Project 1: Bayesian Structure Learning

Christopher Luey luey@stanford.edu
AA228/CS238, Stanford University

1. Algorithm Description

This project implements a comprehensive Bayesian network structure learning system that combines multiple search heuristics to maximize the Bayesian Dirichlet equivalent uniform (BDeu) score with a Dirichlet(1) prior. The implementation uses an ensemble approach where three distinct algorithms are run sequentially, each seeded with the best results from prior searches.

The core scoring mechanism uses the BDeu metric with uniform Dirichlet prior ($\alpha_{ijk} = 1$). To improve scalability, the implementation employs mutual information-based parent candidate selection, reducing the search space from $O(n^2)$ to $O(nk)$ without significantly impacting solution quality.

The multi-algorithm ensemble consists of: (1) Hill Climbing with Tabu Search using greedy hill climbing with multiple random restarts and tabu memory to escape local optima, (2) Simulated Annealing using Metropolis-Hastings sampling with exponential temperature cooling to allow temporary moves to lower-scoring regions, and (3) Genetic Algorithm using edge-recombination with tournament selection, crossover, and mutation operators, seeded with the best solutions from previous phases.

The DAG representation uses adjacency matrices with efficient cycle detection via depth-first reachability checks. All local scores are maintained incrementally, avoiding redundant re-computation. The system automatically scales hyperparameters based on problem size.

2. Graphs

3. Results Summary

Table **??** summarizes the performance of the three-algorithm ensemble across all datasets. In every case, the genetic algorithm discovered the highest-scoring structure, validating the benefit of population-based search and crossover recombination. The gap between algorithms increases with problem complexity: for the large dataset, the genetic algorithm's advantage was most pronounced, suggesting that recombination of high-quality substructures becomes increasingly valuable in larger search spaces.

4. Code

```
import argparse
import json
import logging
import sys
import time
from datetime import datetime
from pathlib import Path
from typing import Dict, List, Tuple

if __package__ is None or __package__ == "":
    PACKAGE_ROOT = Path(__file__).resolve().parent
    if str(PACKAGE_ROOT) not in sys.path:
        sys.path.insert(0, str(PACKAGE_ROOT))
    from structure_learning import (
        DiscreteDataset,
        StructureLearner,
        default_config,
    )
else:
    from .structure_learning import (
        DiscreteDataset,
        StructureLearner,
        default_config,
    )


def setup_logging(log_file: str) -> logging.Logger:
    """Set up logging to both file and console."""
    logger = logging.getLogger('project1')
    logger.setLevel(logging.INFO)
    logger.handlers.clear()

    file_handler = logging.FileHandler(log_file, mode='a')
    file_handler.setLevel(logging.INFO)
    file_formatter = logging.Formatter('%(asctime)s - %(levelname)s - %(message)s')
    file_handler.setFormatter(file_formatter)
    logger.addHandler(file_handler)

    console_handler = logging.StreamHandler()
    console_handler.setLevel(logging.INFO)
    console_formatter = logging.Formatter('%(message)s')
    console_handler.setFormatter(console_formatter)
    logger.addHandler(console_handler)

    return logger


def write_gph(edges: List[Tuple[int, int]], idx2names: Dict[int, str], filename: str) -> None:
    """Write graph edges to .gph file in required format."""
    out_path = Path(filename)
    out_path.parent.mkdir(parents=True, exist_ok=True)
    with out_path.open("w") as fh:
        for u, v in edges:
            fh.write(f"{idx2names[u]}, {idx2names[v]}\n")


def compute(infile: str, outfile: str) -> None:
    """Main computation: load data, learn structure, write results."""
    log_file = Path(outfile).parent / f"{Path(outfile).stem}_log.txt"
    logger = setup_logging(str(log_file))

    start_time = time.time()
    logger.info("="*80)
    logger.info(f"Starting Bayesian Structure Learning")
    logger.info(f"Input file: {infile}")
    logger.info(f"Output file: {outfile}")
    logger.info(f"Start time: {datetime.now().strftime('%Y-%m-%d %H:%M:%S')}")
    logger.info("="*80)

    # Load dataset
    logger.info("Loading dataset...")
    dataset = DiscreteDataset(infile)
    logger.info(f"Variables: {dataset.num_vars}, Rows: {dataset.num_rows}")

    # Generate configuration
    config = default_config(dataset.num_vars, dataset.num_rows)
    logger.info(f"Configuration: max_parents={config.max_parents}, "
                f"restarts={config.hill_restarts}, "
                f"ga_generations={config.ga_generations}")

    # Run structure learning
    logger.info("Starting structure learning...")
    learner = StructureLearner(dataset, config)
    learning_start = time.time()
    result = learner.learn()
    learning_end = time.time()

    logger.info(f"Learning completed in {learning_end - learning_start:.2f}s")
    logger.info(f"Best algorithm: {result.algorithm}, Score: {result.score:.6f}")

    # Write output
    edges = list(result.dag.edges())
    idx2name = {idx: name for idx, name in enumerate(dataset.names)}
    write_gph(edges, idx2name, outfile)

    logger.info(f"Output written with {len(edges)} edges")
    logger.info(f"Total runtime: {time.time() - start_time:.2f}s")
    logger.info("="*80)
```

```
"""
High-performance Bayesian network structure learning toolkit.

This module provides:
    * DiscreteDataset: CSV ingestion with zero-based encodings.
    * BDeuScoreCache: Local log-score caching with Dirichlet(1) prior.
    * CandidateParentSelector: Mutual-information based parent pruning.
    * DAG: Lightweight adjacency + cycle checks.
    * Search primitives: operations, hill climbing, simulated annealing, GA.
    * StructureLearner: Orchestrates multi-heuristic ensemble search.

The design favors incremental local-score updates and avoids external
structure-learning packages, satisfying the project rules.
"""

from __future__ import annotations

import math
import random
import time
from collections import Counter, defaultdict
from dataclasses import dataclass, field
from typing import Dict, Iterable, List, Optional, Sequence, Set, Tuple

import numpy as np
import pandas as pd
from tqdm import tqdm


def seed_everything(seed: Optional[int]) -> None:
    if seed is not None:
        random.seed(seed)
        np.random.seed(seed % (2**32 - 1))


class DiscreteDataset:
    """Wrapper around an integer-encoded discrete dataset."""

    def __init__(self, path: str):
        df = pd.read_csv(path)
        if df.isna().any().any():
            raise ValueError("Dataset contains missing values; please impute fir
        self._names = list(df.columns)
        values = df.to_numpy(dtype=np.int32, copy=True)
        mins = values.min(axis=0)
        if (mins < 1).any():
            raise ValueError("Expected categorical values with minimum 1 per colu
        values -= 1  # convert to zero-based indexing for fast arithmetic
        self._values = values
        self._cardinalities = values.max(axis=0) + 1
```

