Automated reasoning project report

Roberto Tonino

November 9, 2024

1 Problem statement

Delle astronavi aliene sono disposte in modo complanare in modo che possiamo immaginarle nel rettangolo cartesiano (0,0), (max_x,0), (max_x,max_y), (0,max_y). max_x e max_y sono dati di input e vengono anche date le coordinate (intere) delle astronavi presenti. Supponiamo non siano già ad ordinata 0 (ovvero sulla terra).

Ci accorgiamo che ad ogni istante di tempo ogni astronave scende unitariamente (ciascuna componente y cala di 1).

Gli alieni sono ostili e vogliamo fermarli. Disponiamo di un cannone all'inizio in posizione (0,0) che in un istante di tempo può (1) sparare verticalmente distruggendo la astronave più vicina che si trovi nella stessa ascissa, oppure (2) spostarsi (senza sparare) in un'altra ascissa. Il tempo per spostarsi di 1,2,3,4, etc caselle sull'asse x è sempre unitario.

Si vuole trovare un piano (se esiste) per evitare che anche solo una astronave tocchi terra.

P.S. É possibile ci sia una soluzione furba, ad-hoc, per il problema. Non cerchiamo quella. Vogliamo codificarlo e fare risolvere al CP solver o al ASP solver.

2 Clingo model

In the first few lines we find the domain predicates. The time/1 predicate starts from 0. This is to indicate the initial state of the world and means that the model will use t+1 time steps. The cannon predicate is arguably is written such that the model can have more than one cannon, but practically—for this project—the model never takes into account more than one cannon. The sole purpose of this oddity is to allow the use of the cannon predicate with a variable `C`, like so: `cannon(C)`.

The alien/1 predicate comes with an identifier, which is an counter starting from 1.

Line 11—with the choice rule—is a core part of the model. It states that for each time step, except for the last one, the cannon either moves to a different position on the grid or shoots to an alien. The last time step is left out because of how the *shoot* action is defined further on: when the cannon shoots, an aliens is considered dead only in the next time step.

The at/4 predicate is used to define the initial state of the model. It positions the cannon at the coordinates (0,0) as required in the problem statement.

The at/4 predicate is a core part of the model as well. It represents the position on the grid of the cannon or of an alien, at any given time step.

The model contains the *move* and the *shoot* actions: the former represents the movement of the cannon on the x axis, while the latter represents the cannon shooting and killing an alien. The precondition for the *shoot* action is to have the cannon and the alien at the same x coordinate.

There is no *move* action for the alien because the requirement of aliens decreasing their y coordinate at every time step is met via an inertia rule.

A postcondition of the *shoot* action is that the cannon does not change position in the same time step. This meets one of the requirements of the problem.

The inertia rules involve aliens. If an alien dies, it will remain dead and it will preserve its position on the grid. If an alien is not dead, its y coordinate will decrement by 1 at each time step.

Lastly, the constraints avoid the model to accept a solution where the cannon has a y coordinate different than zero—at each time step. Additionally, they avoid a solution where an alien reaches $y = 0$, which would violate the requirements in the problem statement.

3 Minizinc model

In the Minizinc model, we define upfront the variables we need the solver to assign values to.

The model represents time steps as array indices. Like in the clingo model, the `move_x` and the `shoot` variables span in `t` time steps, compared to t+1 time steps of the other decision variables.

Actions are represented by the following first-order logic formula:

$$\forall a \in L \; \exists t \in \{0, \ldots, T_{max} - 1\} \; (P \wedge A \wedge E)$$

Where L is the set of a**l**iens, T_{max} is the maximum number of time steps, and P, A, and E—representing **p**reconditions, **a**ctions and **e**ffects—are first-order logic formulas constructed as follows:

$$P ::\equiv \top \mid \neg isDead(a,t) \mid equals(cannonX(t), alienX(t))$$

$$A ::\equiv shoot(a,t) \mid moveX(t)$$

$$E ::\equiv isDead(a,t) \mid equals(cannonX(t), cannonX(t+1)) \mid equals(cannonX(t), alienX(t))$$

$moveX$ returns the x coordinate where the cannon will move on the next time step. This is a decision variable that the solver has to find.

In the model we also find a constraint that enforces one of the two actions to be present for each time step.

Inertia rules are represented by the following first-order logic formulas:

$$\forall t \in T \; \forall a \in L \; (isDead(a,t) \implies isDead(a,t+1) \wedge equals(alienY(t), alienY(t+1)))$$

$$\forall t \in T \; \forall a \in L \; (\neg isDead(a) \implies equals(alienY(t+1), alienY(t) - 1)$$

Where T is $\{0, \ldots, T_{max} - 1\}$.

Finally, the only global constraint in the model prohibits the cannon to shoot to the same alien twice.

