An Implementation of

SAT-Based Two-Terminal Path Finding

Using Z3 Solver

Yihong Gu
Tsinghua University
gyh15@mails.tsinghua.edu.cn

Outlines

- Introduction
- Model
    - Flow based model
    - Point based model
- Optimization
- Evaluation and Results
- System Architecture
- Demo
- Acknowledgements

Outlines

- **Introduction**
- Model
  - Flow based model
  - Point based model
- Optimization
- Evaluation and Results
- System Architecture
- Demo
- Acknowledgements

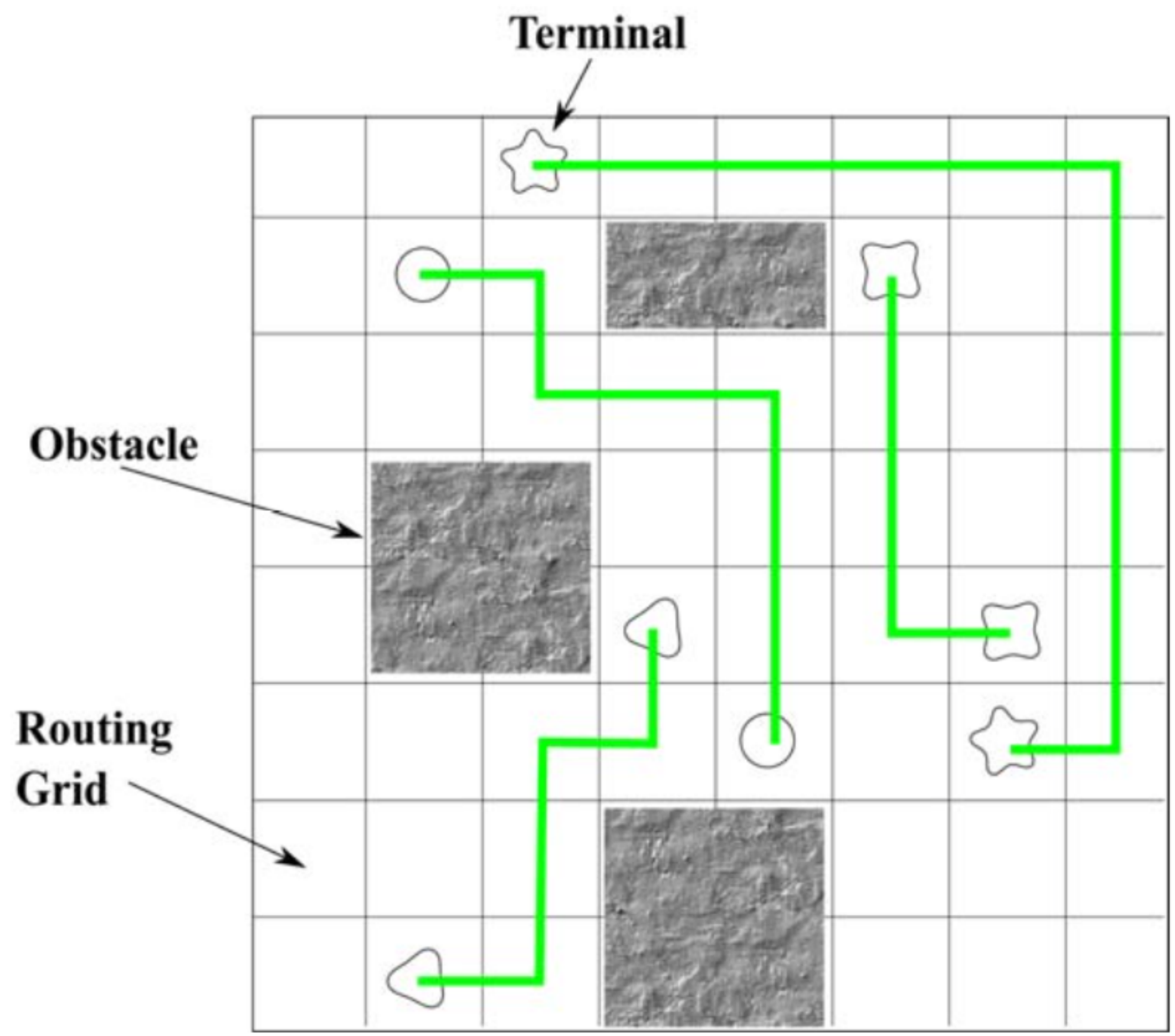
Introduction

- Problem

  given <u>N × N</u> routing grids,
  and <u>M</u> pairs of terminals
  use program to
  find routing paths
  <u>connecting each pair of terminals.</u>