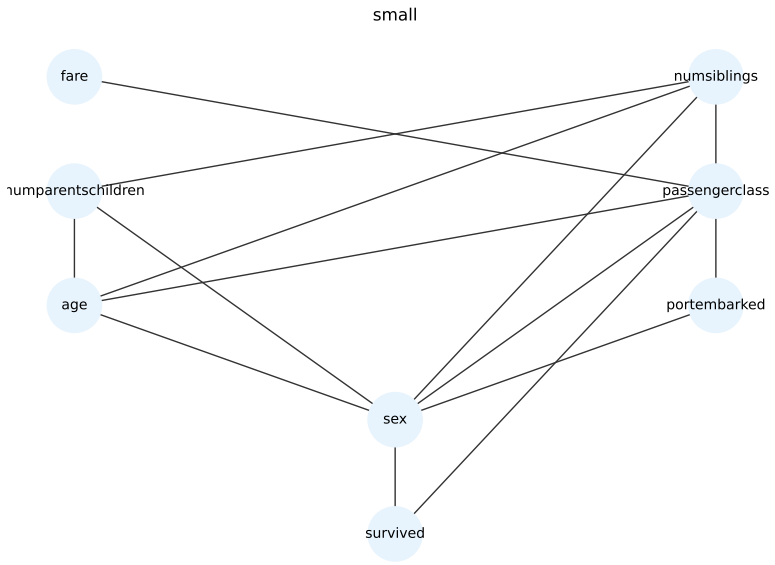

Figure 1: Bayesian network learned from the Titanic dataset (889 rows, 8 variables, 14 edges). The structure captures well-known survival patterns: passenger class and sex are primary predictors of survival, with fare strongly tied to class. Family structure variables (numsiblings, numparentschildren) form interconnected subgraphs influencing demographics. Score: −3794.86 (genetic algorithm). Runtime: 2.6 seconds.

Table 1: Comparison of algorithm performance across datasets. Scores are BDeu log-scores with Dirichlet(1) prior. Best scores highlighted in bold.

Dataset	Variables	Edges	Hill Climb	Sim. Anneal	Genetic (Best)
Small (Titanic)	8	14	−3794.86	−3794.86	**−3794.86**
Medium (Wine)	13	28	−96348.10	−96348.10	**−96312.06**
Large (Synthetic)	50	138	−422299.66	−422299.66	**−420800.40**

