ASP∀: An Open Educational Resource for Answer Set Programming.

Joshua T. Guerin, Ph.D., David Gonzalez Sua, and Jackson Madsen

The University of Northern Colorado

May 18, 2026

Introduction

I like applications of programming languages in CS education.

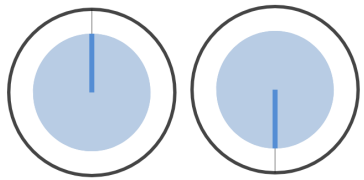
- Programming contest problem development (open.kattis.org)
- Logic programming for discrete courses (ACM Mid-Southeast)
- Quantum programming pedagogy (SIGCSE)

Declarative languages are some of my favorites.

- In particular, Answer Set Programming (ASP) dialects.
- Taught in AI, Programming Languages, etc.
- Over the years I developed several solutions.
  - Organizing my thoughts online felt natural.







Goals

- Simple, well organized/documented examples.
- A wide variety of classical problems.
- Self-contained entries (where possible).
  - Description
  - Illustrative examples.
  - Working source code solutions.
  - Related problems.
  - Additional utilities (where necessary) to improve understanding.
- Intentional focus on rule and predicate-based solutions.

Organization

Category	Problems
Number Theory	Composite Numbers
	Prime Sieve
	Perfect Numbers
Numerical Set/Partitioning	Subset Sum
	Equal Sum Partition
	Numerical 3-Dimensional Matching
Combinatorial Optimization	Knapsack
	Bin-Packing
Puzzles/Games	I'm my own Grandpa!
	N-Queens
	Sudoku
Network Flow	Max-Flow
Graphs	Graph Coloring
	Bipartite Graph Partitioning
	Chromatic Numbers
	Clique
	Max Clique
	Dominating Set
	Minimum Dominating Set
	Independent Set
	Vertex Cover
	Minimum Vertex Cover
	Ramsey Graphs
Deterministic Planning	Hamiltonian Path
	Traveling Salesman
Sequential/Time-Based Planning	Wolf, Goat, and Cabbage

And growing!

Subset-Sum Problem

An illustrative example, the Subset Sum problem:

Definition (The Subset-Sum Problem)

Given an input set $S \subset \mathbb{Z}$, is there a subset $S' \subseteq S$ such that $\sum S' = n$, for some $n \in \mathbb{Z}$?

E.g., If $S = \{1, 3, 5, 7, 9\}$ and $n = 15$ then $S' = \{3, 5, 7\}$.

Subset-Sum Problem

An illustrative example, the Subset Sum problem:

Definition (The Subset-Sum Problem)

Given an input set $S \subset \mathbb{Z}$, is there a subset $S' \subseteq S$ such that $\sum S' = n$, for some $n \in \mathbb{Z}$?

E.g., If $S = \{1, 3, 5, 7, 9\}$ and $n = 15$ then $S' = \{3, 5, 7\}$.

High-level constraints:

- Inputs ($s \in S$) are candidates for the sum.
- The sum of the values must equal n.

This reasoning guides the implementation.

