Swap Design Using Constraint Programming

DOSSEH AMECK GUY-MAX DESIRE

2025-03-10

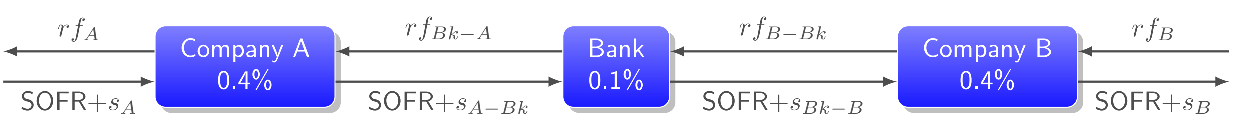
Interest Rate Swap (IRS) on Loans

Companies A and B have been offered the following rates per annum on a $20 million 5-year loan:

	Fixed Rate	Floating Rate
Company A	5%	SOFR+0.1%
Company B	6.4%	SOFR+0.6%

Company A requires a floating-rate loan; company B requires a fixed-rate loan. Design a swap that will net a bank, acting as intermediary, 0.1% per annum and that will appear equally attractive to both companies.

- Bank gain: 0.1%
- A: floating rate loan
- B: fixed rate loan



Let be:

- Known parameters:
  - r_{f_A}: the fixed interest rate of a loan for Company A;
  - s_A: the spread of credit in floating rate for company A;
  - s_B: the spread of credit in floating rate for company B;
  - G_{bank}: Bank profit
  - G_A: Company A profit
  - G_B: Company B profit
- Variables:
  - $r_{f_{Bk-A}}$: the fixed interest rate the bank agrees to pay to company A;
  - s_{A-Bk} the spread of credit company A gives to the bank on floating rate;
  - $r_{f_{B-Bk}}$: the fixed interest rate company B pays to the bank;
  - s_{Bk-B}: the spread of credit the bank pays to company B on floating rate;

Let's define a constraint satisfiability problem:

$$\begin{cases} (r_{f_B} - r_{f_{B-Bk}}) + (S_{Bk-B} - s_B) = G_B \\ (r_{f_{Bk-A}} - r_{f_A}) + (s_A - s_{A-Bk}) = G_A \\ (r_{f_{B-Bk}} - r_{f_{Bk-A}}) + (s_{A-Bk} - s_{Bk-B}) = G_{Bank} \\ r_{f_{Bk-A}} \leq r_{f_{B-Bk}} \\ s_{Bk-B} \leq s_{A-Bk} \\ r_{f_{Bk-A}} \geq r_{f_A} \\ r_{f_{B-Bk}} \leq r_{f_B} \\ s_A \geq s_{A-Bk} \\ s_{Bk-B} \leq s_B \end{cases}$$

On 10 scale basis:

$$\begin{cases} 10(r_{f_B} - s_B) + (S_{Bk-B} - r_{f_{B-Bk}}) = 10G_B \\ (r_{f_{Bk-A}} - s_{A-Bk}) + 10(s_A - r_{f_A}) = 10G_A \\ (r_{f_{B-Bk}} - r_{f_{Bk-A}}) + (s_{A-Bk} - s_{Bk-B}) = 10G_{Bank} \\ r_{f_{Bk-A}} \leq r_{f_{B-Bk}} \\ s_{Bk-B} \leq s_{A-Bk} \\ r_{f_{Bk-A}} \geq 10r_{f_A} \\ r_{f_{B-Bk}} \leq 10r_{f_B} \\ 10s_A \geq s_{A-Bk} \\ s_{Bk-B} \leq 10s_B \end{cases}$$

