Automation of Triangle Ruler-and-Compass Constructions Using Constraint Solvers

Milan Banković[1]

[1]Faculty of Mathematics, University of Belgrade, Serbia

September 21, 2023.
ADG 2023, Belgrade, Serbia

Overview

Introduction

About construction problems

- One of the oldest kinds of problems in geometry
  - Given some elements of a figure, find a sequence of steps to construct the remaining elements
  - Tools available: a ruler (straightedge) and a compass
- Solving by hand – interesting to geometricians
- Automated solving – a challenge to computer scientists

Introduction

Two approaches to automated solving

- Implementing a custom search algorithm in some programming language
  - required geometric knowledge must be compiled into it
  - might require a lot of effort
- Using existing artificial intelligence tools that are good in search in general
  - we may focus on modeling the geometric knowledge, and leave the search to the tool
  - we can search for best solutions (e.g. shortest), using tools that support optimization

Introduction

What are we trying to do?

- Automated solving using off-the-shelf finite domain constraint solvers
  - great in solving search and optimization problems
- Modeling based on automated planning approach
  - the solution is viewed as a sequence of actions producing a state satisfying a given goal
- Evaluate the approach on 74 solvable problems from Wernick's set
  - constructing triangles from three given points
- Compare the approach with state-of-the-art tools
  - ArgoTriCS[1] dedicated triangle construction solver

[1] Vesna Marinković. ArgoTriCS – automated triangle construction solver. Journal of Experimental & Theoretical Artificial Intelligence. 2017.

Overview

Constraint Solving

Constraint Satisfaction Problem (CSP)

A Constraint satisfaction problem (CSP) consists of:

- a finite set of variables $\{x_1, \ldots, x_n\}$ taking values from their finite domains (denoted by $D(x_i)$)
- a finite set of constraints $\{C_1, C_2, \ldots, C_m\}$ – relations over subsets of the problem's variables

A solution of the CSP is an assignment $x_1 = d_1, \ldots, x_n = d_n$ such that $d_i \in D(x_i)$ and all the constraints are satisfied.

Constraint Solving

Constrained Optimization Problem (COP)

In addition, we have a function $f : D(x_1) \times \ldots \times D(x_n) \rightarrow \mathbb{R}$, and we are looking for a solution that minimizes or maximizes f.

Constraint Solving

Solving CSPs and COPs

- CSPs and COPs are NP-hard in general
- Solved by combination of search and constraint propagation
- Constraint solvers – tools for solving CSPs and COPs
- Constraint modeling: the task of representing the real-world problem as a CSP or a COP
  - MiniZinc – a modeling language of our choice

Overview

Model Description

Automated planning

For modeling, we use the approach based on automated planning:

- state is represented by a set of variables
    - initial state given in advance
- a finite set of operators (or actions), changing the state
    - each operator may have a precondition for its application
- a goal: the condition that should be satisfied in the final state

The objective is to find a plan:

- a finite sequence of operators applicable to the initial state, such that the obtained final state satisfies the given goal

Model Description

Constructions as planning problems

- states $\Leftrightarrow$ sets of constructed objects (points, lines, angles,...)
- operators $\Leftrightarrow$ construction steps
- the goal: the triangle vertices A, B and C are in the final state

Model Description

Encoding objects

We use MiniZinc enumeration types:

```
enum Point = { A, B, C, O, I, G, H, Ma, Mb, Mc, ... };
enum Line = {a, b, c, ma, mb, mc, sa, sb, sc, ha, hb, hc,...};
enum Circle = { kO, kI, kMa, kMb, kMc, kNa, kNb, kNc,...};
enum Angle = { Alpha, Beta, Gamma,...};
```

Important!

Only the objects listed as enumerators can be constructed.

Model Description

Encoding relations

We use MiniZinc model parameters to statically encode the following geometric knowledge:

- incidence relations (points belonging to lines and circles)
- relations between lines (parallel and perpendicular lines)
- circles information (circle centers, diameters and tangents)
- vector ratios, angles information, harmonic conjugates, loci of points . . .

MiniZinc data structures used

Arrays of sets, multidimensional arrays, arrays of tuples. . .

