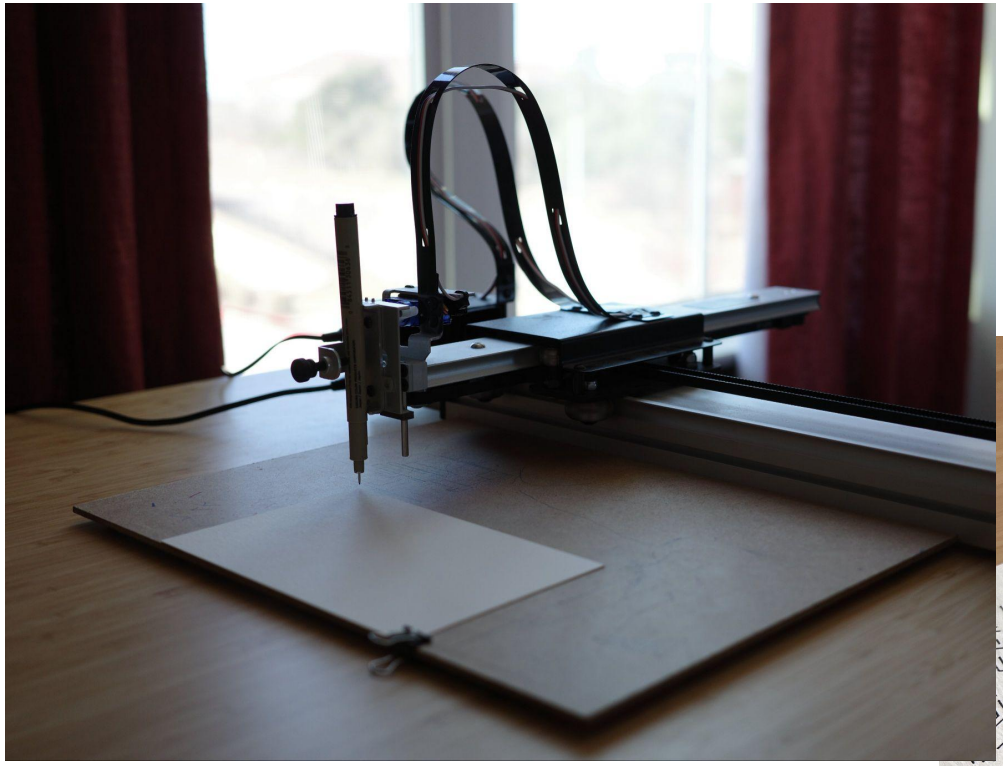
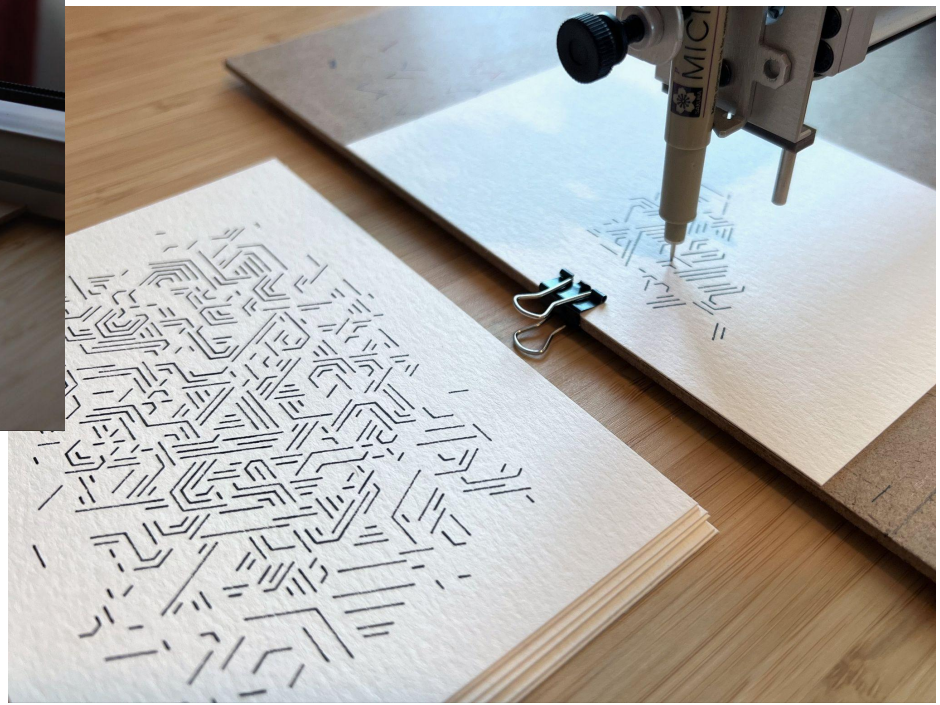
Organizing an international postcard exchange with Rust

A gentle introduction to constrained optimization



@ippsketch
#ptpx 2023



Left:
@taylorbaldwin
ptpx 2023

Right:
@adamfuhrer
ptpx 2023

Constrained Optimization

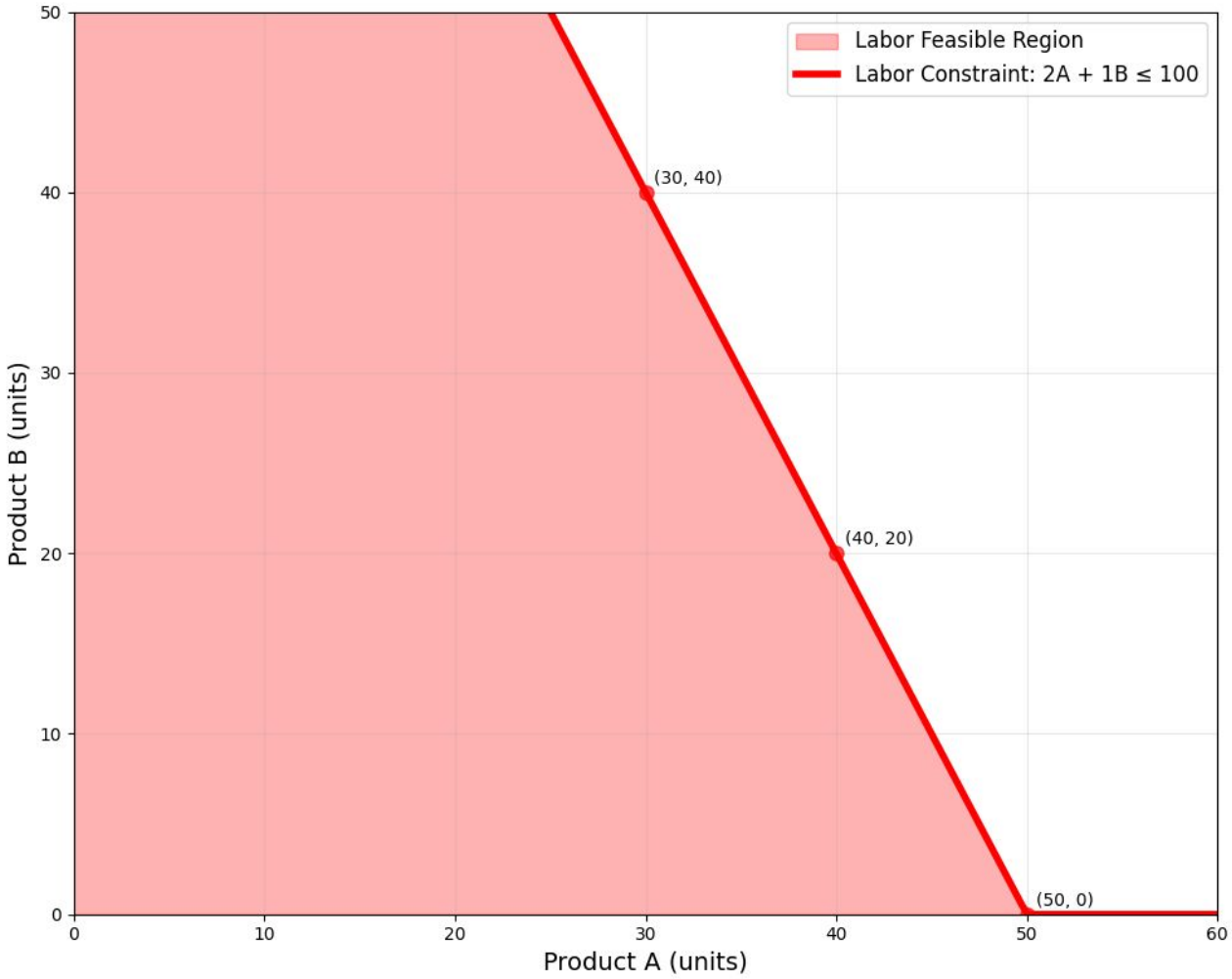
A factory produces two products (A and B)