Finding solutions

```python
class VarArraySolutionPrinter(cp_model.CpSolverSolutionCallback):
    """Print intermediate solutions."""

    def __init__(self, variables, max_dec_digits):
        cp_model.CpSolverSolutionCallback.__init__(self)
        self.__variables = variables
        self.max_dec_digits = max_dec_digits
        self.__solution_count = 0
        self.solutions = []


    def on_solution_callback(self):
        self.__solution_count += 1
        # for v in self.__variables:
        #     print(f"{v}={self.Value(v)/self.max_dec_digits:.2f}", end=" ")
        # print()
        self.solutions.append({v.Name(): self.Value(v)/self.max_dec_digits for v in self.__variables})

    @property
    def solution_count(self):
        return self.__solution_count
```

```
def search_for_all_solutions_sample_sat( rf_A, rf_B, s_A, s_B, G_B, G_A, G_bank):
    """Showcases calling the solver to search for all solutions."""
    # Creates the model.
    model = cp_model.CpModel()
    max_dec_digits = 10**max([len(x) for x in [str(rate).split(".")[-1] for rate in [rf_A, rf_B, s_A, s

    rates_upper_bound = int(max_dec_digits*max(rf_A, rf_B))
    # Define the variables
    r_f_B_Bk = model.NewIntVar(0, rates_upper_bound, 'r_f_B_Bk')  # Upper bound 64.0
    s_Bk_B = model.NewIntVar(-int(max_dec_digits), int(max_dec_digits), 's_Bk_B')       # Upper bound 6
    r_f_Bk_A = model.NewIntVar(0, rates_upper_bound, 'r_f_Bk_A')  # Upper bound 64.0
    s_A_Bk = model.NewIntVar(-int(max_dec_digits), int(max_dec_digits), 's_A_Bk')       # Upper bound 1

    # Define the constraints
    model.Add(int(max_dec_digits*(rf_B-s_B)) - r_f_B_Bk + s_Bk_B == int(max_dec_digits*G_B))  # 58 - r_
    model.Add(r_f_Bk_A - s_A_Bk + int(max_dec_digits*(s_A-rf_A))== int(max_dec_digits*G_A))  # r_f_Bk_A
    model.Add(r_f_B_Bk - r_f_Bk_A + s_A_Bk - s_Bk_B == int(max_dec_digits*G_bank))  # r_f_B_Bk - r_f_Bk
    # model.Add(r_f_Bk_A <= r_f_B_Bk)
    # model.Add(s_Bk_B <= s_A_Bk)
    model.Add(r_f_Bk_A >= int(max_dec_digits*rf_A))
    model.Add(r_f_B_Bk <= int(max_dec_digits*rf_B))
    model.Add(s_A_Bk <= int(max_dec_digits*s_A))
    model.Add(s_Bk_B <= int(max_dec_digits*s_B))

    # Create a solver and solve.
    solver = cp_model.CpSolver()
    solution_printer = VarArraySolutionPrinter([r_f_B_Bk, s_Bk_B, r_f_Bk_A, s_A_Bk], max_dec_digits)
    # Enumerate all solutions.
    solver.parameters.enumerate_all_solutions = True
    # Solve.
    status = solver.Solve(model, solution_printer)

    print(f"Status = {solver.StatusName(status)}")
    print(f"Number of solutions found: {solution_printer.solution_count}")
    return solution_printer.solutions


solutions = search_for_all_solutions_sample_sat(5, 6.4, 0.1, 0.6, 0.4, 0.4, 0.1)
```

```
## Status = OPTIMAL
## Number of solutions found: 85
```

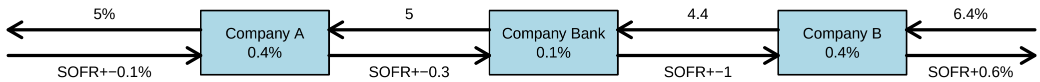
```
pd.DataFrame(solutions).to_csv("solutions.csv", index=False)
pd.DataFrame(solutions)
```

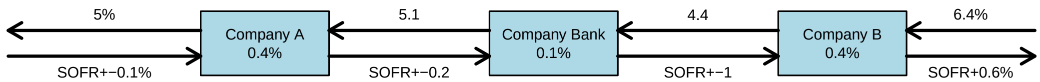
```
##     r_f_B_Bk  s_Bk_B  r_f_Bk_A  s_A_Bk
## 0        4.4    -1.0       5.0    -0.3
## 1        4.4    -1.0       5.1    -0.2
## 2        4.4    -1.0       5.2    -0.1
## 3        4.4    -1.0       5.3     0.0
## 4        4.4    -1.0       5.4     0.1
## ..       ...     ...       ...     ...
```