- Contraints

  different paths <u>cannot cross each other</u>
  no path can cross obstacles

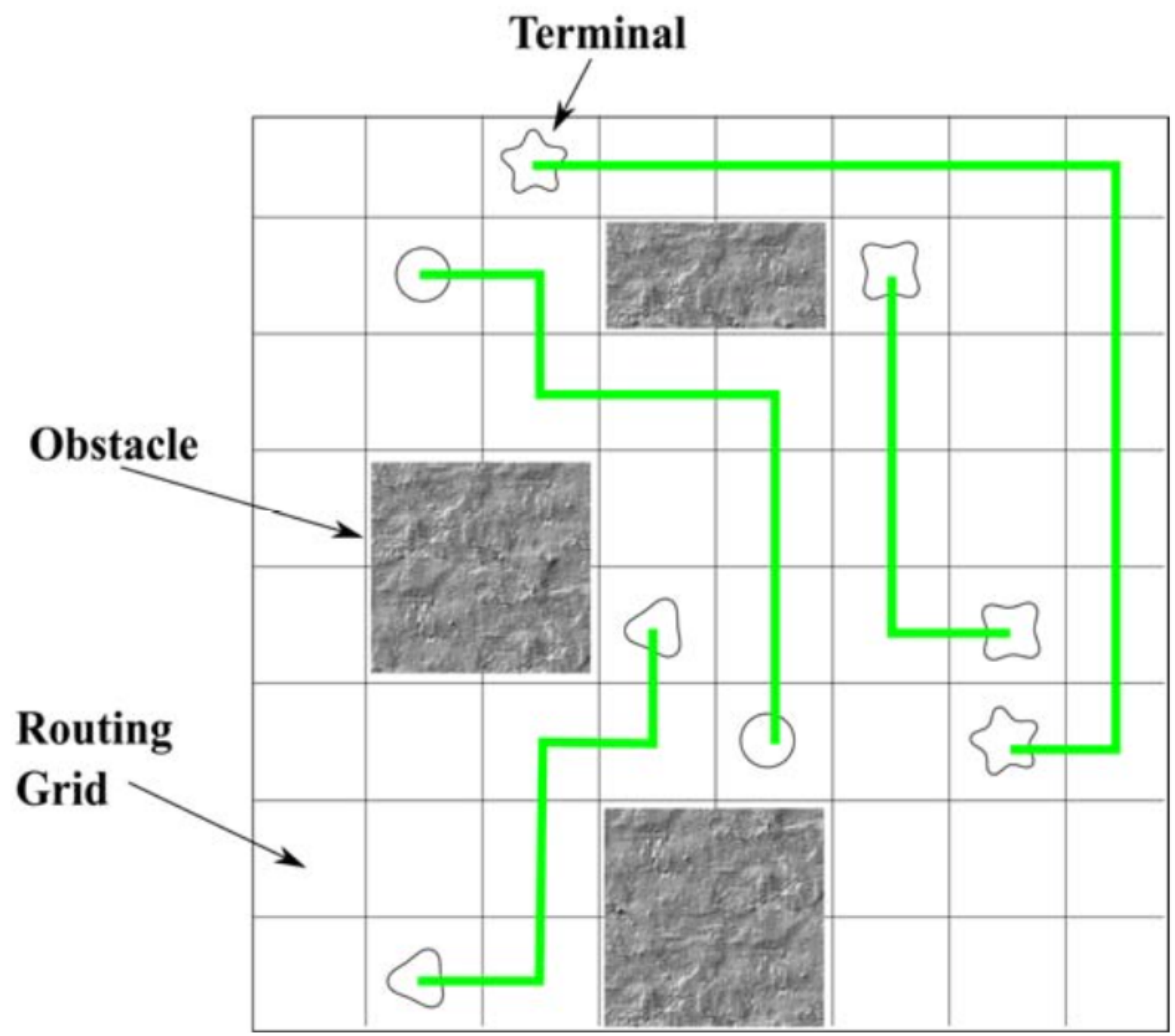


Introduction

- Object

  maximize <u>the number of connected pairs</u> when all pairs of terminals can be connected, should minimize <u>the total length of all the paths.</u>
- Stipulates

  if one grid is defined to be one terminal of pair *d*, then other pairs *d'* can not route across the grid.