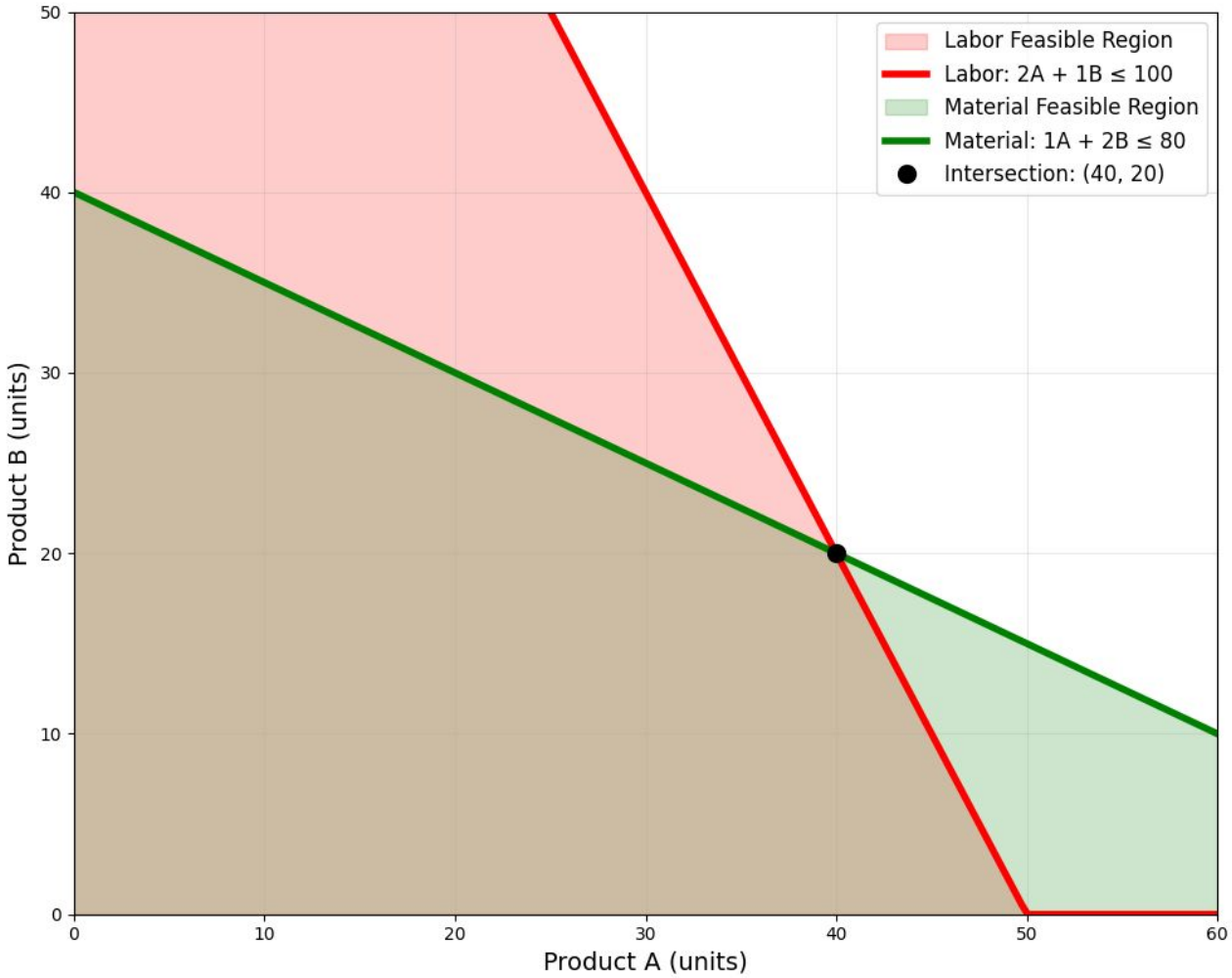
Product A: profit = $40, requires 2 hours labor, 1 unit material

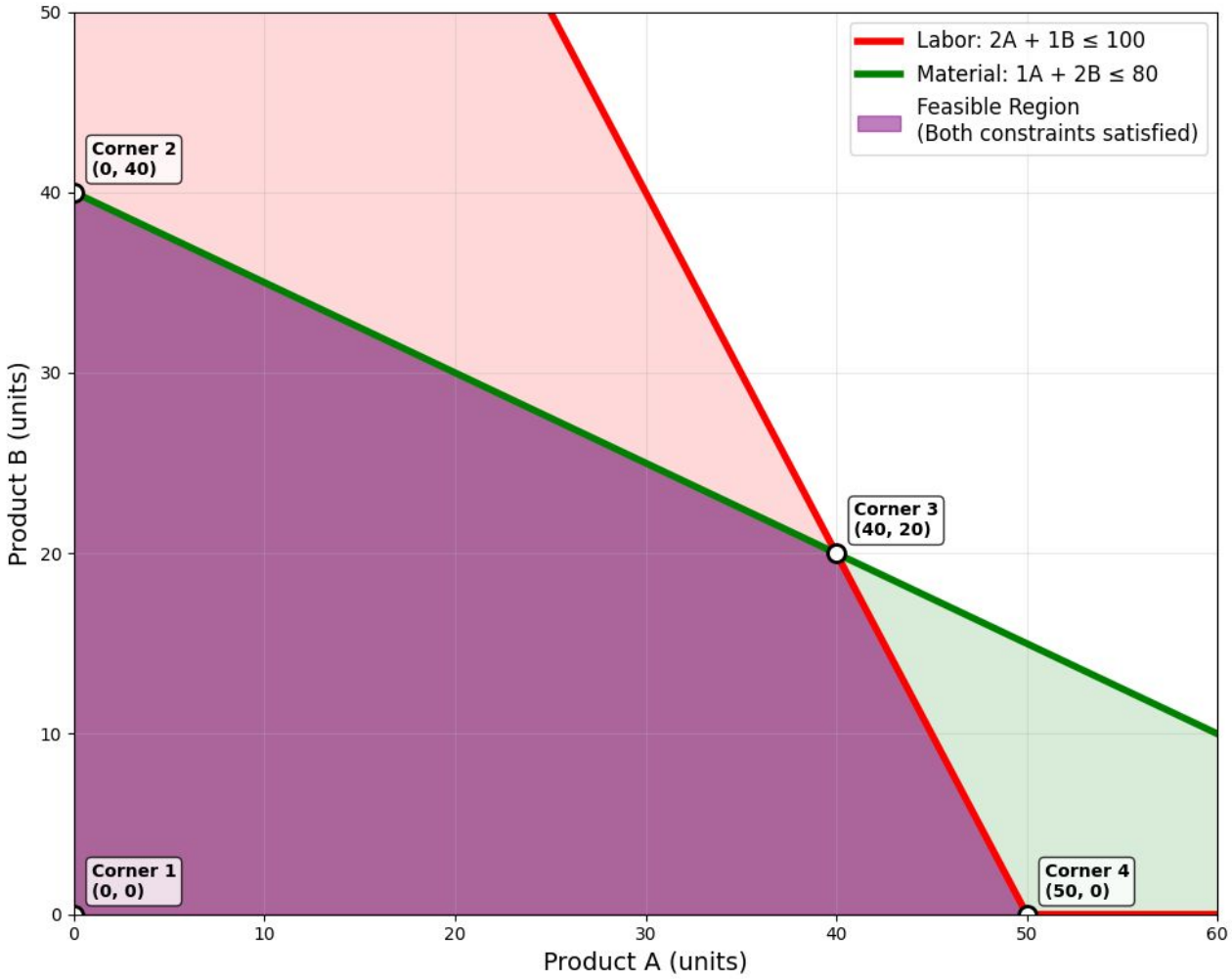
Product B: profit = $30, requires 1 hour labor, 2 units material

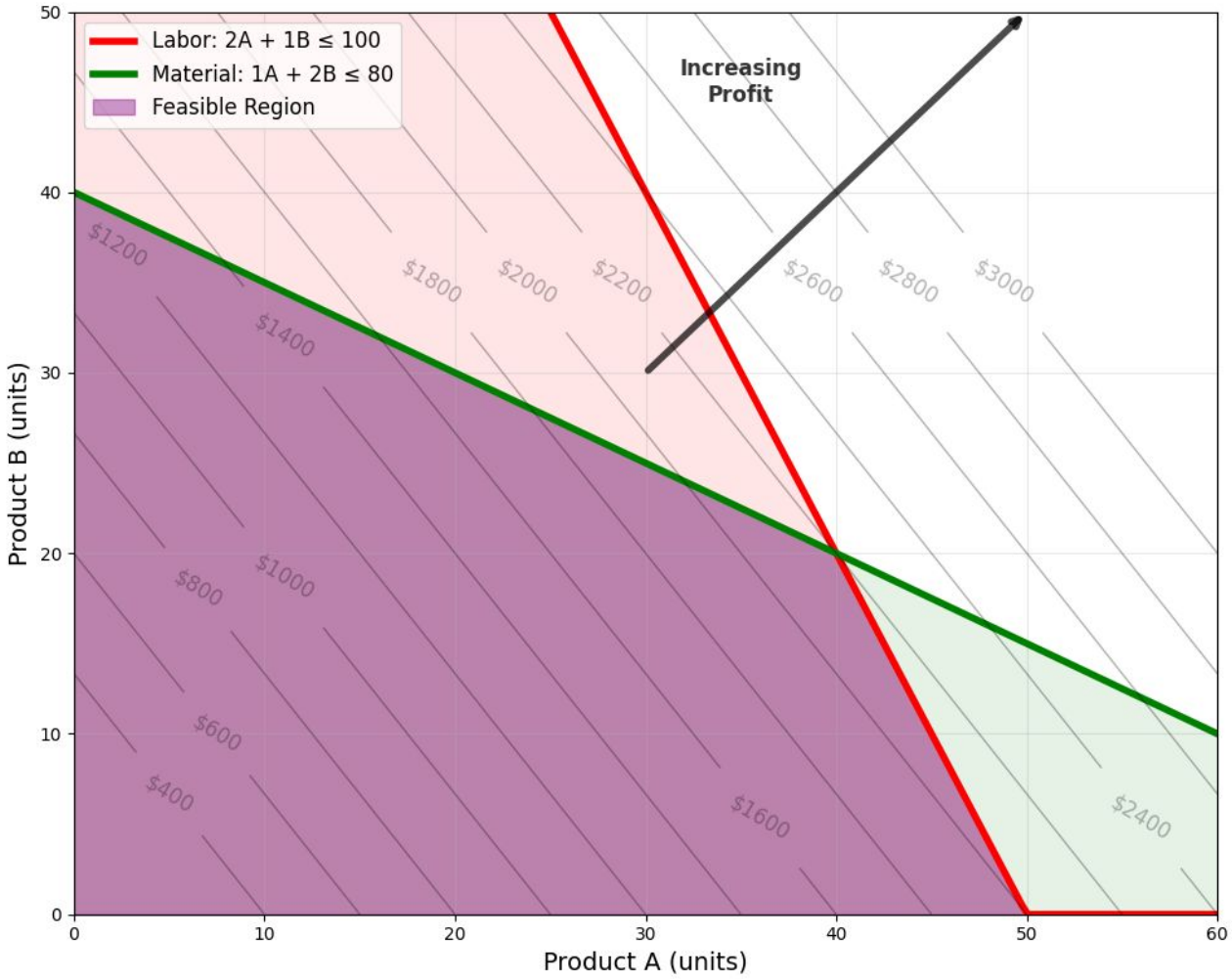
Constraints: max 100 hours labor, max 80 units material