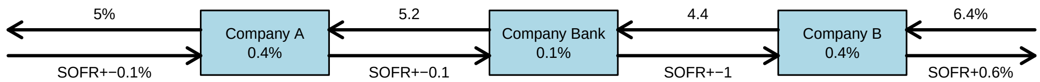
```
## 80       4.9    -0.5       5.0    -0.3
## 81       4.8    -0.6       5.0    -0.3
## 82       4.7    -0.7       5.0    -0.3
## 83       4.6    -0.8       5.0    -0.3
## 84       4.5    -0.9       5.0    -0.3
##
## [85 rows x 4 columns]
```

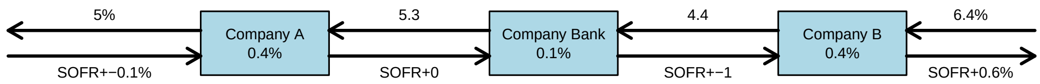
Drawing the solutions

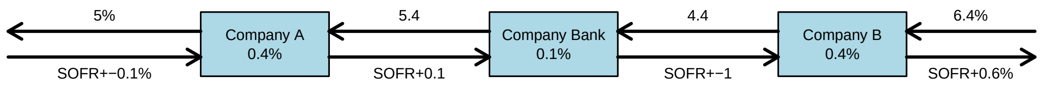
- **Drawing some of the solutions**








5%
Company A
0.4%
SOFR+−0.1%
5.3
Company Bank
0.1%
SOFR+0
4.4
Company B
0.4%
SOFR+−1
6.4%
SOFR+0.6%


5%
Company A
0.4%
SOFR+−0.1%
5.4
Company Bank
0.1%
SOFR+0.1
4.4
Company B
0.4%
SOFR+−1
6.4%
SOFR+0.6%

Interest Rate Swap (IRS) on Investments

Companies X and Y have been offered the following rates per annum on a $5 million 10-year investment:

	Fixed Rate	Floating Rate
Company X	8.0%	LIBOR
Company Y	8.8%	LIBOR

Company X requires a fixed-rate investment; company Y requires a floating-rate investment. Design a swap that will net a bank, acting as intermediary, 0.2% per annum and will appear equally attractive to X and Y.

- Bank gain: 0.2%
- X: fixed rate investment
- Y: floating rate investment
- Equal gain for X and Y: 0.3%

Finding solutions

```
solutions = search_for_all_solutions_sample_sat(8, 8.8, 0.0, 0.0, 0.3, 0.3, 0.2)
```

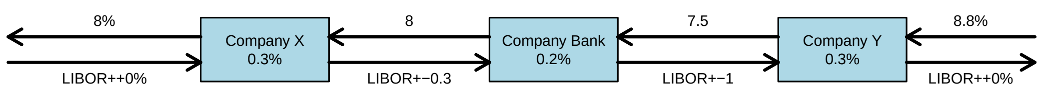
```
## Status = OPTIMAL
## Number of solutions found: 44
```

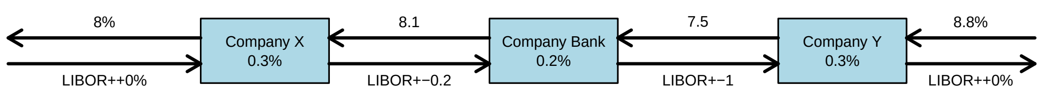
```
pd.DataFrame(solutions).to_csv("solutions.csv", index=False)
pd.DataFrame(solutions)
```

