828	0.278643	0.049815
{w1,w2,w3}	0.7775591	0.808950	0.808950	0.000000
{w1,w2,w4}	0.5794675	0.611051	0.611051	0.000000
{w1,w3,w4}	0.5389325	0.570524	0.570524	0.000000
{w2,w3,w4}	0.6001125	0.631691	0.631691	0.000000
{w1,w2,w3,w4}	1.0000000	1.009241	1.009241	0.000000

Table 5 — Jeffreys lower bounds with $\alpha = 0.02$

Reducing the confidence level to $\alpha = 0.02$ produces wider Jeffreys intervals and therefore **smaller lower bounds** $g(A)$.
These relaxed bounds reduce the dominance pressure on the UAP recursion, allowing it to satisfy all inequalities while keeping

$$f(\Omega) = 1.$$

The UAP therefore **succeeds**, giving the belief function listed in Table 5. This demonstrates how the statistical confidence level directly influences the feasibility of constructing a dominating belief function.

```r
## Table 5 - Jeffreys lower bounds (alpha = 0.02)

# Compute pseudo-belief function g(A)
g5_res  <- jeffreys_pseudo_belief_function(Omega4, counts_raw, alpha = 0.02)
g5_vals <- g5_res$g
names(g5_vals) <- g5_res$subset_names

# Apply UAP
res5 <- upper_approximation(Omega4, g5_vals)

# Expected: TRUE (matches article Table 5)
res5$success
```

```
## [1] TRUE
```

```r
# Construct table
tab5 <- uap_table(res5, g5_vals)

# Print table

# Knit-friendly table
knitr::kable(
  tab5,
  caption = "Table 5 - UAP succeeds for Jeffreys lower bounds with alpha = 0.02.",
  align = "lrrrr",
  row.names = FALSE
)
```

Table 4: Table 5 — UAP succeeds for Jeffreys lower bounds with alpha = 0.02.

A	g	sum_prev	f	m_f
{w1}	0.1465951	0.000000	0.146595	0.146595
{w2}	0.1923940	0.000000	0.192394	0.192394
{w3}	0.1615697	0.000000	0.161570	0.161570
{w4}	0.0408735	0.000000	0.040873	0.040873
{w1,w2}	0.4355082	0.338989	0.435508	0.096519
{w1,w3}	0.3980304	0.308165	0.398030	0.089866
{w1,w4}	0.2406251	0.187469	0.240625	0.053157
{w2,w3}	0.4546004	0.353964	0.454600	0.100637
{w2,w4}	0.2910446	0.233267	0.291045	0.057777
{w3,w4}	0.2571967	0.202443	0.257197	0.054754
{w1,w2,w3}	0.7545245	0.787580	0.787580	0.000000
{w1,w2,w4}	0.5538838	0.587315	0.587315	0.000000
{w1,w3,w4}	0.5133675	0.546814	0.546814	0.000000
{w2,w3,w4}	0.5745835	0.608005	0.608005	0.000000
{w1,w2,w3,w4}	1.0000000	0.994141	1.000000	0.005859

Table 6 — Discounted pseudo-belief function ($\delta = 0.02$)

Another way to restore feasibility is **Shafer's discounting**.
Applying a discount factor $\delta = 0.02$ multiplies every proper subset by $(1-\delta)$ while keeping $g(\Omega) = 1$.
This uniform reduction weakens the dominance requirements across all levels of the lattice:

$$\hat{g}(A) = (1-\delta)\, g(A), \qquad A \neq \Omega.$$

The discounted function $\hat{g}$ becomes compatible with a valid belief function, and the UAP again **succeeds**, reproducing Table 6 of the article.