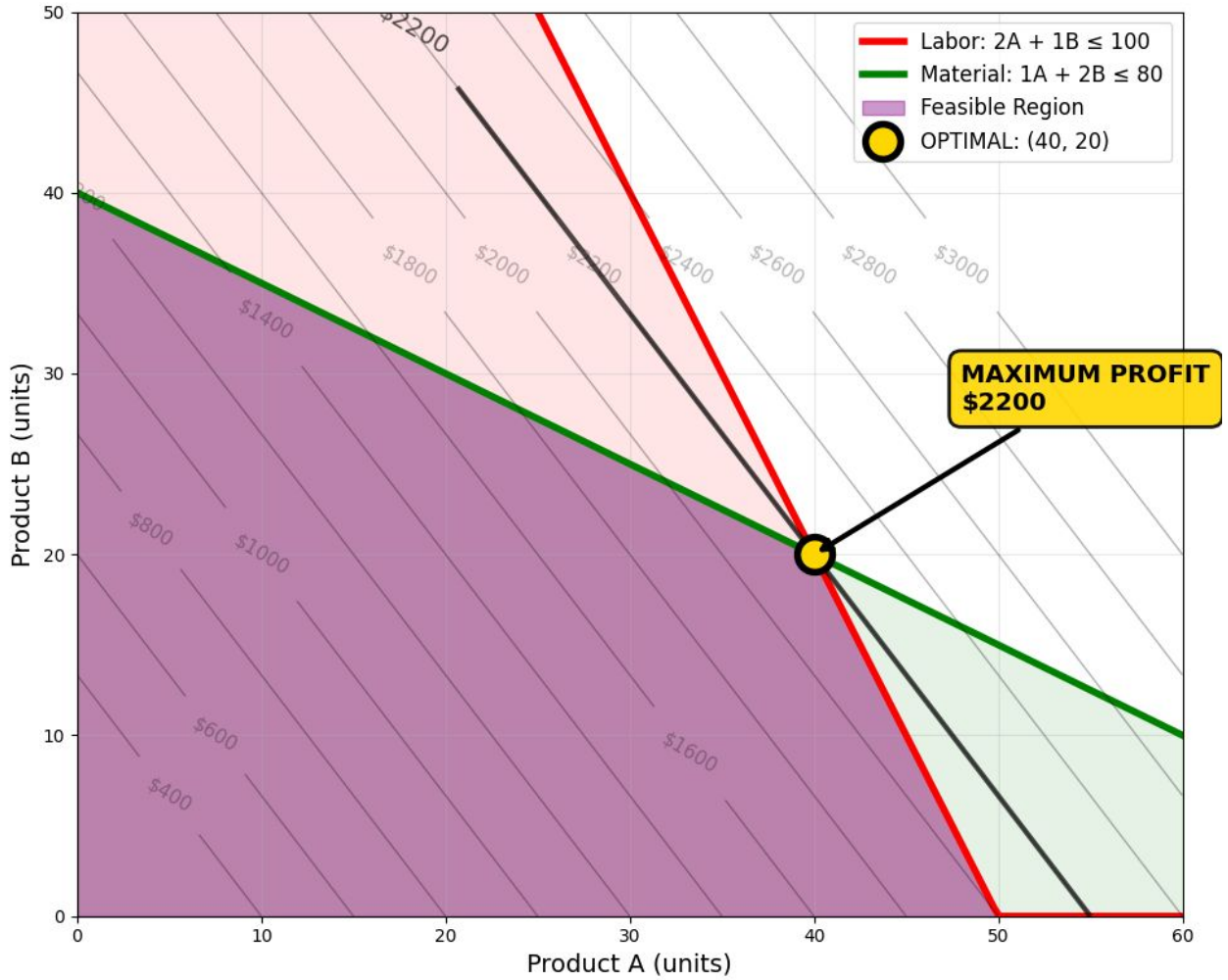
Goal: **maximize profit**

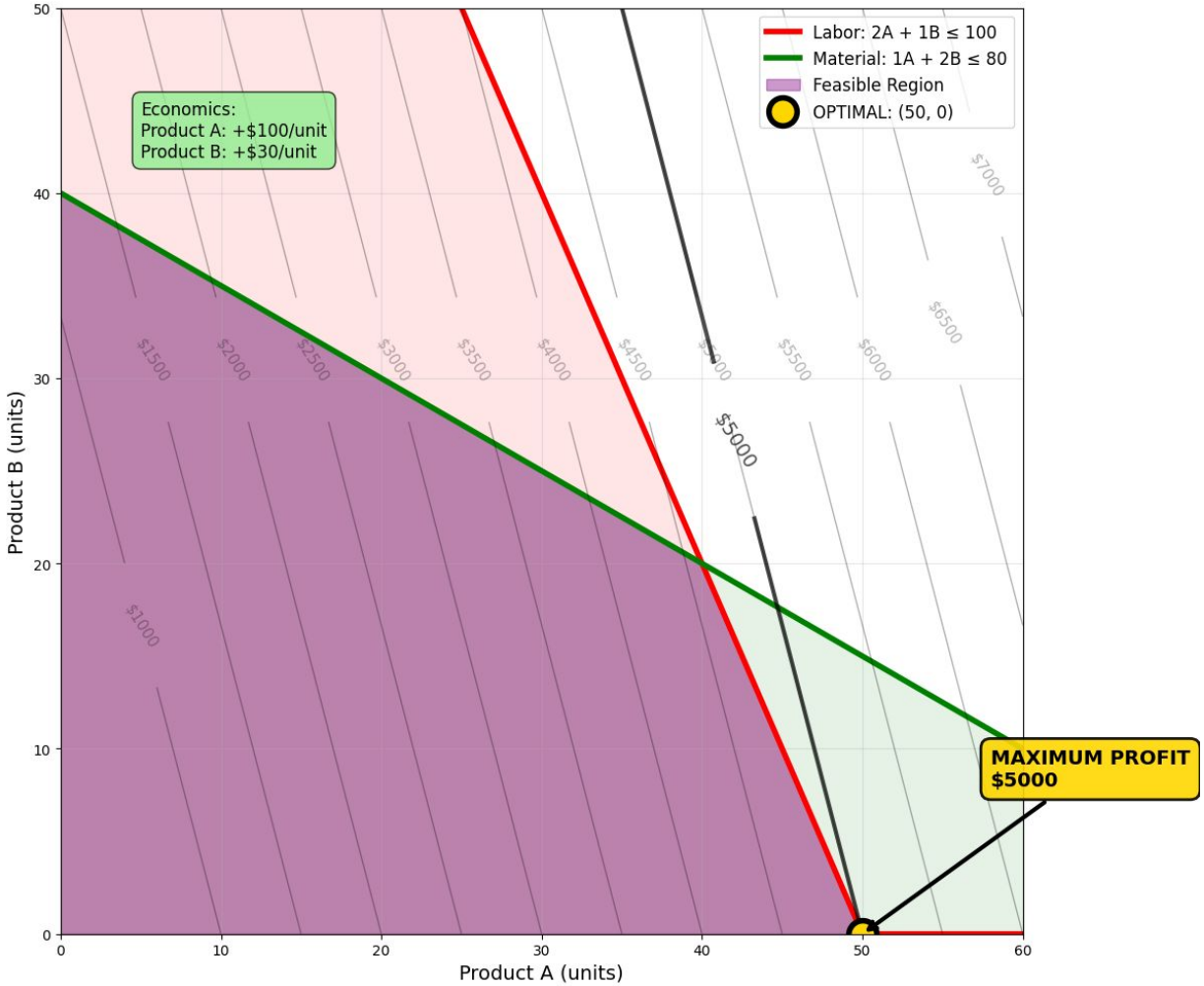

Labor Feasible Region
Labor Constraint: 2A + 1B ≤ 100
(30, 40)
(40, 20)
(50, 0)
Product B (units)
Product A (units)
0
10
20
30
40
50
60


Labor Feasible Region
Labor: 2A + 1B ≤ 100
Material Feasible Region
Material: 1A + 2B ≤ 80
Intersection: (40, 20)
Product B (units)
Product A (units)
0
10
20
30
40
50
60


Labor: 2A + 1B ≤ 100
Material: 1A + 2B ≤ 80
Feasible Region
(Both constraints satisfied)
Corner 2
(0, 40)
Corner 3
(40, 20)
Corner 1
(0, 0)
Corner 4
(50, 0)
Product B (units)
Product A (units)
0
10
20
30
40
50
60


Labor: 2A + 1B ≤ 100
Material: 1A + 2B ≤ 80
Feasible Region
Increasing Profit
$1200
$1400
$1800
$2000
$2200
$2600
$2800
$3000
$800
$1000
$600
$400
$1600
$2400
Product B (units)
Product A (units)
0
10
20
30
40
50
60