Model Description

Encoding the states (for a fixed plan length n)

The states $S_0, \ldots, S_n$ are encoded using the arrays of set variables:

- known_points[i]
- known_lines[i]
- known_angles[i]
- known_circles[i]

representing the sets of points (lines, angles, circles) belonging to the ith state (i.e. constructed up to the ith step), for each $i \in \{0, 1, \ldots, n\}$.

Model Description

Encoding the plan

To encode the plan, for each step, we must specify the following:

- the type of operator (i.e. construction) used in this step
  - constructing the line through two given points
  - constructing the point that is the intersection of two given lines
  - constructing the circle centered in one given point and containing another given point, ...
- the objects used in the construction step (used for the construction or being constructed)
  - for instance, if we construct the line through two given points, we must fix:
    - which two points are used
    - which line is being constructed (the one passing through these points)

Model Description

Encoding the plan (2)

- We use the enumeration type:

```
enum ConsType = {  LineThrough, LineIntersect, KnownRatio,
                   PerpendicularLine, ParallelLine,
                   TangentCirclePoint, TangentsCirclePoint,
                   LineCircleIntersects, ...
                     };
```

  to enumerate all supported types of construction steps.
- The array of variables `construct`[i] denotes the type of construction applied in ith step ($i \in \{1, 2, \ldots, n\}$)

Model Description

Encoding the plan (3)

We define the arrays of variables:

- `points`$[i][j]$, `lines`$[i][j]$, `circles`$[i][j]$, `angles`$[i][j]$

denoting the objects used in ith step.

- for instance, in the `LineThrough` construction:
  - `points`$[i][1]$ and `points`$[i][2]$ denote the used points
  - `lines`$[i][1]$ denotes the constructed line

Model Description

Encoding the state transitions

Connecting the state variables in successive states:

```
constraint forall(i in 1..n)
(
   construct[i] = LineIntersect ->
       % Precondition
       (lines[i,1] in known_lines[i-1] /\
        lines[i,2] in known_lines[i-1] /\
        lines[i,1] != lines[i,2] /\
        not (lines[i,1] in parallel_lines[lines[i,2]]) /\
        lines[i,1] in inc_lines[points[i,1]] /\
        lines[i,2] in inc_lines[points[i,1]] /\
        not (points[i,1] in known_points[i-1]) /\
       % Effects
        known_points[i] = known_points[i-1] union { points[i,1] } /\
        known_lines[i] = known_lines[i-1] /\
        known_circles[i] = known_circles[i-1] /\
        known_angles[i] = known_angles[i - 1]
       )
);
```

Model Description

Encoding the goal

We require that the triangle vertices are constructed in the final state:

```
{ A, B, C } subset known_points[n];
```

Overview

Evaluation

Evaluation setups

To construct a plan of a minimal length, we try three different setups:

- linear setup: successively solve CSPs for plans of length $n = 1, 2, 3, \ldots, maxLength$, until a satisfiable CSP is encountered
- minimization setup: let n be a variable in domain $\{1, \ldots, maxLength\}$ and solve the corresponding COP that minimizes n
- incremental setup: successively solve COPs (minimizing n) for incremental domain ranges for n (with step k)
  - $n \in \{1, \ldots, k\}$,$\{k+1, \ldots, 2k\}$,$\{2k+1, \ldots, 3k\}$,$\ldots$, $< maxLength$

Evaluation

Comparison to ArgoTriCS

ArgoTriCS[2] – a state-of-the-art dedicated triangle construction solver (developed in Prolog).

[2] Vesna Marinković. ArgoTriCS – automated triangle construction solver. Journal of Experimental & Theoretical Artificial Intelligence. 2017.

Evaluation

Setup	# solved	Avg. time	Median time	Avg. time on solved	Avg. length
linear	63	97.9	22.0	58.5	6.3
minimization	63	43.8	10.8	29.7	6.3
incremental ($k = 3$)	63	66.1	12.0	39.9	6.3
ArgoTriCS	65	54.5	21.6	54.4	7.5

Overall results on 74 Wernick's problems for different setups, compared to ArgoTriCS.

Times are given in seconds

Overview

Conclusions

Advantages of our approach

- Comparable to state-of-the-art tools
- Much less effort to implement
- Finds plans of minimal lengths
- Easy to extend

Conclusions

Thank you for your attention!

Questions?