These three examples highlight that the UAP is highly sensitive to the shape of the pseudo-belief function, and even minimal adjustments—either statistical (α) or structural (δ)—may determine whether a feasible upper approximation exists.

```
## Table 6 - Discounting delta = 0.02 applied to Jeffreys g from Table 4

delta <- 0.02

# Start from g of Table 4 (alpha = 0.05)
g4_vals <- g4_res$g
names(g4_vals) <- g4_res$subset_names

# Apply Shafer discounting to all proper subsets
g6_vals <- g4_vals
is_top <- names(g6_vals) == "{w1,w2,w3,w4}"

g6_vals[!is_top] <- (1 - delta) * g6_vals[!is_top]
g6_vals[is_top]  <- 1   # enforce g(Omega) = 1

# Run UAP
res6 <- upper_approximation(Omega4, g6_vals)

# Expected: TRUE
res6$success
```

```
## [1] TRUE
```

```
# Construct table
tab6 <- uap_table(res6, g6_vals)

# print table
knitr::kable(
  tab6,
  caption = "Table 6 - UAP succeeds after Shafer discounting with delta = 0.02.",
  align = "lrrrr",
  row.names = FALSE
)
```

Table 5: Table 6 — UAP succeeds after Shafer discounting with delta = 0.02.

A	g	sum_prev	f	m_f
{w1}	0.1601868	0.000000	0.160187	0.160187
{w2}	0.2072201	0.000000	0.207220	0.207220
{w3}	0.1756192	0.000000	0.175619	0.175619

A	g	sum_prev	f	m_f
{w4}	0.0486323	0.000000	0.048632	0.048632
{w1,w2}	0.4513635	0.367407	0.451363	0.083957
{w1,w3}	0.4141842	0.335806	0.414184	0.078378
{w1,w4}	0.2562967	0.208819	0.256297	0.047478
{w2,w3}	0.4702490	0.382839	0.470249	0.087410
{w2,w4}	0.3072090	0.255852	0.307209	0.051357
{w3,w4}	0.2730705	0.224251	0.273071	0.048819
{w1,w2,w3}	0.7620079	0.792771	0.792771	0.000000
{w1,w2,w4}	0.5678782	0.598830	0.598830	0.000000
{w1,w3,w4}	0.5281538	0.559113	0.559113	0.000000
{w2,w3,w4}	0.5881103	0.619057	0.619057	0.000000
{w1,w2,w3,w4}	1.0000000	0.989056	1.000000	0.010944

Polyhedral Representation of the Upper Approximation

The Upper Approximation of pseudo-belief function g defined in the article is the set of all basic probability assignments m whose induced belief function f dominates the pseudo-belief function g.
For every subset $A \subseteq \Omega$,

$$f(A) = \sum_{B \subseteq A} m(B) \qquad \text{and} \qquad f(A) \geq g(A).$$

Together with the standard belief-function constraints

$$m(\emptyset) = 0, \qquad m(A) \geq 0, \qquad \sum_{A} m(A) = 1,$$

the feasible region becomes a **polytope** in $\mathbb{R}^{2^{|\Omega|}}$, described entirely by linear inequalities:

$$\mathcal{M}_g = \big\{ m \mid \mathbf{M}_{\text{bel}}\, m \geq g,\ m \geq 0,\ m(\emptyset) = 0,\ \sum_{A} m(A) = 1 \big\}.$$

Here, $\mathbf{M}_{\text{bel}}$ is the *incidence matrix* of the subset lattice: its entry is 1 exactly when $B \subseteq A$.
This matrix linearly maps a BPA m to the corresponding belief function f.

The polyhedral formulation serves two purposes:

1. it provides a global geometric interpretation of the dominance condition $f \geq g$;
2. it allows the use of dedicated tools (such as **cddlib**) to enumerate the **vertices** of $\mathcal{M}_g$ and analyse its structure.

In particular, the ternary example in the article yields **28** extreme points,
while the quaternary example expands to **13 889** vertices,
illustrating the rapid combinatorial growth of the polytope as the dimension increases.