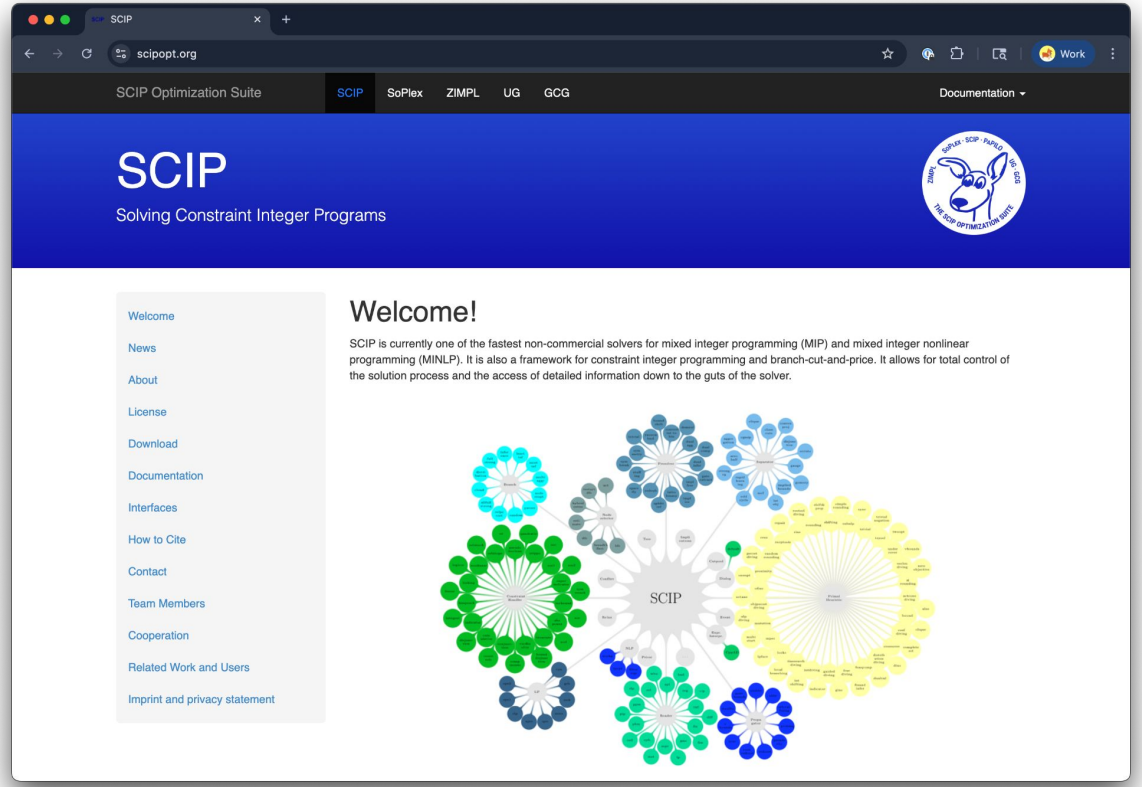

Labor: 2A + 1B ≤ 100
Material: 1A + 2B ≤ 80
Feasible Region
OPTIMAL: (40, 20)
MAXIMUM PROFIT
$2200
$2200
$1800
$2000
$2400
$2600
$2800
$3000
$1400
$800
$1000
$600
$400
$1600
Product B (units)
Product A (units)
0
10
20
30
40
50
60


Labor: 2A + 1B ≤ 100
Material: 1A + 2B ≤ 80
Feasible Region
OPTIMAL: (50, 0)
Economics:
Product A: +$100/unit
Product B: +$30/unit
$1000
$1500
$2000
$2500
$3000
$3500
$4000
$4500
$5000
$5500
$6000
$6500
$7000
MAXIMUM PROFIT
$5000
Product A (units)
Product B (units)
0
10
20
30
40
50
60

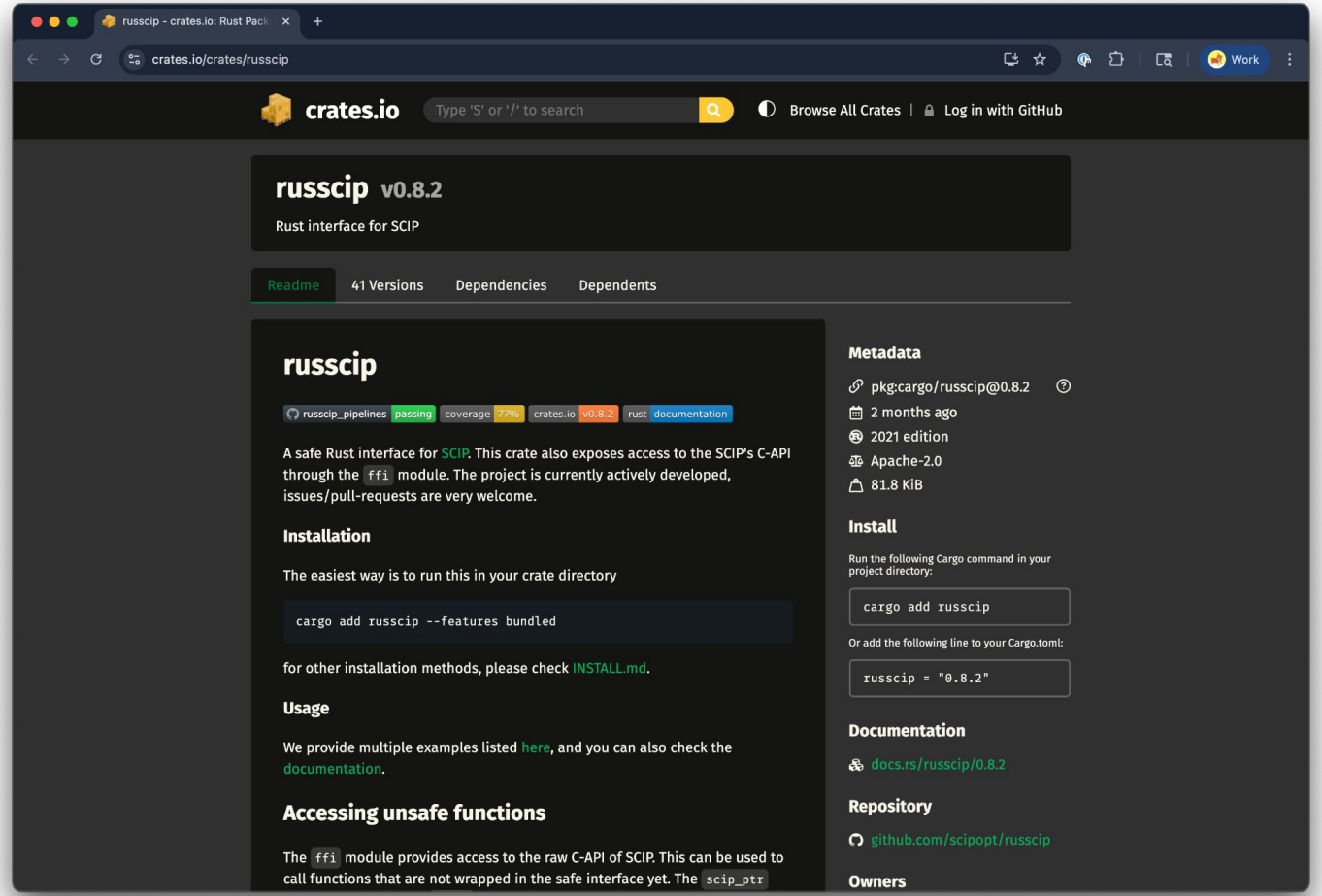
SCIP

Zuse Institute Berlin

Apache 2.0



https://www.scipopt.org
https://github.com/scipopt/scip


russcip - crates.io: Rust Pack
crates.io/crates/russcip
Work
crates.io
Type 'S' or '/' to search
Browse All Crates
Log in with GitHub
russcip v0.8.2
Rust interface for SCIP
Readme
41 Versions
Dependencies
Dependents
russcip
russcip_pipelines passing
coverage 77%
crates.io v0.8.2
rust documentation
A safe Rust interface for SCIP. This crate also exposes access to the SCIP's C-API through the ffi module. The project is currently actively developed, issues/pull-requests are very welcome.
Installation
The easiest way is to run this in your crate directory
cargo add russcip --features bundled
for other installation methods, please check INSTALL.md.
Usage
We provide multiple examples listed here, and you can also check the documentation.
Accessing unsafe functions
The ffi module provides access to the raw C-API of SCIP. This can be used to call functions that are not wrapped in the safe interface yet. The scip_ptr
Metadata
pkg:cargo/russcip@0.8.2
2 months ago
2021 edition
Apache-2.0
81.8 KiB
Install
Run the following Cargo command in your project directory:
cargo add russcip
Or add the following line to your Cargo.toml:
russcip = "0.8.2"
Documentation
docs.rs/russcip/0.8.2
Repository
github.com/scipopt/russcip
Owners

```rust
use russcip::prelude::*;

fn main() {
  let mut model = Model::new()
    .hide_output()
    .include_default_plugins()
    .create_prob("production_planning")
    .set_obj_sense(ObjSense::Maximize);
  // ...
}
```

```rust
let var_a = model.add_var(
    0.,              // lower bound (can't create negative products)
    f64::INFINITY,   // upper bound (no inherent limit)
    40.,             // objective coefficient ($40/unit)
    "product_A",     // name (arbitrary label)
    VarType::Integer // always a whole number
);
```

```rust
let var_a = model.add_var(
    0.,                 // lower bound (can't create negative products)
    f64::INFINITY,      // upper bound (no inherent limit)
    40.,                // objective coefficient ($40/unit)
    "product_A",        // name (arbitrary label)
    VarType::Integer // always a whole number
);
let var_b = model.add_var(
    0.,                 // lower bound (can't create negative products)
    f64::INFINITY,      // upper bound (no inherent limit)
    30.,                // objective coefficient ($30/unit)
    "product_B",        // name (arbitrary label)
    VarType::Integer // always a whole number
);
```

```rust
let _labor_constraint = model.add_cons(
    vec![&var_a, &var_b], // variables involved
    &[2.0, 1.0],          // coefficients on each variable, respectively
    -f64::INFINITY,       // lower bound
    100.0,                // upper bound
    "labor_constraint"    // name (arbitrary label)
);
```

```
let _labor_constraint = model.add_cons(
    vec![&var_a, &var_b],
    &[2.0, 1.0],
    -f64::INFINITY,
    100.0,
    "labor_constraint"
);
```