Outlines

- Introduction
- **Model**
  - Flow based model
  - Point based model
- Optimization
- Evaluation and Results
- System Architecture
- Demo
- Acknowledgements

Flow Based Model

- For every droplet
  - establish a network flow model
  - edge: binary variable (SAT-Based model)
  - contraints: flow conservation
- Every grid can only be visited once
  - linear-like term
- 2*N^2*M variables, O(N^2 * M) contraints.
- Is <u>equivalent</u> to the original model

Point Based Model

- $c_{p,d}$: whether the droplet *d* visit across the position *p*
- limit terms
  - barrier/not corresponding layer: FALSE
  - one grid be visited once:

$$\sum_{d=1}^{M} [c_{p,d}] \leq 1$$

- connection terms:

$$c_{p,d} \rightarrow \left(\sum_{p' \in B(p)} [c_{p',d}] = 2 \right) \qquad \text{(free grid)}$$

$$c_{p,d} \rightarrow \left(\sum_{p' \in B(p)} [c_{p',d}] = 1 \right) \qquad \text{(terminal grid)}$$

- source and terminal terms:

$$c_{p^*_d,d} \leftrightarrow c_{p^\dagger_d,d}$$

- objective:

maxmize $\sum_{d=1}^{M} [c_{p^*_d,d}]$ minimize $\sum_{d} \sum_{p} [c_{p,d}]$

Point Based Model

- $c_{p,d}$: whether the droplet *d* visit across the position *p*
- limit terms
  - barrier/not corresponding layer: FALSE
  - one grid be visited once:

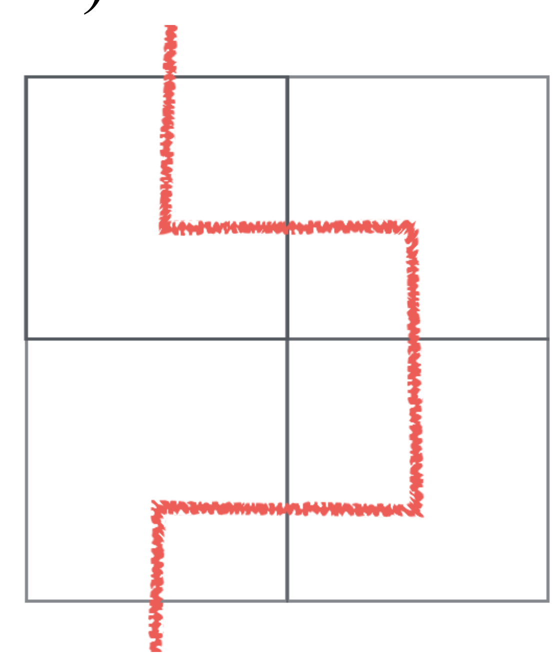
$$\sum_{d=1}^{M}[c_{p,d}] \leq 1$$

- connection terms:

$$c_{p,d} \rightarrow \left(\sum_{p' \in B(p)} [c_{p',d}] = 2 \right) \qquad \text{(free grid)}$$

Is **not equivalent** to the original model

$$c_{p,d} \rightarrow \left(\sum_{p' \in B(p)} [c_{p',d}] = 1 \right) \qquad \text{(terminal grid)}$$



- source and terminal terms:

$$c_{p_d^*,d} \leftrightarrow c_{p_d^\dagger,d}$$

- objective:

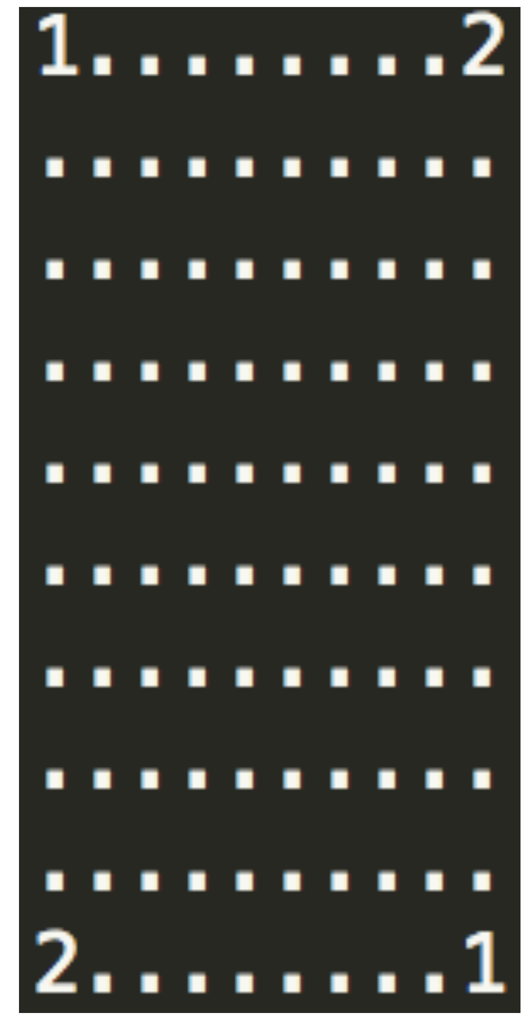
maxmize $\sum_{d=1}^{M}[c_{p_d^*,d}]$ minimize $\sum_{d}\sum_{p}[c_{p,d}]$