Constructing the Polytope in R

The following function constructs this polyhedron directly in terms of its linear inequality and equality constraints. This representation is *unified*: it is suitable both for **vertex enumeration** (`rcdd`) and for **linear programming** (LP-based approximations).

```r
### Constructing the polytope M_g using a unified LP representation
build_lp_constraints <- function(Omega, g_values) {

  subsets       <- powerset(Omega)
  subset_names  <- sapply(subsets, subset_name)
  n_vars        <- length(subsets)

  stopifnot(length(g_values) == n_vars)

  # -------------------------------
  # Dominance constraints: M_bel m >= g
  # -------------------------------
  M_dom <- matrix(0, nrow=n_vars, ncol=n_vars)
  for (i in seq_along(subsets)) {
    A <- subsets[[i]]
    M_dom[i, sapply(subsets, function(B) all(B %in% A))] <- 1
  }
  b_dom <- g_values

  # -------------------------------
  # Nonnegativity: m >= 0
  # -------------------------------
  M_nonneg <- diag(n_vars)
  b_nonneg <- rep(0, n_vars)

  # Stack all inequalities
  M_ineq <- rbind(M_dom, M_nonneg)
  b_ineq <- c(b_dom, b_nonneg)

  # -------------------------------
  # Equalities:
  #   sum m(A) = 1
  #   m(emptyset)   = 0
  # -------------------------------
  M_eq <- matrix(0, nrow=2, ncol=n_vars)
  M_eq[1, ] <- 1          # normalization
  M_eq[2, 1] <- 1         # empty set mass = 0
  b_eq <- c(1, 0)

  colnames(M_ineq) <- subset_names
  colnames(M_eq)   <- subset_names

  list(
    subsets = subsets,
    subset_names = subset_names,
    M_ineq = M_ineq,
    b_ineq = b_ineq,
    M_eq   = M_eq,
    b_eq   = b_eq
  )
}

# Example build
```

```r
Omega <- c("w1","w2","w3")
g_example <- c(0, 0.1,0.2,0.3, 0.5,0.6,0.8, 1.0)

lp_data <- build_lp_constraints(Omega, g_example)
lp_data[3:6]
```

```
## $M_ineq
##       {} {w1} {w2} {w3} {w1,w2} {w1,w3} {w2,w3} {w1,w2,w3}
##  [1,]  1    0    0    0       0       0       0          0
##  [2,]  1    1    0    0       0       0       0          0
##  [3,]  1    0    1    0       0       0       0          0
##  [4,]  1    0    0    1       0       0       0          0
##  [5,]  1    1    1    0       1       0       0          0
##  [6,]  1    1    0    1       0       1       0          0
##  [7,]  1    0    1    1       0       0       1          0
##  [8,]  1    1    1    1       1       1       1          1
##  [9,]  1    0    0    0       0       0       0          0
## [10,]  0    1    0    0       0       0       0          0
## [11,]  0    0    1    0       0       0       0          0
## [12,]  0    0    0    1       0       0       0          0
## [13,]  0    0    0    0       1       0       0          0
## [14,]  0    0    0    0       0       1       0          0
## [15,]  0    0    0    0       0       0       1          0
## [16,]  0    0    0    0       0       0       0          1
##
## $b_ineq
##  [1] 0.0 0.1 0.2 0.3 0.5 0.6 0.8 1.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
##
## $M_eq
##      {} {w1} {w2} {w3} {w1,w2} {w1,w3} {w2,w3} {w1,w2,w3}
## [1,]  1    1    1    1       1       1       1          1
## [2,]  1    0    0    0       0       0       0          0
##
## $b_eq
## [1] 1 0
```

Vertex Enumeration of $\mathcal{M}_g$

The unified LP description of $\mathcal{M}_g$ can be converted into the H-representation required by `rcdd`.
Each vertex of this polytope corresponds to an *extreme* belief function that still dominates the pseudo-belief function g.

The number of vertices grows rapidly, as illustrated on polytopes related to above mentioned examples on $|\Omega| = 3, 4$. We also list first 10 vertices for each polytope.