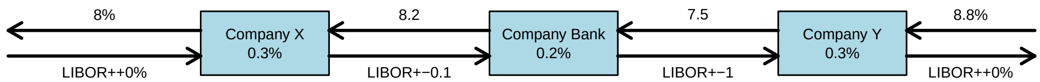
```
##     r_f_B_Bk  s_Bk_B  r_f_Bk_A  s_A_Bk
## 0        7.5    -1.0       8.0    -0.3
## 1        7.5    -1.0       8.1    -0.2
## 2        7.5    -1.0       8.2    -0.1
## 3        7.5    -1.0       8.3     0.0
## 4        7.6    -0.9       8.3     0.0
## 5        7.6    -0.9       8.2    -0.1
## 6        7.6    -0.9       8.1    -0.2
## 7        7.7    -0.8       8.1    -0.2
## 8        7.7    -0.8       8.2    -0.1
## 9        7.7    -0.8       8.3     0.0
## 10       7.8    -0.7       8.3     0.0
## 11       7.8    -0.7       8.2    -0.1
## 12       7.8    -0.7       8.1    -0.2
## 13       7.9    -0.6       8.1    -0.2
## 14       7.9    -0.6       8.2    -0.1
## 15       7.9    -0.6       8.3     0.0
## 16       8.0    -0.5       8.3     0.0
```

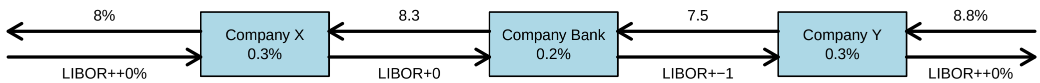
```
## 17        8.0    -0.5       8.2    -0.1
## 18        8.0    -0.5       8.1    -0.2
## 19        8.1    -0.4       8.1    -0.2
## 20        8.1    -0.4       8.2    -0.1
## 21        8.1    -0.4       8.3     0.0
## 22        8.2    -0.3       8.3     0.0
## 23        8.2    -0.3       8.2    -0.1
## 24        8.2    -0.3       8.1    -0.2
## 25        8.3    -0.2       8.1    -0.2
## 26        8.3    -0.2       8.2    -0.1
## 27        8.3    -0.2       8.3     0.0
## 28        8.4    -0.1       8.3     0.0
## 29        8.4    -0.1       8.2    -0.1
## 30        8.4    -0.1       8.1    -0.2
## 31        8.5     0.0       8.1    -0.2
## 32        8.5     0.0       8.2    -0.1
## 33        8.5     0.0       8.3     0.0
## 34        8.5     0.0       8.0    -0.3
## 35        8.4    -0.1       8.0    -0.3
## 36        8.3    -0.2       8.0    -0.3
## 37        8.2    -0.3       8.0    -0.3
## 38        8.1    -0.4       8.0    -0.3
## 39        8.0    -0.5       8.0    -0.3
## 40        7.9    -0.6       8.0    -0.3
## 41        7.8    -0.7       8.0    -0.3
## 42        7.7    -0.8       8.0    -0.3
## 43        7.6    -0.9       8.0    -0.3
```

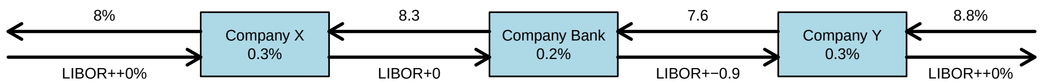
Drawing some of the solution






8%
8.2
7.5
8.8%
Company X
0.3%
Company Bank
0.2%
Company Y
0.3%
LIBOR++0%
LIBOR+−0.1
LIBOR+−1
LIBOR++0%


8%
8.3
7.5
8.8%
Company X
0.3%
Company Bank
0.2%
Company Y
0.3%
LIBOR++0%
LIBOR+0
LIBOR+−1
LIBOR++0%


8%
8.3
7.6
8.8%
Company X
0.3%
Company Bank
0.2%
Company Y
0.3%
LIBOR++0%
LIBOR+0
LIBOR+−0.9
LIBOR++0%