Workflow

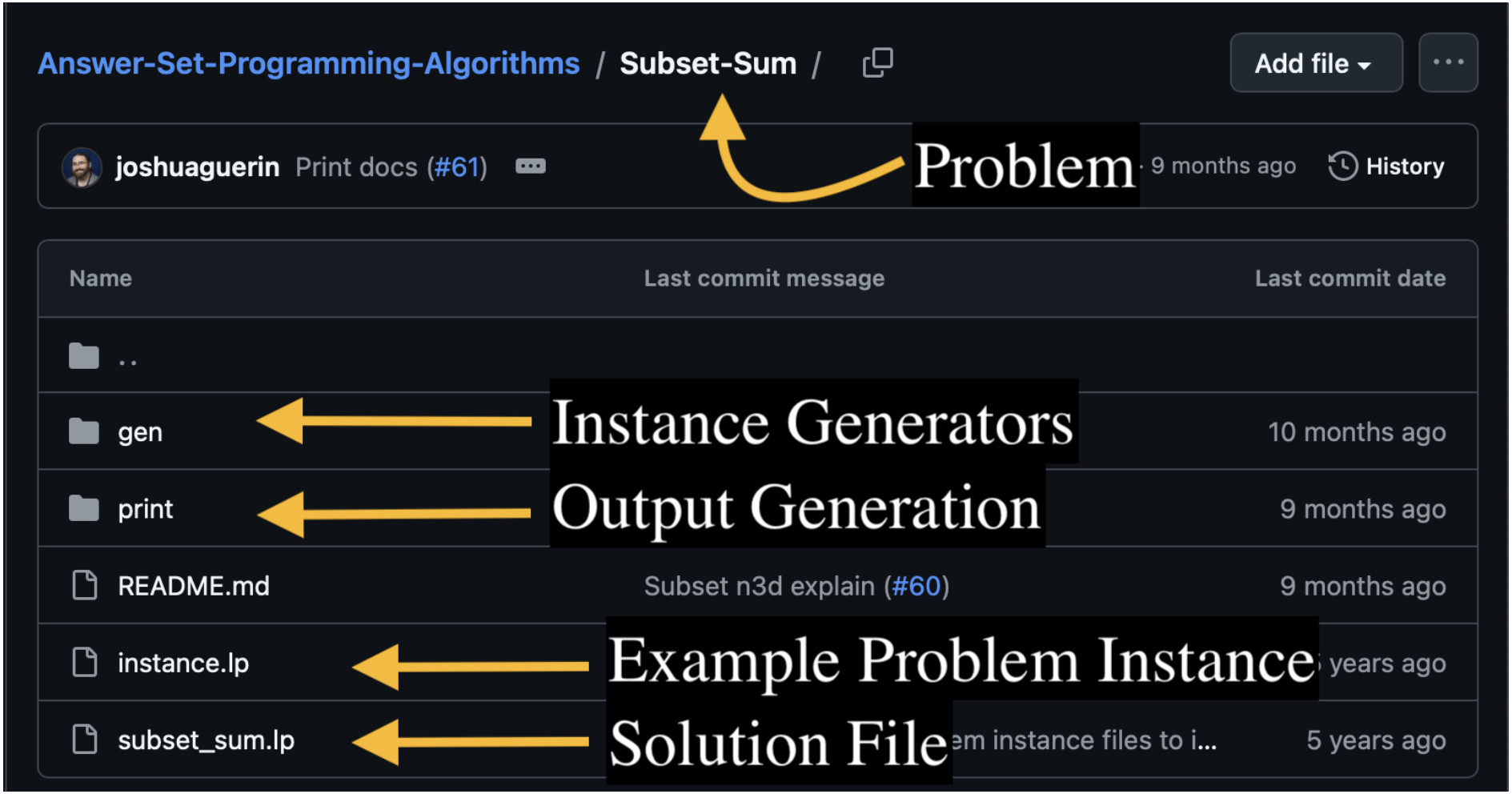
The implementation is derived directly from the problem constraints:

```
% Values are candidates to be included in the sum.
{sum(X) : value(X)}.

%% TEST
% Values in sum/1 must sum to n.
:- #sum {X : sum(X)} != n.

% DISPLAY
#show sum/1.
```

We just need a problem instance:

```
value(1 ; 20 ; 3 ; 18 ; 2 ; 10 ; 11 ; 14 ; 4 ; 12).
```

Example: File Organization



Example: Entry Text (README)

Problem: Subset Sum

Description

The Subset Sum problem is a classical NP-Complete problem in computing and number theory. Given a set of integers, *S*, and a target value *n*, is there a subset of *S* that sums to exactly *S*.

Problem Overview

Example

Let *S* = {1, 2, 3, 4, 10, 11, 12, 14, 18, 20}, and a query of *n*=26.

- This would be satisfied by the subset: {2, 3, 10, 11}, as 2+3+10+11=26.

Example Instance & Solution

Implementational Details

Note that the declaration of `value(k ; k).` in an instance.lp file will result in a *single* instance of `k` having property `value`. For simplicity this implementation was designed where *S* is a *set*, allowing inclusion, but not duplicates. For a multi-set-based implementation (see below), modification would be necessary.

Relevant Details

Problem Variants

There are several variants of this problem that are NP-Complete. Some assumptions in this encoding are:

- A set of values is encoded (vs. a multiset) because the encoding of values is a single predicate.
- Generating software is over positive integers (an arbitrary decision that should be easy to modify).