This illustrates the exponential geometric complexity of the dominance constraints.

```r
library(rcdd)

# Convert LP constraints to rcdd H-representation
lp_to_H <- function(lp) {
  H_ineq <- cbind(0, -lp$b_ineq, lp$M_ineq)  # inequalities: 0 + Mx - b >= 0
  H_eq   <- cbind(1, -lp$b_eq,   lp$M_eq)    # equalities:  type=1
  d2q(rbind(H_ineq, H_eq))
}
```

```r
# Extract vertices returned by rcdd
extract_vertices <- function(Vout) {
  out <- Vout$output
  V   <- out[out[,1]=="0", , drop=FALSE]    # rows marked as vertices
  if (nrow(V) == 0) return(NULL)
  q2d(V[,-c(1,2), drop=FALSE])              # drop type and constant
}

# Wrapper for computing vertices directly from g-values
enumerate_vertices <- function(Omega, g_values) {
  lp <- build_lp_constraints(Omega, g_values)
  H  <- lp_to_H(lp)
  V  <- scdd(H)
  verts <- extract_vertices(V)
  colnames(verts) <- names(g_values)
  list(
    subsets = lp$subset_names,
    vertices = verts,
    n_vertices = if (is.null(verts)) 0L else nrow(verts)
  )
}

# --- Ternary example ---
res <- enumerate_vertices(Omega3, g_ternary)
res$n_vertices  # expected: 28
```

```
## [1] 28
```

```r
res$vertices[1:10, ]
```

```
##       {} {w1} {w2} {w3}      {w1,w2} {w1,w3} {w2,w3} {w1,w2,w3}
##  [1,]  0  0.1  0.4  0.3 5.551115e-17     0.0     0.0        0.2
##  [2,]  0  0.1  0.2  0.5 2.000000e-01     0.0     0.0        0.0
##  [3,]  0  0.2  0.5  0.2 1.000000e-01     0.0     0.0        0.0
##  [4,]  0  0.1  0.4  0.3 2.000000e-01     0.0     0.0        0.0
##  [5,]  0  0.2  0.2  0.2 1.000000e-01     0.0     0.3        0.0
##  [6,]  0  0.1  0.2  0.3 2.000000e-01     0.0     0.2        0.0
##  [7,]  0  0.1  0.3  0.2 1.000000e-01     0.1     0.2        0.0
##  [8,]  0  0.1  0.5  0.2 1.000000e-01     0.1     0.0        0.0
##  [9,]  0  0.1  0.4  0.2 0.000000e+00     0.1     0.1        0.1
## [10,]  0  0.1  0.4  0.2 0.000000e+00     0.2     0.1        0.0
```

```r
# --- Quaternary example ---
res4 <- enumerate_vertices(Omega4, g5_vals)
res4$n_vertices  # expected: 13889
```

```
## [1] 13889
```

```r
res4$vertices[1:10, ]
```

```
##       {}      {w1}      {w2}      {w3}       {w4}    {w1,w2} {w1,w3} {w1,w4}
##  [1,]  0 0.1997516 0.2501711 0.2415141 0.04087347 0.06308764       0       0
##  [2,]  0 0.1997516 0.1923940 0.2992912 0.04087347 0.06308764       0       0
##  [3,]  0 0.1465951 0.1923940 0.3190163 0.11837567 0.09651909       0       0
##  [4,]  0 0.1465951 0.1923940 0.3190163 0.24547552 0.09651909       0       0
##  [5,]  0 0.1465951 0.1923940 0.3190163 0.11837567 0.09651909       0       0
##  [6,]  0 0.1465951 0.1923940 0.3190163 0.09402999 0.09651909       0       0
##  [7,]  0 0.1465951 0.1923940 0.3190163 0.09402999 0.09651909       0       0
##  [8,]  0 0.1465951 0.1923940 0.2727424 0.09402999 0.14279295       0       0
##  [9,]  0 0.1465951 0.1923940 0.2727424 0.09402999 0.09651909       0       0
## [10,]  0 0.1465951 0.1923940 0.2727424 0.09402999 0.09651909       0       0
##       {w2,w3}    {w2,w4}   {w3,w4} {w1,w2,w3} {w1,w2,w4} {w1,w3,w4} {w2,w3,w4}
##  [1,]       0 0.00000000 0.0000000 0.00000000          0 0.03122834  0.1733737
##  [2,]       0 0.05777713 0.0000000 0.00000000          0 0.14682492  0.0000000
##  [3,]       0 0.00000000 0.0000000 0.00000000          0 0.00000000  0.1270999
##  [4,]       0 0.00000000 0.0000000 0.00000000          0 0.00000000  0.0000000
##  [5,]       0 0.00000000 0.1270999 0.00000000          0 0.00000000  0.0000000
##  [6,]       0 0.02434568 0.0000000 0.00000000          0 0.00000000  0.1270999
##  [7,]       0 0.15144554 0.0000000 0.00000000          0 0.00000000  0.0000000
##  [8,]       0 0.15144554 0.0000000 0.00000000          0 0.00000000  0.0000000
##  [9,]       0 0.02434568 0.0000000 0.04627386          0 0.00000000  0.1270999
## [10,]       0 0.15144554 0.0000000 0.04627386          0 0.00000000  0.0000000
##       {w1,w2,w3,w4}
##  [1,]             0
##  [2,]             0
##  [3,]             0
##  [4,]             0
##  [5,]             0
##  [6,]             0
##  [7,]             0
##  [8,]             0
##  [9,]             0
## [10,]             0
```

Linear Programming Approximations

The polytope $\mathcal{M}_g$ contains all belief functions f dominating the pseudo-belief function g.
To select a *single representative* belief function, we minimise a linear objective

$$\min_{m \in \mathcal{M}_g} c^\top m,$$

where the vector c encodes a preference for certain focal sets.