Product A: profit = $40, requires 2 hours labor, 1 unit material

Product B: profit = $30, requires 1 hour labor, 2 units material

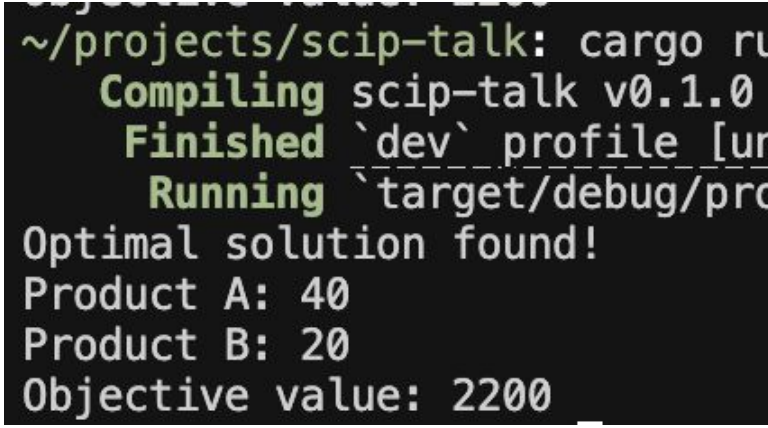
Constraints: max 100 hours labor, max 80 units material

```
let _labor_constraint = model.add_cons(
    vec![&var_a, &var_b], // variables involved
    &[2.0, 1.0],          // coefficients on each variable, respectively
    -f64::INFINITY,       // lower bound
    100.0,                // upper bound
    "labor_constraint"    // name (arbitrary label)
);
```

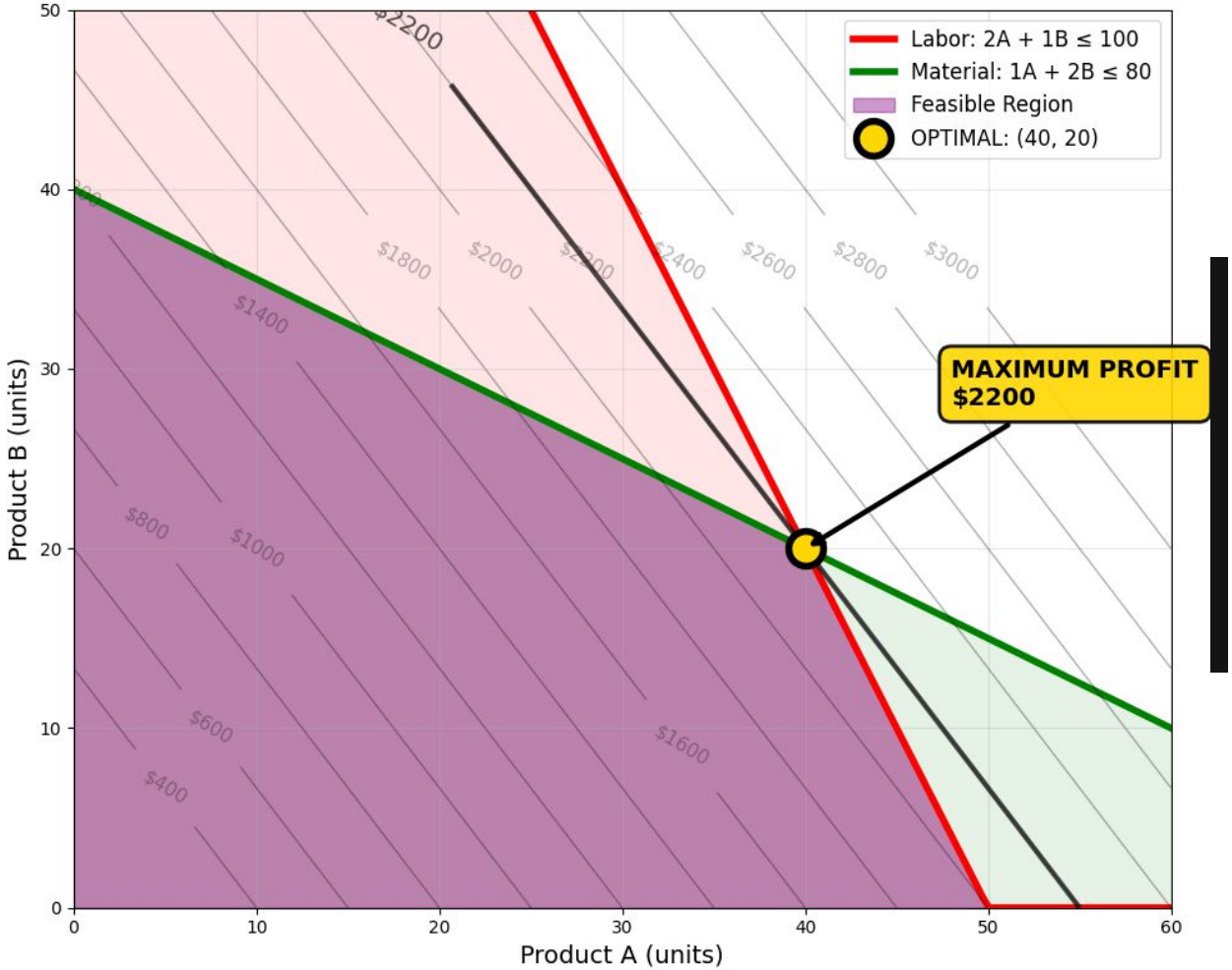
```rust
let _labor_constraint = model.add_cons(
    vec![&var_a, &var_b], // variables involved
    &[2.0, 1.0],          // coefficients on each variable, respectively
    -f64::INFINITY,       // lower bound
    100.0,                // upper bound
    "labor_constraint"    // name (arbitrary label)
);
let _material_constraint = model.add_cons(
    vec![&var_a, &var_b], // variables involved
    &[1.0, 2.0],          // coefficients on each variable, respectively
    -f64::INFINITY,       // lower bound
    80.0,                 // upper bound
    "material_constraint" // name (arbitrary label)
);
```

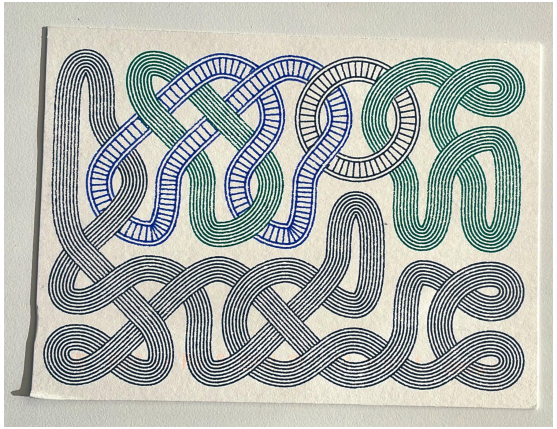
```
let solved_model = model.solve();

match solved_model.status() {
  Status::Optimal => {
    let sol = solved_model.best_sol().unwrap();
    let a_value = sol.val(&var_a);
    let b_value = sol.val(&var_b);
    let objective_value = solved_model.obj_val();
    println!("Optimal solution found!");
    println!("Product A: {}", a_value);
    println!("Product B: {}", b_value);
    println!("Objective value: {}", objective_value);
  }
  _ => {
    return Err(
      format!("Solver finished with status: {:?}",
        solved_model.status()).into()
    );
  }
}
```

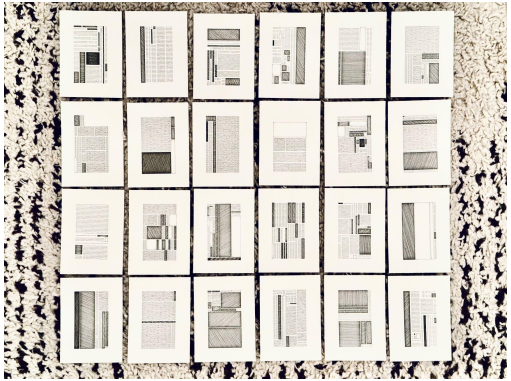
```rust
let solved_model = model.solve();

match solved_model.status() {
  Status::Optimal => {
    let sol = solved_model.best_sol().unwrap();
    let a_value = sol.val(&var_a);
    let b_value = sol.val(&var_b);
    let objective_value = solved_model.obj_val();
    println!("Optimal solution found!");
    println!("Product A: {}", a_value);
    println!("Product B: {}", b_value);
    println!("Objective value: {}", objective_value);
  }
  _ => {
    return Err(
      format!("Solver finished with status: {:?}",
        solved_model.status()).into()
    );
  }
}
```



```
~/projects/scip-talk: ca
   Compiling scip-talk v
    Finished `dev` profi
     Running `target/deb
Optimal solution found!
Product A: 40
Product B: 20
Objective value: 2200
```







Back to postcards!

Model

Variables: N^2 variables represented in NxN matrix **X**

$x_{i,j}$ = 1 if participant *i* sends a postcard to participant *j*, otherwise 0.

Constraints

- No participant sends a card to themself.
- No participant sends more cards than they signed up for.
- Every participant receives as many cards as they receive.
- No pair of participants send and receive from each other.