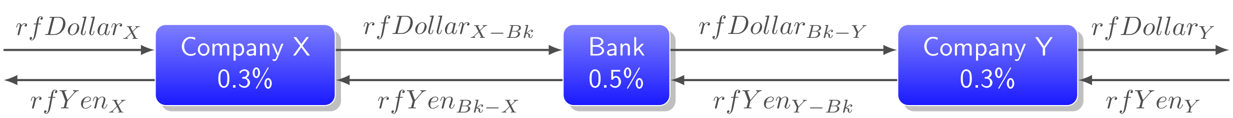
Fixed for fixed Currency Swap

Company X wishes to borrow U.S. dollars at a fixed rate of interest. Company Y wishes to borrow Japanese yen at a fixed rate of interest. The amounts required by the two companies are roughly the same at the current exchange rate. The companies have been quoted the following interest rates, which have been adjusted for the impact of taxes:

	Yen	Dollars
Company X	5.0%	9.6%
Company Y	6.5%	10.0%

Design a swap that will net a bank, acting as intermediary, 50 basis points per annum. Make the swap equally attractive to the two companies and ensure that all foreign exchange risk is assumed by the bank.

- Bank gain: 0.5%
- X: fixed rate loan in dollars
- Y: fixed rate loan in yen
- Equal gain for X and Y: ((6.5-5)-(10-9.6)-0.5)/2=0.3%



Finding solutions

```python
class VarArraySolutionPrinter(cp_model.CpSolverSolutionCallback):
    """Print intermediate solutions."""

    def __init__(self, variables, max_dec_digits):
        cp_model.CpSolverSolutionCallback.__init__(self)
        self.__variables = variables
        self.max_dec_digits = max_dec_digits
        self.__solution_count = 0
        self.solutions = []


    def on_solution_callback(self):
        self.__solution_count += 1
        # for v in self.__variables:
        #     print(f"{v}={self.Value(v)/self.max_dec_digits:.2f}", end=" ")
        # print()
        self.solutions.append({v.Name(): self.Value(v)/self.max_dec_digits for v in self.__variables})

    @property
    def solution_count(self):
        return self.__solution_count


def search_for_all_solutions_sample_sat( rfDollar_A, rfDollar_B, rfYen_A, rfYen_B, G_B, G_A, G_bank):
```

```
    """Showcases calling the solver to search for all solutions."""
    # Creates the model.
    model = cp_model.CpModel()
    max_dec_digits = 10**max([len(x) for x in [str(rate).split(".")[-1] for rate in [rfDollar_A, rfDolla

    rates_upper_bound = int(max_dec_digits*max(rfDollar_A, rfDollar_B, rfYen_A, rfYen_B))

    # Define the variables
    r_f_Dollar_A_Bk = model.NewIntVar(0, rates_upper_bound, 'r_f_Dollar_A_Bk')
    r_f_Yen_Bk_A = model.NewIntVar(0, rates_upper_bound, 'r_f_Yen_Bk_A')
    r_f_Dollar_Bk_B = model.NewIntVar(0, rates_upper_bound, 'r_f_Dollar_Bk_B')
    r_f_Yen_B_Bk = model.NewIntVar(0, rates_upper_bound, 'r_f_Yen_B_Bk')

    # Define the constraints
    model.Add(int(max_dec_digits*(rfDollar_B)) - r_f_Dollar_Bk_B - r_f_Yen_B_Bk+int(max_dec_digits*rfYe
    model.Add(int(max_dec_digits*rfDollar_A) - r_f_Dollar_A_Bk  - int(max_dec_digits*(rfYen_A))+ r_f_Yer
    model.Add(r_f_Dollar_A_Bk - r_f_Dollar_Bk_B + r_f_Yen_B_Bk - r_f_Yen_Bk_A == int(max_dec_digits*G_ba
    model.Add(r_f_Dollar_Bk_B >= r_f_Dollar_A_Bk)
    model.Add(r_f_Yen_Bk_A <= r_f_Yen_B_Bk)
    model.Add(r_f_Dollar_A_Bk <= int(max_dec_digits*rfDollar_A))
    model.Add(r_f_Dollar_Bk_B <= int(max_dec_digits*rfDollar_B))
    model.Add(r_f_Yen_B_Bk <= int(max_dec_digits*rfYen_B))
    model.Add(r_f_Yen_Bk_A <= int(max_dec_digits*rfYen_A))

    solver = cp_model.CpSolver()
    solution_printer = VarArraySolutionPrinter([r_f_Dollar_A_Bk, r_f_Yen_Bk_A, r_f_Dollar_Bk_B, r_f_Yen_
    # Enumerate all solutions.
    solver.parameters.enumerate_all_solutions = True
    # Solve.
    status = solver.Solve(model, solution_printer)

    print(f"Status = {solver.StatusName(status)}")
    print(f"Number of solutions found: {solution_printer.solution_count}")
    return solution_printer.solutions


solutions = search_for_all_solutions_sample_sat(9.6, 10, 5, 6.5, 0.3, 0.3, 0.5)
```

```
## Status = OPTIMAL
## Number of solutions found: 501
```