We now implement three criteria used in the article:

- **L1** (minimal upward correction)

- **HD** (negative Dubois–Prade entropy; LP minimisation = entropy maximisation)
- **CW** (cardinality-weighted)

The following unified solver builds the LP constraints and computes the optimal m and induced belief function f for any objective.

```
objective_L1 <- function(subsets, Omega) {
  sapply(subsets, function(B) 2^(length(Omega) - length(B)))
}

objective_HD <- function(subsets, Omega) {
  - sapply(subsets, function(B) if (length(B) == 0) 0 else log(length(B), base = 2))
}
```

```r
objective_CW <- function(subsets, Omega) {
  sapply(subsets, function(B) if (length(B) == 0) 0 else 1/length(B))
}
```

Call the LP solver

```r
# ===================================================================
# solve_lp_for_objective()
# ------------------------------------------------------------------
# Solves the linear program defining the upper approximation f >= g
# for a chosen linear objective function in the belief space.
#
# The LP is formulated over the basic probability assignment vector m,
# subject to the standard belief-function constraints:
#
#   (1)  M_bel * m   >= g    (dominance: f(A) >= g(A))
#   (2)  m(A)        >= 0    (non-negativity)
#   (3)  m(emptyset)        =  0    (empty set has zero mass)
#   (4)  sum_A m(A)  =  1    (normalization)
#
# The objective is always of the form:
#          minimize   sum_A  w(A) * m(A)
# where w(A) is supplied by obj_fun (L1, HD, CW, ...).
#
# Input:
#   Omega     ... frame of discernment
#   g_values  ... pseudo-belief function g(A)
#   obj_fun   ... objective-generating function, e.g. objective_L1
#
# Output:
#   A list with:
#       subset ... subset names in lexicographic order
#       f      ... resulting belief function f(A)
#       m      ... resulting BPA m(A)
#
# ===================================================================
solve_lp_for_objective <- function(Omega, g_values, obj_fun) {

  # Construct all LP constraint matrices for f >= g
  lp_data <- build_lp_constraints(Omega, g_values)

  subsets      <- lp_data$subsets
  subset_names <- lp_data$subset_names
  nA <- length(subsets)

  # -------------------------------------------------------------
  # Objective vector: w(A) for each subset A
  # (determined by objective_L1, objective_HD, objective_CW, ...)
  # -------------------------------------------------------------
  obj <- obj_fun(subsets, Omega)

  # -------------------------------------------------------------
  # Full LP system:
```

```
  #   [M_ineq]  m   >= b_ineq
  #   [M_eq  ]  m   == b_eq
  # ------------------------------------------------------------
  M_full <- rbind(lp_data$M_ineq, lp_data$M_eq)
  dir_full <- c(rep(">=", nrow(lp_data$M_ineq)), rep("=", nrow(lp_data$M_eq)))
  rhs_full <- c(lp_data$b_ineq, lp_data$b_eq)

  # ------------------------------------------------------------
  # Solve LP using lpSolve
  # ------------------------------------------------------------
  sol <- lp(
    direction    = "min",
    objective.in = obj,
    const.mat    = M_full,
    const.dir    = dir_full,
    const.rhs    = rhs_full
  )

  # Extract BPA m(A)
  m <- sol$solution
  names(m) <- subset_names

  # Reconstruct belief f(A) = sum_{B \subseteq A} m(B)
  f <- sapply(subsets, function(A) {
    idx <- sapply(subsets, function(B) all(B %in% A))
    sum(m[idx])
  })

  list(
    subset = subset_names,
    f = f,
    m = m
  )
}
```

LP Reproduction of Tables 7, 8, 9

The following code block reproduces Tables 7, 8, and 9 from the paper by:

1. constructing the polytope $\mathcal{M}_g$,
2. solving the LP for each objective,
3. computing the corresponding belief and mass assignments,
4. presenting all results in a unified table.