Objective

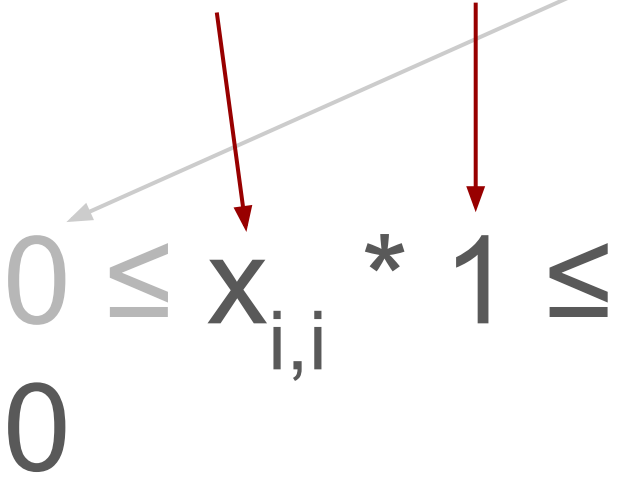
Maximize number of cards sent

```
let mut model = Model::new()
    .hide_output()
    .include_default_plugins()
    .create_prob("pairings")
    .set_obj_sense(ObjSense::Maximize);
```

```rust
// x[i][j] is 1 if person i sends a card to person j
let mut x = Vec::new();
for _ in 0..n {
    let mut row = Vec::new();
    for _ in 0..n {
        row.push(model.add_var(0., 1., 1., "adjacency", VarType::Binary));
    }
    x.push(row);
}
```

```rust
// Nobody sends a card to themself.
for i in 0..n {
    model.add_cons(vec![&x[i][i]], &[1.], 0., 0., "no_self_exchange");
}
```

```rust
// Nobody sends a card to themself.
for i in 0..n {
    model.add_cons(vec![&x[i][i]], &[1.], 0., 0., "no_self_exchange");
}
```



(i.e. $x_{i,i} == 0$)

```
// Nobody sends more cards than they signed up for.
for i in 0..n {
    let num_cards = cards_for_participant[i];
    model.add_cons(
        x[i].iter().collect(),
        &vec![1.0; n],
        0.,
        num_cards as f64,
        "num_cards",
    );
}
```

```rust
// Nobody sends more cards than they signed up for.
for i in 0..n {
    let num_cards = cards_for_participant[i];
    model.add_cons(
        x[i].iter().collect(),
        &vec![1.0; n],
        0.,
        num_cards as f64,
        "num_cards",
    );
}
```

$$x_{i,0} * 1 + x_{i,1} * 1 + \ldots \leq \text{cardsForParticipant}_i$$

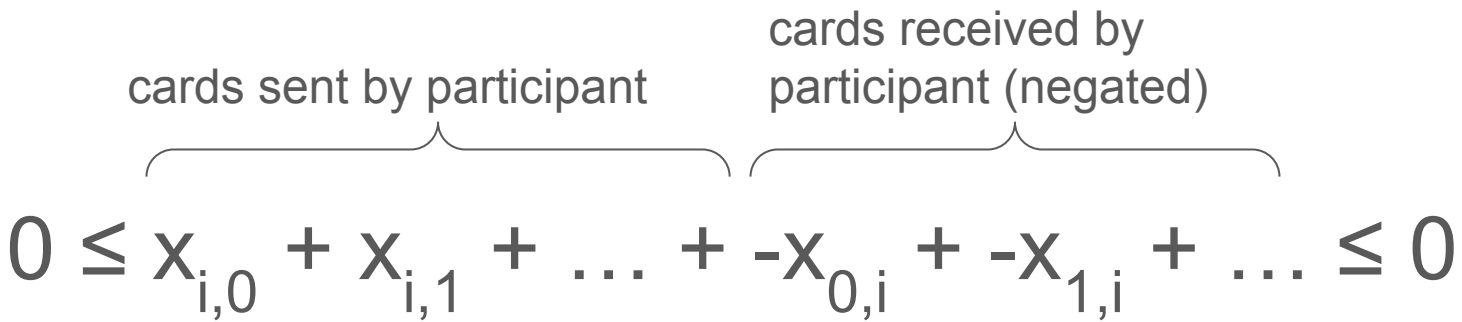
```
// Nobody sends a card to someone who sent a card to them.
for i in 0..n {
    for j in (i + 1)..n {
        model.add_cons(
            vec![&x[i][j], &x[j][i]],
            &[1., 1.],
            0.,
            1.,
            "no_mutual_exchange",
        );
    }
}
```

```
// Nobody sends a card to someone who sent a card to them.
for i in 0..n {
    for j in (i + 1)..n {
        model.add_cons(
            vec![&x[i][j], &x[j][i]],
            &[1., 1.],
            0.,
            1.,
            "no_mutual_exchange",
        );
    }
}
```

$$x_{i,j} * 1 + x_{j,i} * 1 \leq 1$$

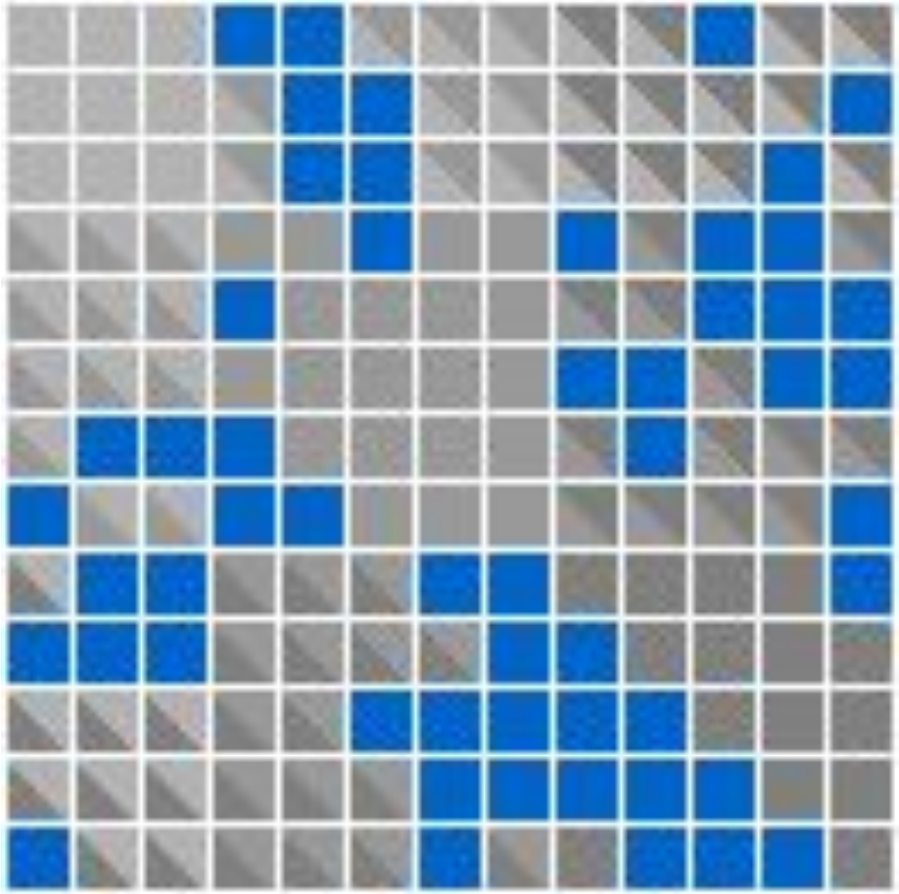
```rust
// Everyone receives a card for every card they send.
for i in 0..n {
    // Collect variables representing cards that i sends, and give them
    // a coefficient of +1.
    let mut vars: Vec<_> = x[i].iter().cloned().collect();
    let mut coefs = vec![1.0; n];
    // Collect variables representing cards that i receives, and give them
    // a coefficient of -1.
    vars.extend(x.iter().map(|row| row[i].clone()));
    coefs.extend_from_slice(vec![-1.0; n].as_ref());
    model.add_cons(vars.iter().collect(), &coefs, 0., 0., "card_balance");
}
```

```rust
// Everyone receives a card for every card they send.
for i in 0..n {
    // Collect variables representing cards that i sends, and give them
    // a coefficient of +1.
    let mut vars: Vec<_> = x[i].iter().cloned().collect();
    let mut coefs = vec![1.0; n];
    // Collect variables representing cards that i receives, and give them
    // a coefficient of -1.
    vars.extend(x.iter().map(|row| row[i].clone()));
    coefs.extend_from_slice(vec![-1.0; n].as_ref());
    model.add_cons(vars.iter().collect(), &coefs, 0., 0., "card_balance");
}
```