Outlines

- Introduction
- Model
    - Flow based model
    - Point based model
- **Optimization**
- Evaluation and Results
- System Architecture
- Demo
- Acknowledgements

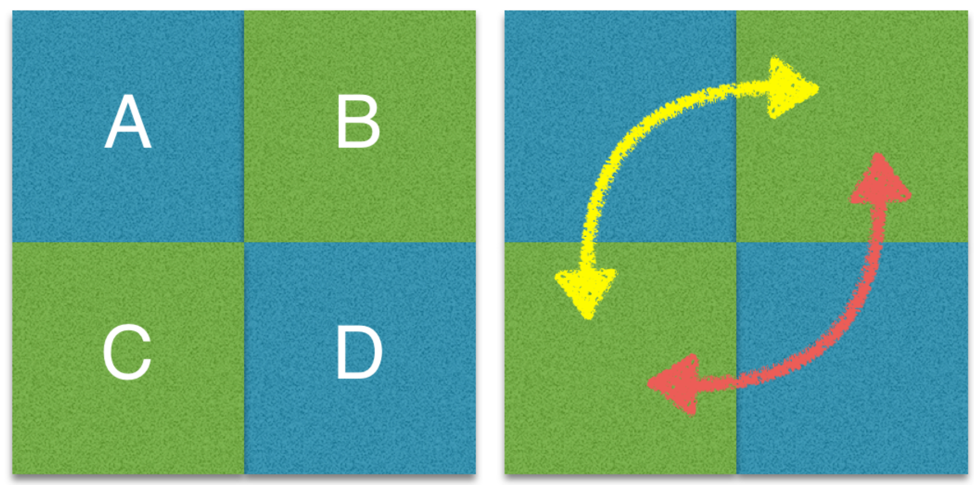
Optimization

- Problem Confronted: Too slow
- Solution:
  - Combined Terms
    Combine several constraints together (in Z3 solver)
  - Prune Terms

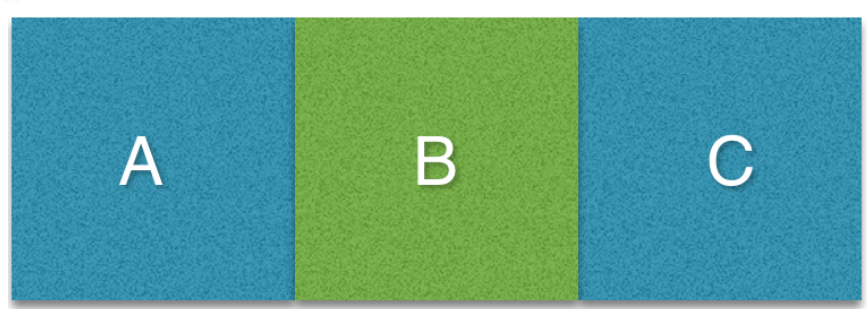


Prune Terms

- Aim: told Z3 directly how shortest path like
- Prune Term 0: $\neg(c_{p_1,d} \land c_{p_2,d} \land c_{p_3,d} \land c_{p_4,d})$



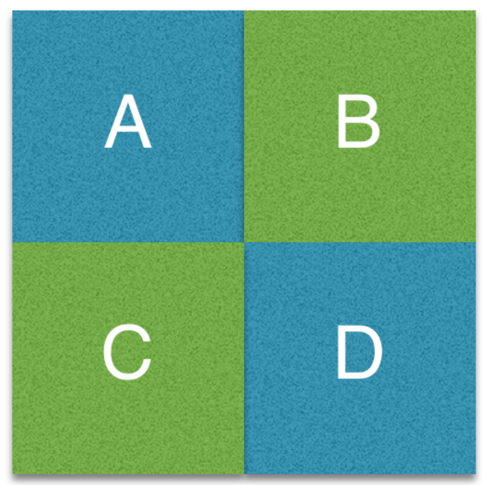
- Prune Term 1: $(\sum_{x=1}^{M} [c_{p_1,x}] = 0) \rightarrow \neg(c_{p_2,d} \land c_{p_3,d} \land c_{p_4,d})$