As in the article, we omit the empty set from printed tables.

```
# ================================================================
# compute_table()
# ---------------------------------------------------------------
# Computes all mass and belief values for several approximation
# methods (L1, CW, HD, UAP) and returns a unified table.
#
# Input:
#   Omega     - vector of states ("w1","w2",...)
#   g_values  - pseudo-belief function g(A), already ordered and normalized
```

```r
#
# Output:
#   A list with:
#     $table   - data frame containing g(A), f(A), and m(A) for all methods
#     $subsets - list of subsets of Omega (used by criteria function)
#
# This function does NOT compute criterion values; that is delegated
# to compute_criteria(), keeping responsibilities cleanly separated.
# ================================================================
compute_table <- function(Omega, g_values) {

  lp_data <- build_lp_constraints(Omega, g_values)
  subsets <- lp_data$subsets
  subset_names <- lp_data$subset_names
  idx <- 2:length(subset_names)   # remove empty set

  # ---- Solve all LP variants ----
  sol_L1  <- solve_lp_for_objective(Omega, g_values, objective_L1)
  sol_HD  <- solve_lp_for_objective(Omega, g_values, objective_HD)
  sol_CW  <- solve_lp_for_objective(Omega, g_values, objective_CW)
  uap     <- upper_approximation(Omega, g_values)

  # ---- Build main table (body) ----
  # Each row corresponds to a subset A in lexicographic order.
  # Columns contain:
  #   g(A)       - pseudo-belief
  #   f_method   - belief produced by each approximation method
  #   m_method   - corresponding mass assignments
  table_body <- data.frame(
    subset = subset_names,
    g      = g_values,

    f_L1   = sol_L1$f,
    f_HD   = sol_HD$f,
    f_CW   = sol_CW$f,
    f_UAP  = uap$f,

    m_L1   = sol_L1$m,
    m_HD   = sol_HD$m,
    m_CW   = sol_CW$m,
    m_UAP  = uap$m,

    row.names = NULL
  )

  list(
    table_body = table_body,
    subsets = subsets
  )
}

# ================================================================
# evaluate_lp_criteria()
# ----------------------------------------------------------------
# Computes values of each criterion (L1, CW, HD) for each mass
# vector in the table produced by compute_table_explicit().
#
# Input:
#   table_body - data frame containing m_* columns
#   subsets    - list of subsets (same order as masses)
#   Omega      - frame of discernment
#
# Output:
#   A matrix with rows = criteria (L1, CW, HD)
#                 columns = mass vectors (m_L1, m_CW, m_HD, m_rSP, m_UAP ...)
#
# The formula is always:
#     criterion_value = sum_A  w(A) * m(A)
# where w(A) is given by objective_* functions.
# ================================================================
evaluate_lp_criteria <- function(table_body, subsets, Omega) {

  # Find all mass columns automatically (m_L1, m_CW, m_HD, m_rSP, m_UAP, ...)
  m_cols <- grep("^m_", names(table_body), value = TRUE)

  # Available criteria (objective functions defined earlier)
  crit_map <- list(
    L1 = objective_L1,
    CW = objective_CW,
    HD = function(subsets, Omega) {-objective_HD(subsets, Omega)}
  )

  # Initialize (criteria × mass vectors)
  crit_matrix <- matrix(
    NA,
    nrow = length(crit_map),
    ncol = length(m_cols),
    dimnames = list(names(crit_map), m_cols)
  )

  # Calculate weighted sum for each criterion-method pair
  for (crit_name in names(crit_map)) {
    w <- crit_map[[crit_name]](subsets, Omega)  # objective weights
    for (m_col in m_cols) {
      m_vec <- table_body[[m_col]]
      crit_matrix[crit_name, m_col] <- sum(w * m_vec)
    }
  }

  crit_matrix
}
```

Table 7

```r
table <- compute_table(g_values = g_ternary, Omega = Omega3)
knitr::kable(
  table$table_body[-1,],
  digits = 3,
  caption = "Upper approximations (L1, CW, HD, UAP) for the ternary example \\(g\\)"
)
```