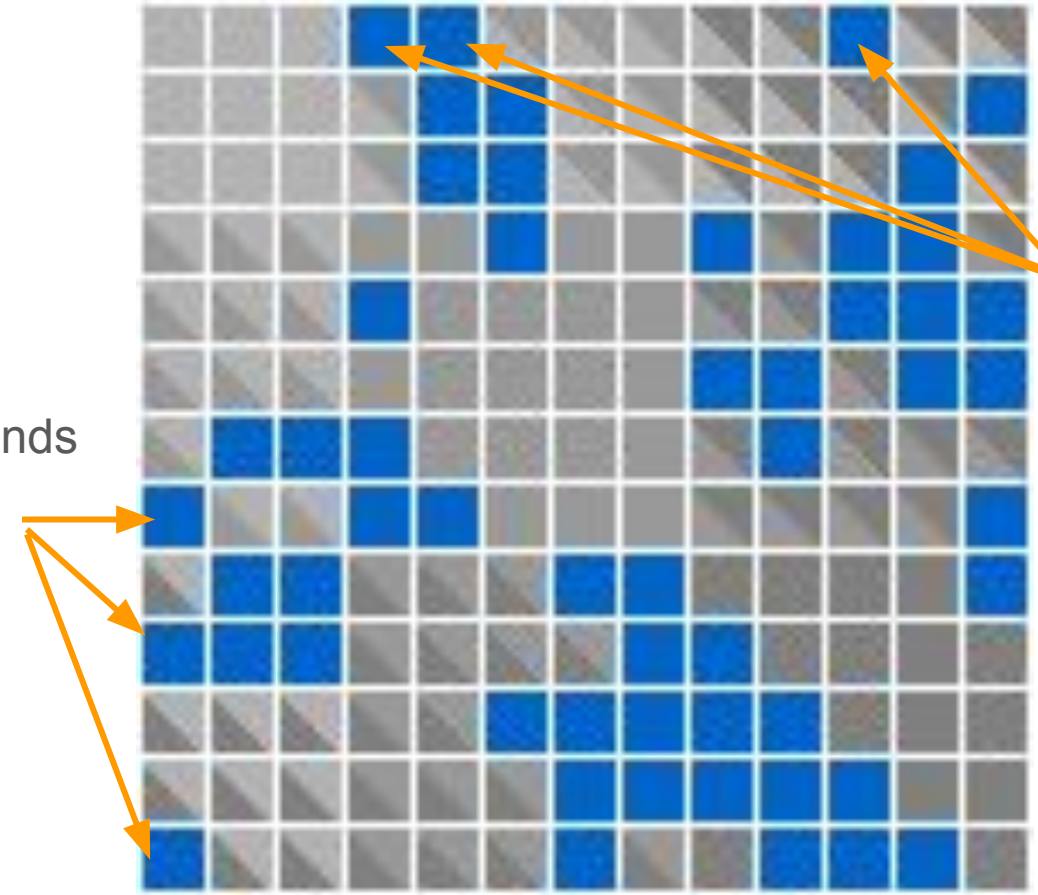


```rust
let solved_model = model.solve();
let sol = solved_model.best_sol().unwrap();

let mut result: Vec<(usize, usize)> = Vec::new();
for i in 0..n {
    for j in 0..n {
        if sol.val(&x[i][j]) >= 0.9 {
            result.push((i, j));
        }
    }
}
```