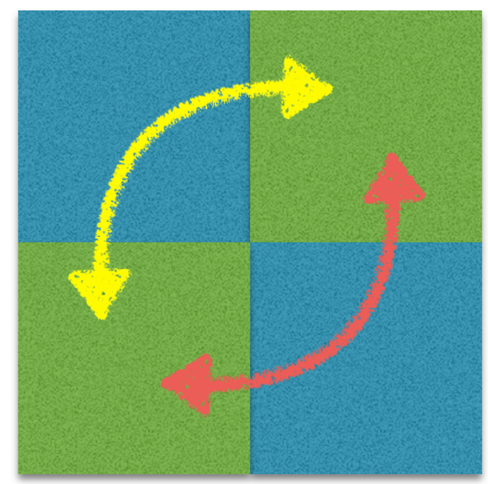


- Prune Term 2: $(\sum_{x=1}^{M} [c_{p_1,x}] = 0) \rightarrow \neg(c_{p_2,d} \land c_{p_3,d})$

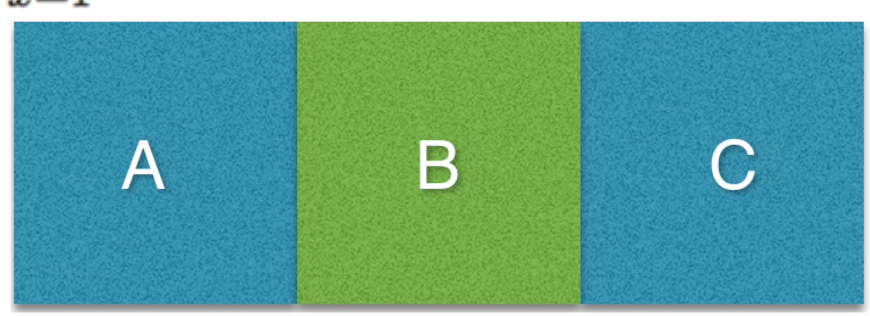
Prune Terms

- Aim: told Z3 <u>directly</u> how shortest path like
- Prune Term 0: $\neg(c_{p_1,d} \wedge c_{p_2,d} \wedge c_{p_3,d} \wedge c_{p_4,d})$





- Prune Term 1: $(\sum_{x=1}^{M} [c_{p_1,x}] = 0) \rightarrow \neg(c_{p_2,d} \wedge c_{p_3,d})$



- Prune Term 2: $(\sum_{x=1}^{M} [c_{p_1,x}] = 0) \rightarrow \neg(c_{p_2,d} \wedge c_{p_3,d})$

Outlines

- Introduction
- Model
  - Flow based model
  - Point based model
- Optimization
- **Evaluation and Results**
- System Architecture
- Demo
- Acknowledgements

Evaluation & Results

- Correctness
  - legality check
  - pair-wise check (using 2 different models)
  - N in {6,7,8}, M in {1,..,9}, proportion of obstacles 10%
  - randomly generate 30 tests each batch
- Efficiency
  - proportion of obstacles 0%, M in {1..9}, N up to 12
  - randomly generate 100 tests each batch

N	flow	flow_p	point_p	$\text{point}_p^\dagger$
6	0.07	....	....	...
7	0.09	....	....	...
8	120	0.43	0.13	0.12
9	....	1.70	0.40	0.35
10	....	8.24	1.44	1.18
11	....	49.6	4.80	4.47
12	....	349	36.0	27.3

Table 1. Mean Time

N	flow	flow_p	point_p	$\text{point}_p^\dagger$
6	0.21(2)	....(.)	....(.)	....(.)
7	3.38(3)	....(.)	....(.)	....(.)
8	482(4)	0.81(4)	0.21(4)	0.18(4)
9	....(.)	4.02(4)	0.61(5)	0.56(4)
10	....(.)	18.4(4)	2.37(6)	1.88(7)
11	....(.)	159(5)	8.30(5)	7.46(6)
12	....(.)	1125(5)	70.6(6)	50.0(6)

Table 2. Max Mean Time[1]

[1] unit: sec, the data format is X(Y), which X represent the total time(seconds) and Y in {1,…,9} means the value M which has the largest mean time.