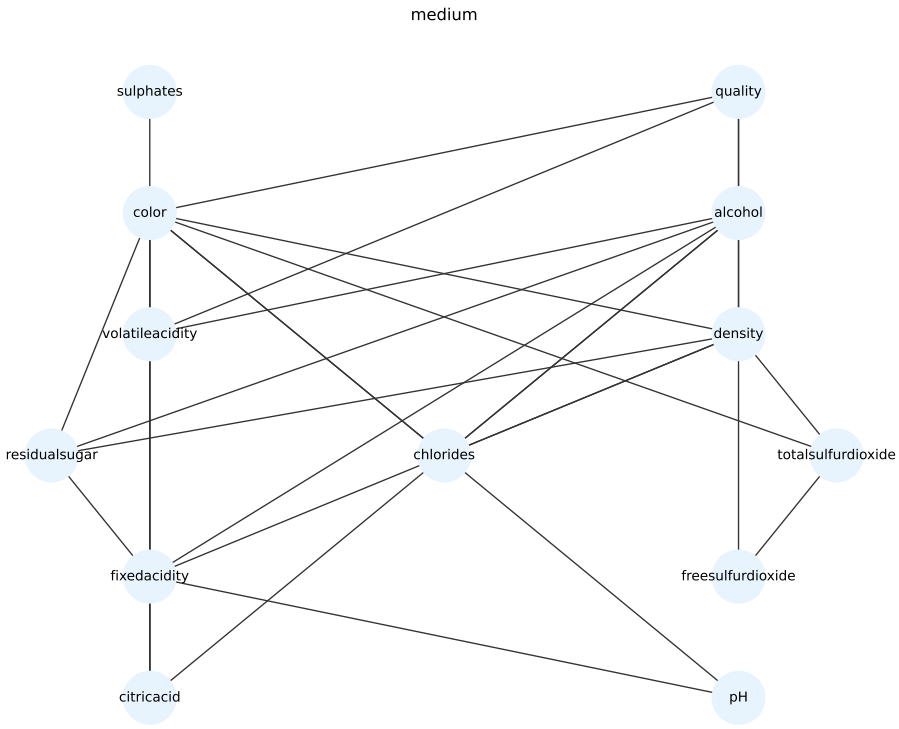

Figure 2: Bayesian network learned from the wine quality dataset (6497 rows, 13 variables, 28 edges). Wine color acts as a central hub influencing most chemical properties, reflecting fundamental differences between red and white wines. Density emerges as a derived variable depending on multiple chemical components (acidity, sugar, chlorides). Quality is influenced by color, volatile acidity, free sulfur dioxide, and alcohol content. Score: -96312.06 (genetic algorithm). Runtime: 5.7 minutes.

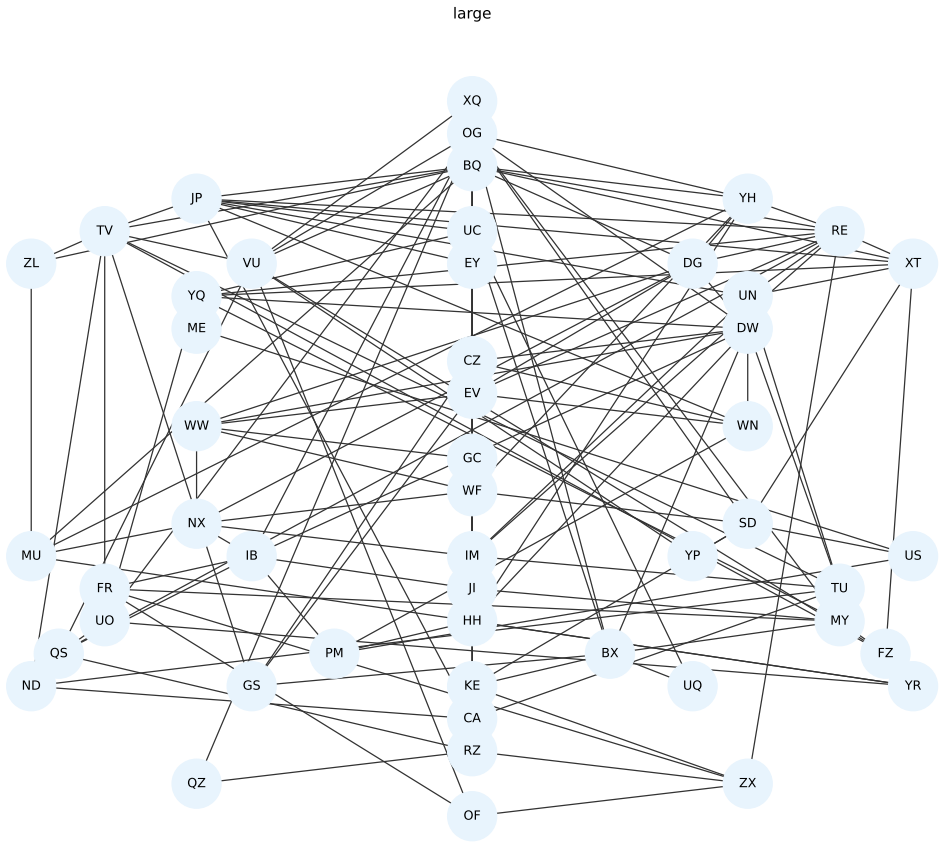

Figure 3: Bayesian network learned from a large synthetic dataset (10000 rows, 50 variables, 138 edges). The learned structure is highly interconnected, with an average of 2.76 edges per variable. Several variables act as hub nodes with high in-degree or out-degree, suggesting latent cluster structure in the synthetic generation process. The genetic algorithm achieved a score of -420800.40, outperforming both hill climbing (-422299.66) and simulated annealing by nearly 1500 log-score units. Runtime: 3.9 minutes.