Solution for
3x3 4x5 5x5



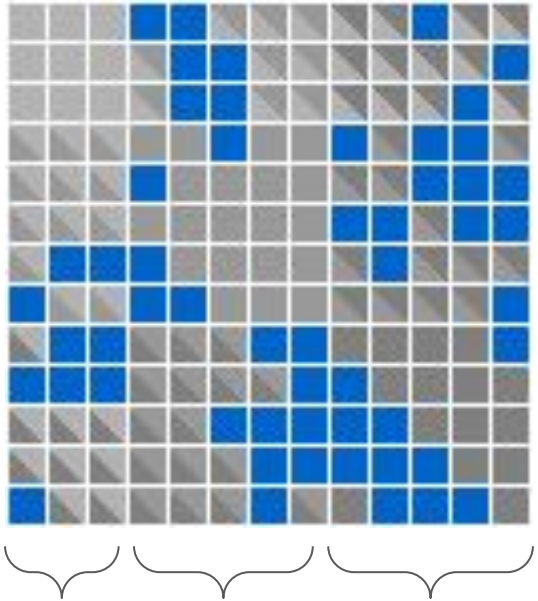
Solution for
3x3 4x5 5x5



Participant 0 receives
from participants
3, 4, 10

Participant 0 sends
to participants
7, 9, 12

Solution for
3x3 4x5 5x5



3 participants receive 3

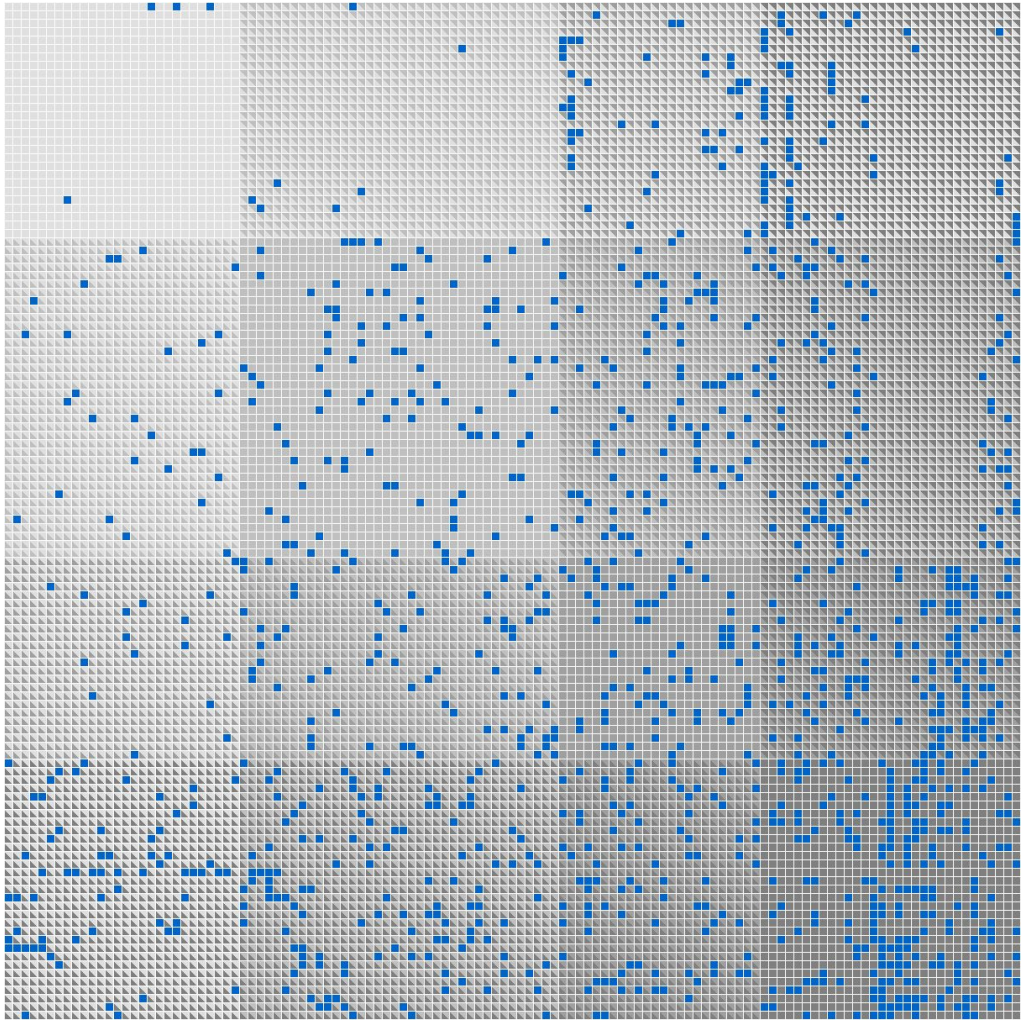
5 participants receive 4

5 participants receive 5

3 participants send 3

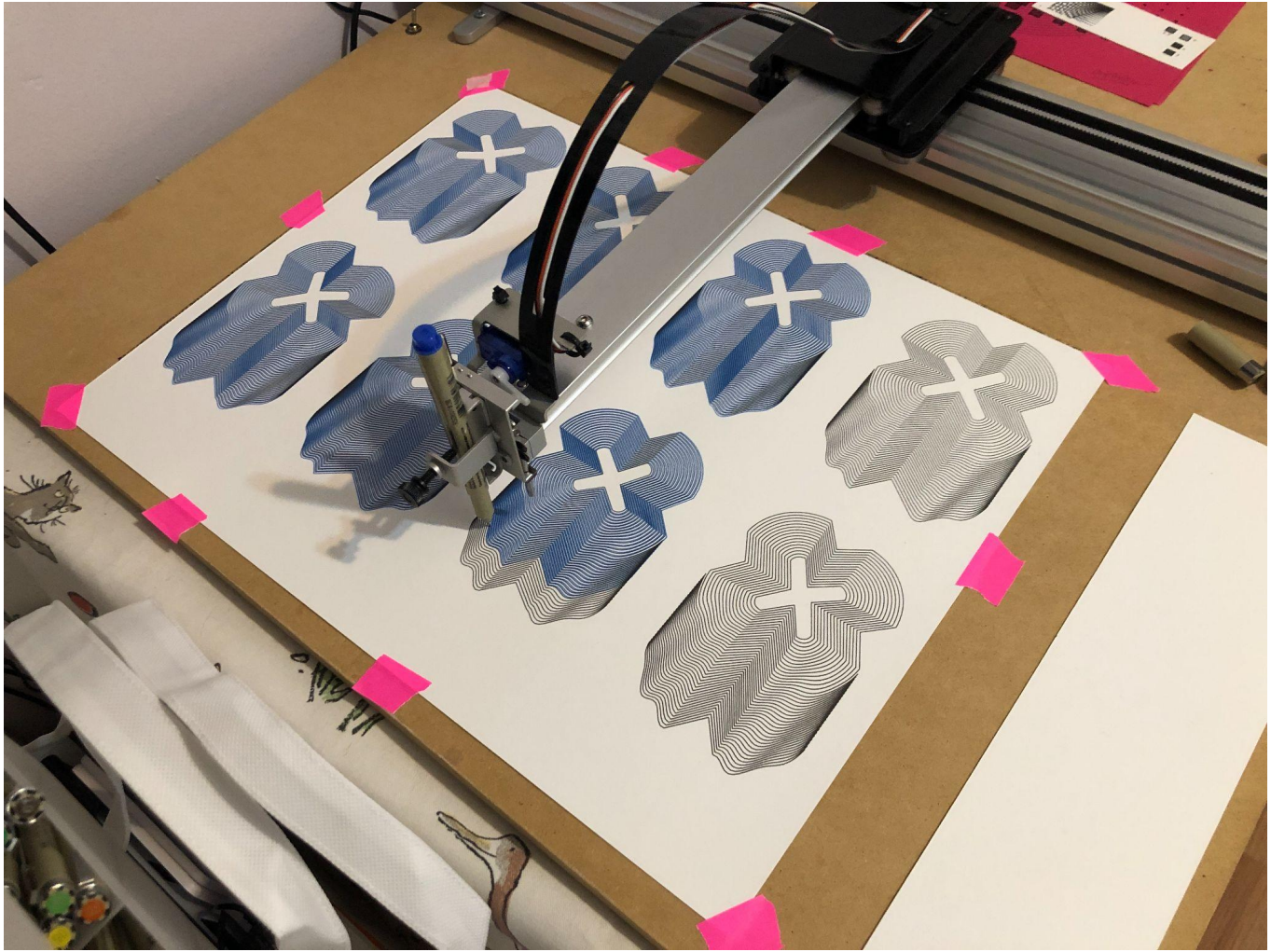
5 participants send 4

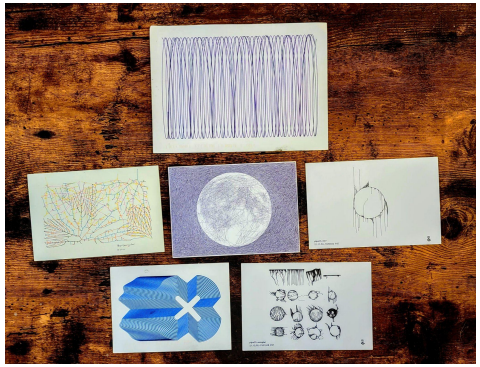
5 participants send 5

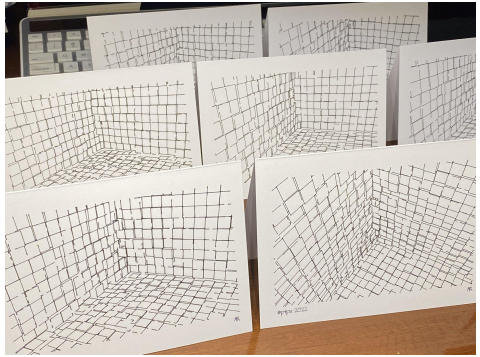


```
real    0m1.841s
user    0m1.736s
sys     0m0.041s
```

```
Total number of participants: 121
Total number of pairings: 1200
```





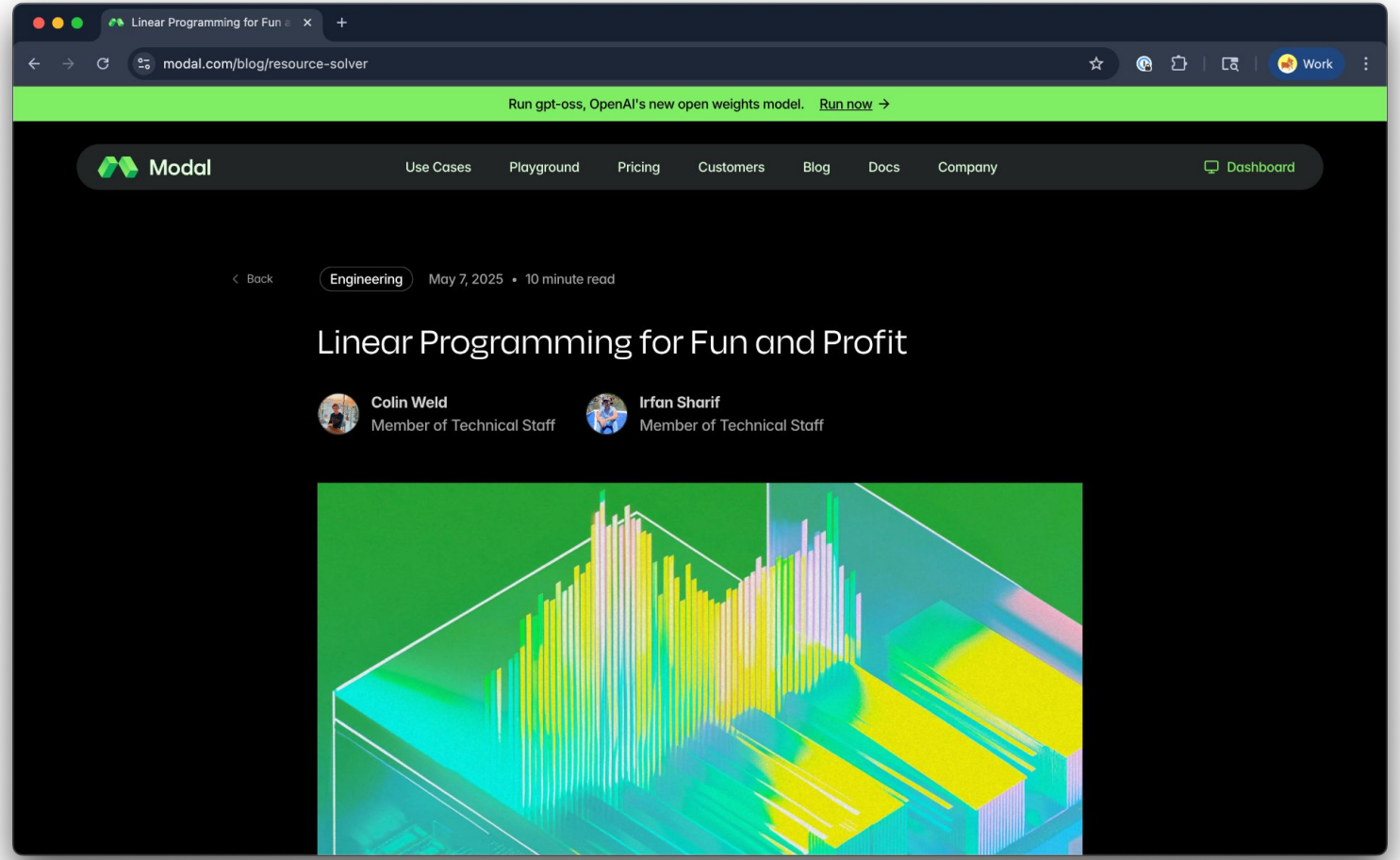


V · T · E	Mathematical optimization software	[hide]
Data formats	Mathematica · MPS · nl · sol	
Modeling tools	AIMMS · AMPL · APMonitor · ECLiPSe-CLP · Gekko · GAMS · GNU MathProg · JuMP · LINDO · OPL · Mathematica · MiniZinc · OptimJ · PuLP · Pyomo · TOMLAB · Xpress Mosel · ZIMPL · CasADi	
Solvers	**LP, MILP***	APOPT* · ANTIGONE* · Artelys Knitro* · BCP* · CLP · CBC* · CPLEX* · FortMP* · GCG* · GLOP* · GLPK/GLPSOL* · Gurobi Optimizer* · HiGHS* · LINDO* · Lp_solve · LOQO · Mathematica · MINOS · MINTO* · MOSEK* · NAG · SCIP* · SoPlex · Octeract Engine* · SYMPHONY* · Xpress Optimizer*
	QP, MIQP*	APOPT* · ANTIGONE* · Artelys Knitro* · CBC* · CLP · CPLEX* · FortMP* · HiGHS · Gurobi Optimizer* · IPOPT · LINDO* · Mathematica · MINOS · MOSEK* · NAG · Octeract Engine* · SCIP* · Xpress Optimizer*
	QCP, MIQCP*	APOPT* · ANTIGONE* · Artelys Knitro* · CPLEX* · Gurobi Optimizer* · IPOPT · LINDO* · Mathematica · MINOS · MOSEK* · NAG · SCIP* · Octeract Engine* · Xpress Optimizer* · Xpress NonLinear*
	SOCP, MISOCP*	Artelys Knitro* · CPLEX* · Gurobi Optimizer* · LINDO* · LOQO · Mathematica · MOSEK* · NAG · SCIP* · Xpress Optimizer*
	SDP, MISDP*	Mathematica · MOSEK · NAG
	NLP, MINLP*	AOA* · APOPT* · ANTIGONE* · Artelys Knitro* · BARON* · Couenne* · Galahad library · Gurobi Optimizer* · IPOPT · LINDO* · LOQO · MIDACO* · MINOS · NAG · NLPQLP · NPSOL · SCIP* · SNOPT* · Octeract Engine* · WORHP · Xpress NonLinear*
	GO	ANTIGONE* · BARON · Couenne* · Xpress Global · Mathematica · LINDO · SCIP · Octeract Engine
	CP	Artelys Kalis · Comet · CPLEX CP Optimizer · Gecode · Mathematica · JaCoP · Xpress Kalis
List of optimization software · Comparison of optimization software		

https://www.youtube.com/watch?v=njISXhlc1Rw

PARTIAL LIST OF 2019 RULES & CONSTRAINTS

Teams playing in London home or close the week prior, on BYE (or home?) the week after
Two teams must play Thursday of Week 14 after playing Thanksgiving Day
No team plays more than two road games against team coming off their BYE
No team plays consecutive road games involving cross-country trips unless requested
All teams playing road Thursday games are home the previous week
All teams playing home Thursday games have limited travel the previous week
Teams can travel no more than 2 time zones for a Thursday game
No team has earliest BYE in consecutive seasons
Manage 3-game road trips (and 3-game home stands for teams with ticket issues)
Minimize number of teams playing road game after road MNF
Minimize number of division series that end in first half of season
Minimize number of division series that are played less than 3 weeks apart
Minimize number of teams that play two consecutive road games to start or end season
Maximize number of late-season division games
Minimize early-season 1pm games for teams with weather concerns
CBS/FOX have at least three 1pm games every week, preferably geographically diverse
CBS/FOX have at least five total games, preferably six or more when doubleheader network
CBS and FOX shouldn't "lose" their key teams or an entire division all on same weekend



https://modal.com/blog/resource-solver

Thanks!

paul@modal.com

github.com/paulgb/scip-talk

modal.com/careers