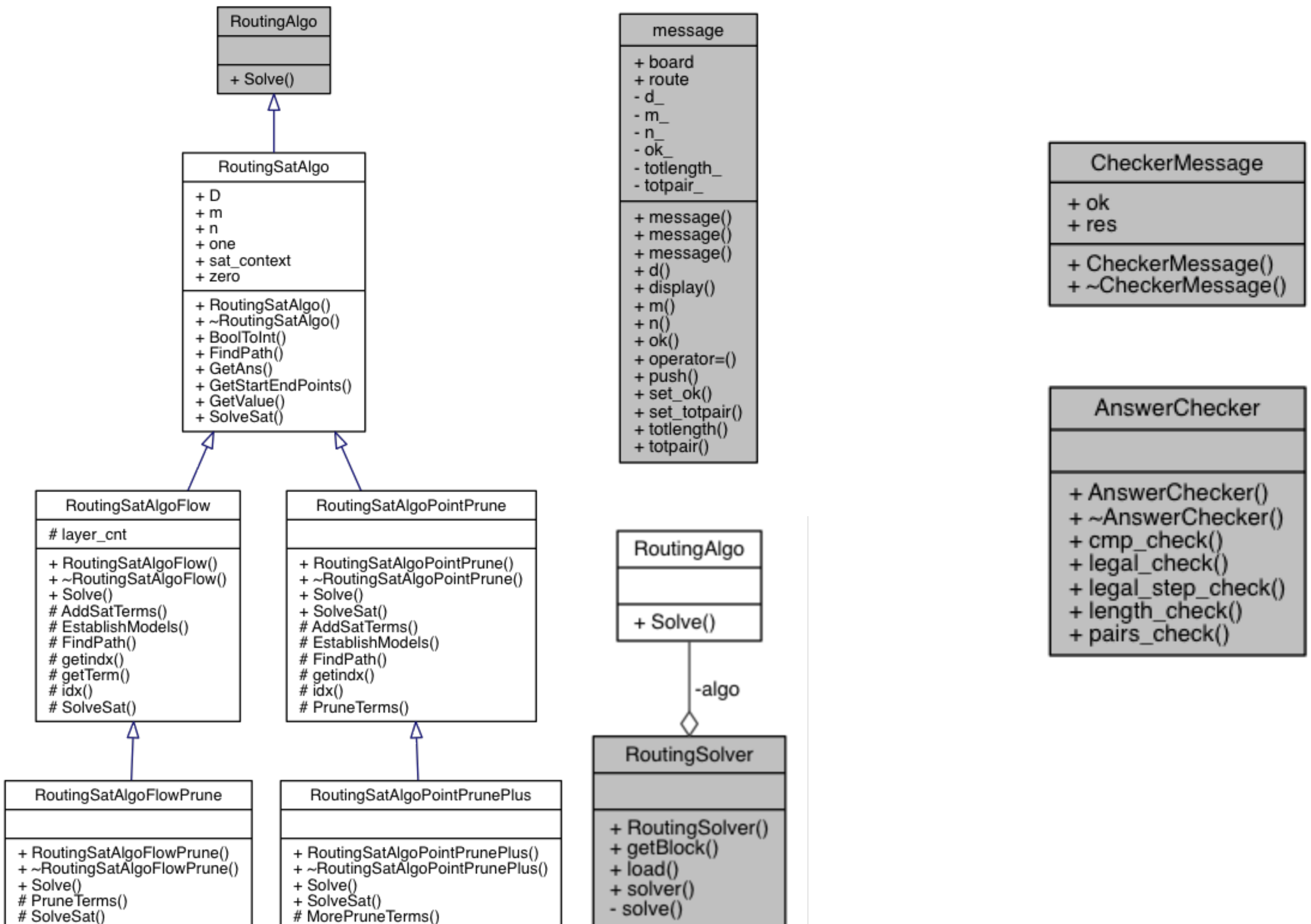
Outlines

- Introduction
- Model
  - Flow based model
  - Point based model
- Optimization
- Evaluation and Results
- **System Architecture**
- Demo
- Acknowledgements