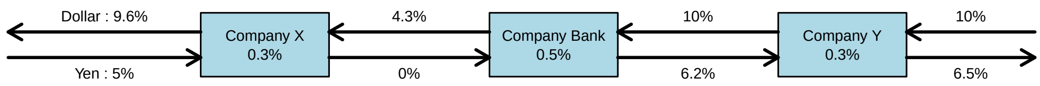
```
pd.DataFrame(solutions).to_csv("solutions.csv", index=False)
pd.DataFrame(solutions)
```

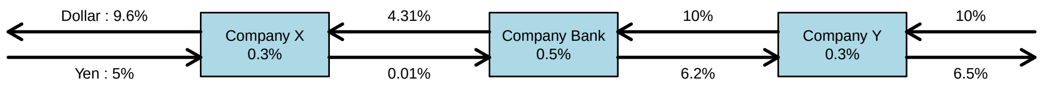
```
##      r_f_Dollar_A_Bk  r_f_Yen_Bk_A  r_f_Dollar_Bk_B  r_f_Yen_B_Bk
## 0               4.30          0.00             10.0           6.2
## 1               4.31          0.01             10.0           6.2
## 2               4.32          0.02             10.0           6.2
## 3               4.33          0.03             10.0           6.2
## 4               4.34          0.04             10.0           6.2
## ..               ...           ...              ...           ...
## 496             9.26          4.96             10.0           6.2
```

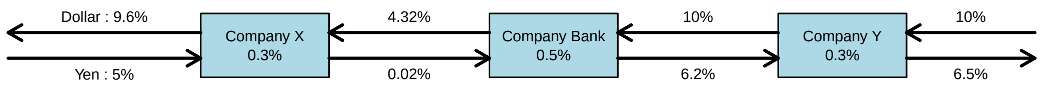
```
## 497                9.27          4.97            10.0           6.2
## 498                9.28          4.98            10.0           6.2
## 499                9.29          4.99            10.0           6.2
## 500                9.30          5.00            10.0           6.2
##
## [501 rows x 4 columns]
```

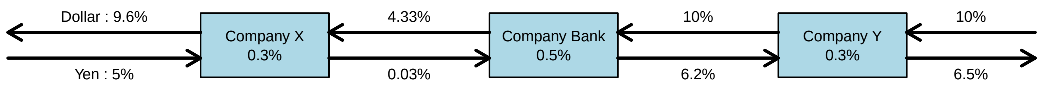
Drawing some of the solutions

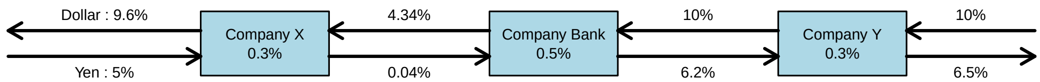
- **Drawing some of the solutions**










Dollar : 9.6%
Yen : 5%
Company X
0.3%
4.34%
0.04%
Company Bank
0.5%
10%
6.2%
Company Y
0.3%
10%
6.5%