Table 6: Upper approximations (L1, CW, HD, UAP) for the ternary example g

	subset	g	f_L1	f_HD	f_CW	f_UAP	m_L1	m_HD	m_CW	m_UAP
2	{w1}	0.1	0.2	0.1	0.2	0.1	0.2	0.1	0.2	0.1
3	{w2}	0.1	0.2	0.3	0.2	0.1	0.2	0.3	0.2	0.1
4	{w3}	0.2	0.2	0.2	0.2	0.2	0.2	0.2	0.2	0.2
5	{w1,w2}	0.5	0.5	0.5	0.5	0.5	0.1	0.1	0.1	0.3
6	{w1,w3}	0.4	0.4	0.4	0.4	0.4	0.0	0.1	0.0	0.1
7	{w2,w3}	0.7	0.7	0.7	0.7	0.7	0.3	0.2	0.3	0.4
8	{w1,w2,w3}	1.0	1.0	1.0	1.0	1.2	0.0	0.0	0.0	0.0

```r
evaluate_lp_criteria(table_body = table$table_body, subsets = table$subsets, Omega = Omega3)
```

```
##    m_L1 m_HD m_CW m_UAP
## L1  3.2  3.2  3.2   3.2
## CW  0.8  0.8  0.8   0.8
## HD  0.4  0.4  0.4   0.8
```

Table 8

```r
table <- compute_table(g_values = g_ternary_tilde, Omega = Omega3)
knitr::kable(
  table$table_body[-1,],
  digits = 3,
  caption = "Upper approximations (L1, CW, HD, UAP) for the ternary example \\(\\tilde{g}\\)"
)
```

Table 7: Upper approximations (L1, CW, HD, UAP) for the ternary example $\tilde{g}$

	subset	g	f_L1	f_HD	f_CW	f_UAP	m_L1	m_HD	m_CW	m_UAP
2	{w1}	0.2	0.2	0.2	0.2	0.2	0.2	0.2	0.2	0.2
3	{w2}	0.2	0.2	0.2	0.2	0.2	0.2	0.2	0.2	0.2
4	{w3}	0.3	0.3	0.3	0.3	0.3	0.3	0.3	0.3	0.3
5	{w1,w2}	0.5	0.5	0.5	0.5	0.5	0.1	0.1	0.1	0.1
6	{w1,w3}	0.4	0.5	0.5	0.5	0.5	0.0	0.0	0.0	0.0
7	{w2,w3}	0.7	0.7	0.7	0.7	0.7	0.2	0.2	0.2	0.2
8	{w1,w2,w3}	1.0	1.0	1.0	1.0	1.0	0.0	0.0	0.0	0.0

```r
evaluate_lp_criteria(table_body = table$table_body, subsets = table$subsets, Omega = Omega3)
```

```
##    m_L1 m_HD m_CW m_UAP
## L1 3.40 3.40 3.40  3.40
## CW 0.85 0.85 0.85  0.85
## HD 0.30 0.30 0.30  0.30
```

Table 9

```r
table <- compute_table(g_values = g5_vals, Omega = Omega4)
knitr::kable(
  table$table_body[-1,],
  digits = 3,
  caption = "Upper approximations (L1, CW, HD, UAP) for the quaternary example"
)
```