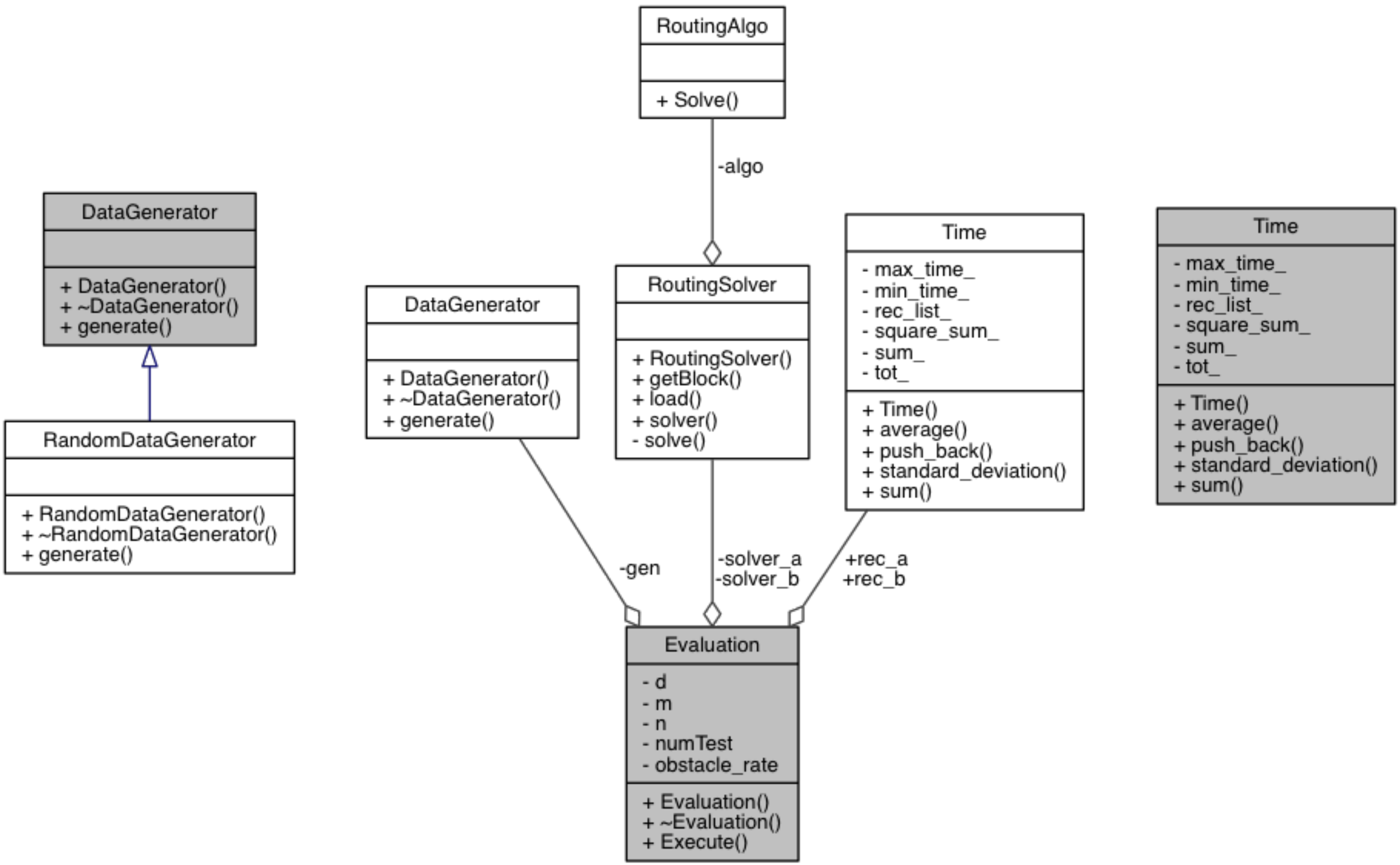
System Architecture

- Four main parts
  - **core**: the whole algorithm.
  - **checker**: check if the answer calculated by the algorithm is legal and correct.
  - **evaluation**: evaluate the efficiency and correctness of the algorithm(including data generator).
  - **demo**.