Problem Variants

Related

Related Problems

See also:

- equal sum partitioning and

Workflow

Genrate a new instance:

```
python3 ./gen/generate.py > in.lp
value(58 ; 34 ; 87 ; 92 ; 95 ; 1 ; 85 ; 36 ;...).
```

Solve:

```
clingo subset_sum.lp in.lp -c n=2567 > out.txt
Answer: 1
sum(34) sum(87) sum(92) sum(95) sum(1) sum(85) sum(36)...
```

Print formatted output:

```
cat out.txt | python3 ./print/print.py
2567 = 1 + 6 + 8 + 9 + 13 + 16 + 21 + 30 + 32 + 33 + 34 +
35 + 36 + 37 + 40 + 41 + 48 + 49 + 50 + 51 + 53 + 54 + 56 +
57 + 59 + 60 + 62 + 63 + 65 + 66 + 72 + 77 + 78 + 79 + 80 +
81 + 83 + 85 + 87 + 91 + 92 + 93 + 94 + 95 + 97 + 98
```

Note: Utilities are unix-style, and can be joined by pipes.

Instance Generation & Visualization

Additional tooling was deemed necessary.

- Outputs can be a bit obtuse (e.g., sudoku, graphs).
- Predicate-based inputs require specialized formatting.
- A single, toy domain may not be sufficient for learners.
- An entire pipeline:
  - Instance generation.
  - Problem solutions, with examples.
  - Output formatting and presentation.

Instance Generation

Algorithms for generation depended on the problem domain.

I.e., Many *completely random* instances would have no solution.

Example generation methods:

- Randomly add or subtract graph edges from a spanning tree or clique.
- Sticks-stones/stars-bars: for many number theory problems.
- Backtracking DFS: to generate a Sudoku, then remove tiles.

★★★★||★|★★|

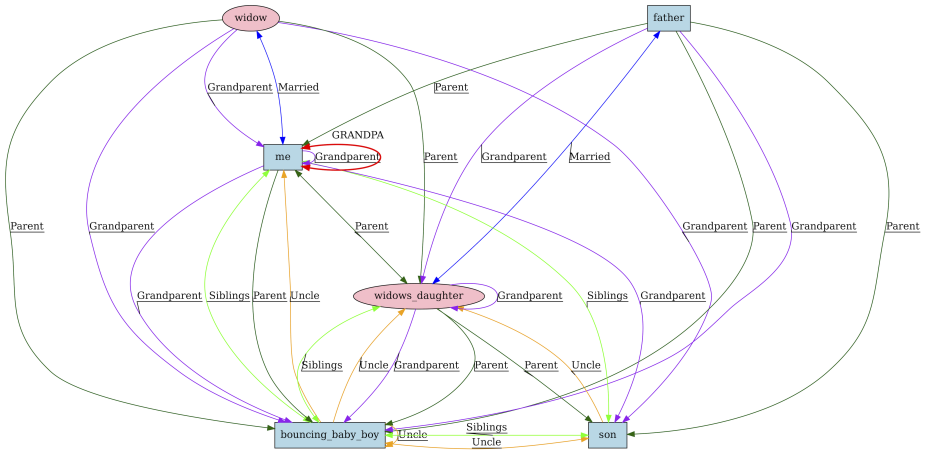
Visualization

Large predicate solutions are not human readable.

- Clingo doesn't support traditional IO.
- Solution: Roll our own parsers.

Standard methods of generation: Formatted text or graphviz.

```
Time:  0 ['f', 'w', 'g', 'c'] g > []
Time:  1 ['c', 'w'] <   ['g', 'f']
Time:  2 ['c', 'w', 'f'] c > ['g']
Time:  3 ['w'] < g ['c', 'g', 'f']
Time:  4 ['g', 'w', 'f'] w > ['c']
Time:  5 ['g'] <   ['c', 'w', 'f']
Time:  6 ['g', 'f'] g > ['c', 'w']
Time:  7 [] <   ['f', 'w', 'g', 'c']
```



Future Directions

- A companion system, for absolute beginners.
  - Basic logic & language mechanics.
  - A walk-through of solution construction.
  - Exercises and appropriate workflows.

- New problem categories and solutions.
  - Data Mining
  - Machine learning
  - Scheduling
  - Stochastic planning

Author Information

Contact information:

- joshua.guerin@unco.edu
- gonz9032@bears.unco.edu
- jackson.madsen@unco.edu

`github.com/joshuaguerin/Answer-Set-Programming-Algorithms`

Or search "Answer Set Programming Algorithms" in your favorite engine!