Table 8: Upper approximations (L1, CW, HD, UAP) for the quaternary example

	subset	g	f_L1	f_HD	f_CW	f_UAP	m_L1	m_HD	m_CW	m_UAP
2	{w1}	0.147	0.158	0.158	0.147	0.147	0.158	0.158	0.147	0.147
3	{w2}	0.192	0.203	0.203	0.192	0.192	0.203	0.203	0.192	0.192
4	{w3}	0.162	0.173	0.173	0.162	0.162	0.173	0.173	0.162	0.162
5	{w4}	0.041	0.052	0.052	0.041	0.041	0.052	0.052	0.041	0.041
6	{w1,w2}	0.436	0.436	0.436	0.436	0.436	0.074	0.074	0.097	0.097
7	{w1,w3}	0.398	0.398	0.398	0.398	0.398	0.068	0.068	0.090	0.090
8	{w1,w4}	0.241	0.241	0.241	0.241	0.241	0.031	0.031	0.053	0.053
9	{w2,w3}	0.455	0.455	0.455	0.455	0.455	0.079	0.079	0.101	0.101
10	{w2,w4}	0.291	0.291	0.291	0.291	0.291	0.035	0.035	0.058	0.058
11	{w3,w4}	0.257	0.257	0.257	0.257	0.257	0.032	0.032	0.055	0.055
12	{w1,w2,w3}	0.755	0.755	0.755	0.788	0.788	0.000	0.000	0.000	0.000
13	{w1,w2,w4}	0.554	0.554	0.554	0.587	0.587	0.000	0.000	0.000	0.000
14	{w1,w3,w4}	0.513	0.513	0.513	0.547	0.547	0.000	0.000	0.000	0.000
15	{w2,w3,w4}	0.575	0.575	0.575	0.608	0.608	0.000	0.000	0.000	0.000
16	{w1,w2,w3,w4}	1.000	1.000	1.000	1.000	1.000	0.095	0.095	0.006	0.006

```r
evaluate_lp_criteria(table_body = table$table_body, subsets = table$subsets, Omega = Omega4)
```

```
##         m_L1      m_HD      m_CW     m_UAP
## L1 6.0592484 6.0592484 6.1481516 6.1481516
## CW 0.7692513 0.7692513 0.7692513 0.7692513
## HD 0.5088785 0.5088785 0.4644269 0.4644269
```

Table 10

Number of vertrices with the same value

```r
# ================================================================
# compute_vertex_minima()
# ----------------------------------------------------------------
# Computes, for a pseudo-belief function g, how many extreme points
# of the polytope M_g minimize each LP criterion (L1, HD, CW).
#
# The function:
#   1. constructs the polytope M_g via enumerate_vertices(),
#   2. enumerates all extreme points (vertices),
#   3. evaluates each LP objective over all vertices,
#   4. counts how many vertices achieve the minimum value.
#
# Input:
#   g_values ... numeric vector of g(A) in the same subset order
#   Omega    ... frame of discernment, e.g. c("w1","w2","w3")
#
# Output:
#   data.frame with one row and columns:
#       all_vertices ... total number of extreme points
#       L1           ... count of vertices minimizing L1 objective
#       HD           ... count of vertices minimizing HD objective
#       CW           ... count of vertices minimizing CW objective
#
# The result corresponds to Table 10 in the article.
# ==================================================================
compute_vertex_minima <- function(g_values, Omega) {

  # enumerate all vertices of M_g
  verts <- enumerate_vertices(Omega, g_values)$vertices
  subsets <- powerset(Omega)

  # number of vertices
  n_verts <- nrow(verts)

  # criterion → objective function
  crit_map <- list(
    L1 = objective_L1,
    HD = objective_HD,
    CW = objective_CW
  )

  # initialize result vector
  out <- c(
    all_vertices = n_verts,
    L1 = NA,
    HD = NA,
    CW = NA
  )

  # compute for each criterion
  for (crit in names(crit_map)) {
    w <- crit_map[[crit]](subsets, Omega)
    vals <- verts %*% w
    min_val <- min(vals)
    out[crit] <- sum(abs(vals - min_val) < 1e-14)
  }

  as.data.frame(t(out))
}


## ============================================================
## Construct Table 10 - counts of vertices minimizing each criterion
## ============================================================

row1 <- compute_vertex_minima(g_ternary,       Omega3)
row2 <- compute_vertex_minima(g_ternary_tilde, Omega3)
```

```r
row3 <- compute_vertex_minima(g5_vals,         Omega4)

# Add descriptive labels (first column)
Table10 <- rbind(
  "Ternary original g"= row1,
  "Ternary modified g_tilde" = row2,
  "Quaternary example"= row3
)

# Convert row names into a proper column
Table10 <- cbind(
  Example = rownames(Table10),
  Table10,
  row.names = NULL
)

# Print table
knitr::kable(
  Table10,
  caption = "Table 10 - Counts of extreme points and numbers achieving minimum value under each objecti
  align = "lcrrr",
  digits = 0
)
```

Table 9: Table 10 — Counts of extreme points and numbers achieving minimum value under each objective.

Example	all_vertices	L1	HD	CW
Ternary original g	28	3	3	3
Ternary modified g_tilde	27	4	4	4
Quaternary example	13889	1	1	14