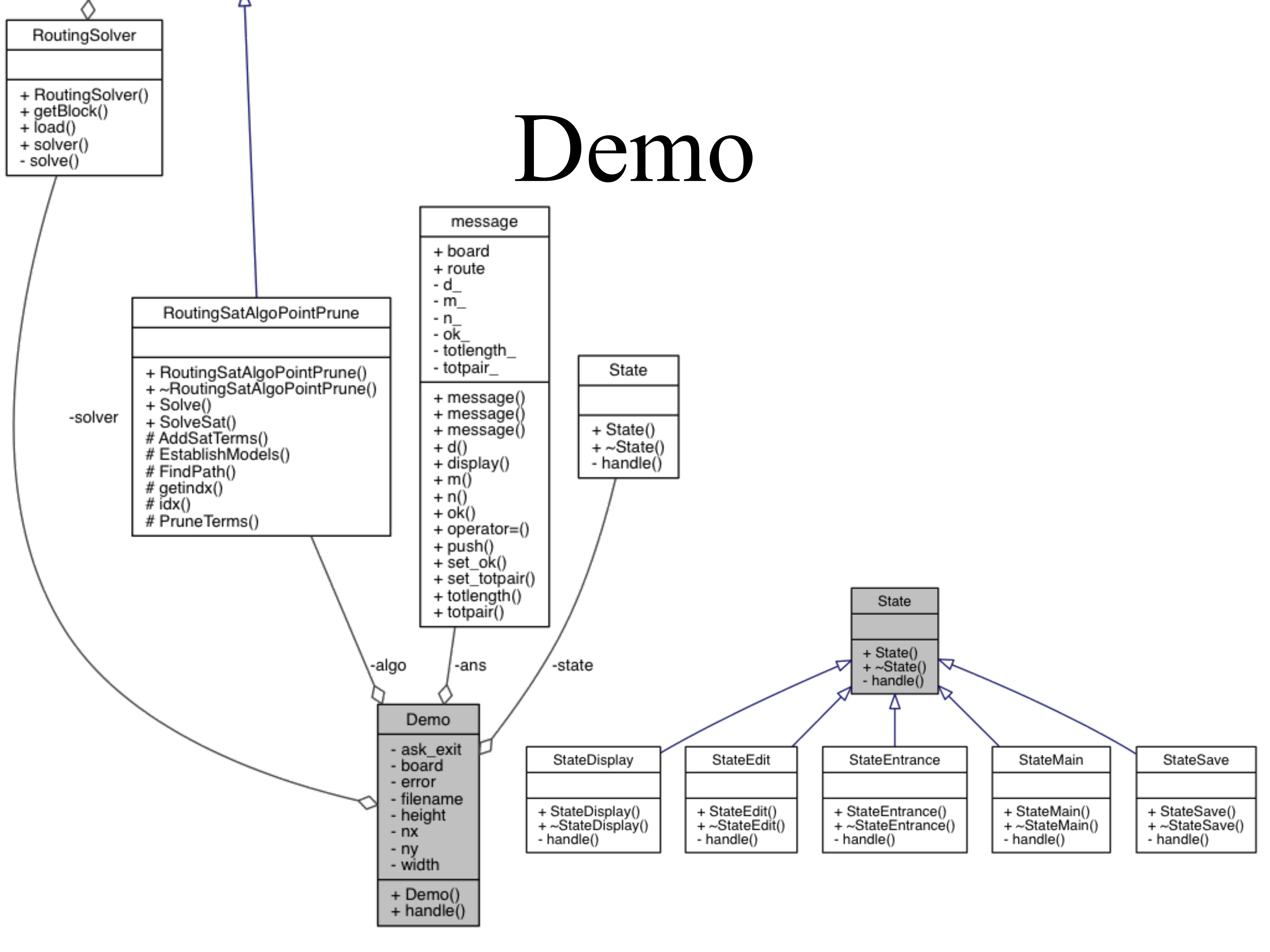
Core & Checker



Evaluation


RoutingAlgo
+ Solve()
-algo
DataGenerator
+ DataGenerator()
+ ~DataGenerator()
+ generate()
RandomDataGenerator
+ RandomDataGenerator()
+ ~RandomDataGenerator()
+ generate()
DataGenerator
+ DataGenerator()
+ ~DataGenerator()
+ generate()
RoutingSolver
+ RoutingSolver()
+ getBlock()
+ load()
+ solver()
- solve()
Time
- max_time_
- min_time_
- rec_list_
- square_sum_
- sum_
- tot_
+ Time()
+ average()
+ push_back()
+ standard_deviation()
+ sum()
Time
- max_time_
- min_time_
- rec_list_
- square_sum_
- sum_
- tot_
+ Time()
+ average()
+ push_back()
+ standard_deviation()
+ sum()
-gen
-solver_a
-solver_b
+rec_a
+rec_b
Evaluation
- d
- m
- n
- numTest
- obstacle_rate
+ Evaluation()
+ ~Evaluation()
+ Execute()

Demo


RoutingSolver
+ RoutingSolver()
+ getBlock()
+ load()
+ solver()
- solve()
-solver
RoutingSatAlgoPointPrune
+ RoutingSatAlgoPointPrune()
+ ~RoutingSatAlgoPointPrune()
+ Solve()
+ SolveSat()
# AddSatTerms()
# EstablishModels()
# FindPath()
# getindx()
# idx()
# PruneTerms()
message
+ board
+ route
- d_
- m_
- n_
- ok_
- totlength_
- totpair_
+ message()
+ message()
+ message()
+ d()
+ display()
+ m()
+ n()
+ ok()
+ operator=()
+ push()
+ set_ok()
+ set_totpair()
+ totlength()
+ totpair()
State
+ State()
+ ~State()
- handle()
-algo
-ans
-state
Demo
- ask_exit
- board
- error
- filename
- height
- nx
- ny
- width
+ Demo()
+ handle()
State
+ State()
+ ~State()
- handle()
StateDisplay
+ StateDisplay()
+ ~StateDisplay()
- handle()
StateEdit
+ StateEdit()
+ ~StateEdit()
- handle()
StateEntrance
+ StateEntrance()
+ ~StateEntrance()
- handle()
StateMain
+ StateMain()
+ ~StateMain()
- handle()
StateSave
+ StateSave()
+ ~StateSave()
- handle()

Outlines

- Introduction
- Model
  - Flow based model
  - Point based model
- Optimization
- Evaluation and Results
- System Architecture
- **Demo**
- **Acknowledgements**

Demo

Acknowledgements

- Yue Yu
  - for the prune terms
- Zihan Xu
  - for the discussion on the implementation

Thank you

Q&A[1]

[1] More details can be seen in the whitepaper in https://github.com/wmyw96/IndividualProject_OOP/blob/master/doc/whitepaper.pdf