4 Benchmarks

4.1 Clingo instances

Both clingo and Minizinc models benchmarks were run on 10 problem instances. The instances were generated with the program `generate.ts` (included in the zip file) that is run with the command

```
bun generate.ts <x> <y> <numberOfAliens>
```

(It uses bun, a modern TypeScript runtime, but can be ran with node.js as well with some additional work.)

The program contains 3 inputs: `x`, `y` and `numberOfAliens`. The y coordinates of the aliens are generated by taking the input `y` and decrementing it for each alien present in the instance, obtaining a sequence that starts at $y - 1$ and ends at $y - numberOfAliens$. The x coordinates of the aliens are randomly generated between 0 and the input `x`. This method of generating the coordinates ensures that no aliens are in the same position on the grid.

It is worth noting that if the input `y` is smaller than $2 * numberOfAliens$, the model will most likely be unsatisfiable because the cannon won't be able to kill all the aliens before they reach the 0 coordinate because the combination of shoot and move takes 2 time steps. For example, if there are 50 aliens, all with different x, then 100 time steps are required for the cannon to kill all the aliens in time. To enable that, the aliens need to be in a high-enough y coordinate to not reach 0.

4.2 Minizinc instances

The `generate.ts` file generates instances suitable to be used with clingo. A combination of a shell script, a TypeScript script and a manual step are used to convert them to Minizinc instances. The two scripts can be found in the zip file and are named `minizinc/convert` and `minizinc/convert.ts` respectively. The `convert` script creates 10 files names `aliens.1.mzn`, `aliens2.mzn`, etc. The files contain two arrays which values are extracted from `clingo/aliens1.lp`, `clingo/aliens2.lp`, etc. The manual step consists in adding four integer parameters to the generated files. The four integer parameters are: `t`, `mx` (max_x), `my` (max_y), and `a` (number of aliens).

4.3 Benchmark setup

Every instance is ran 10 times with 3 warmup runs. The clingo model was ran with three different configurations: *auto*, *jumpy*, and *trendy*. The Minizinc model was ran with two different solvers: default, and *chuffed*.

The results—in JSON format—are stored in `clingo/benchmarks` and `minizinc/benchmarks`.

4.4 Clingo model

In Figure 1 a visualization of the results is presented. On the x axis there are the 10 aliens files, while on the y axis there is running time. There are 3 bars for each file representing the 3 different configurations. In Table 2 the full benchmark results are presented.

The *trendy* configuration is the fastest for this problem. The performance measured on `aliens8` suggests that *trendy* could be faster when using instances of bigger size.

It is interesting to note how `aliens3` reaches the timeout with the default configuration, while with *jumpy* and *trendy* the resolution takes around 1 second. This is the breakdown of the flags that the configurations are setting (generated by running `clingo --help=3`):

```
[trendy]:
 --sat-p=2,iter=20,occ=25,time=240 --trans-ext=dynamic --heuristic=Vsids
 --restarts=D,100,0.7 --deletion=basic,50 --del-init=3.0,500,19500
 --del-grow=1.1,20.0,x,100,1.5 --del-cfl=+,10000,2000 --del-glue=2
 --strengthen=recursive --update-lbd=less --otfs=2 --save-p=75
 --counter-restarts=3,1023 --reverse-arcs=2 --contraction=250 --loops=common
```

```
[jumpy]:
 --sat-p=2,iter=20,occ=25,time=240 --trans-ext=dynamic --heuristic=Vsids
 --restarts=L,100 --deletion=basic,75,mixed --del-init=3.0,1000,20000
 --del-grow=1.1,25,x,100,1.5 --del-cfl=x,10000,1.1 --del-glue=2
 --update-lbd=glucose --strengthen=recursive --otfs=2 --save-p=70
```

From these exhaustive lists of command line flags, we can extract simply `--sat-p=2` to get a significant running time speedup compared to the *auto* configuration. The flag enables "SAT preprocessing", more precisely "SatELite-like" preprocessing [1]. SAT preprocessing alone shows a 13-fold improvement, as presented in Table 1.

Aliens	Configuration	Mean	Std Dev	Min	Max	Times faster	Percentage
aliens3	auto	300.01 s	0.0012 s	300.00 s	300.01 s	1.0x	baseline
aliens3	`--sat-p=2`	23.06 s	0.99 s	21.59 s	24.67 s	13.0x	-92.31%

Table 1: `aliens3` with `sat-p` flag

When the search space becomes of considerable dimensions, the *auto* configuration struggles. It almost reaches the timeout with `alien6` and always reaches it from `aliens7` upwards. *Jumpy* and *trendy* perform well under the timeout with `aliens6`, but reach it with `aliens7`. With `aliens8` both configurations run well under the timeout, with *trendy* running under the minute threshold.

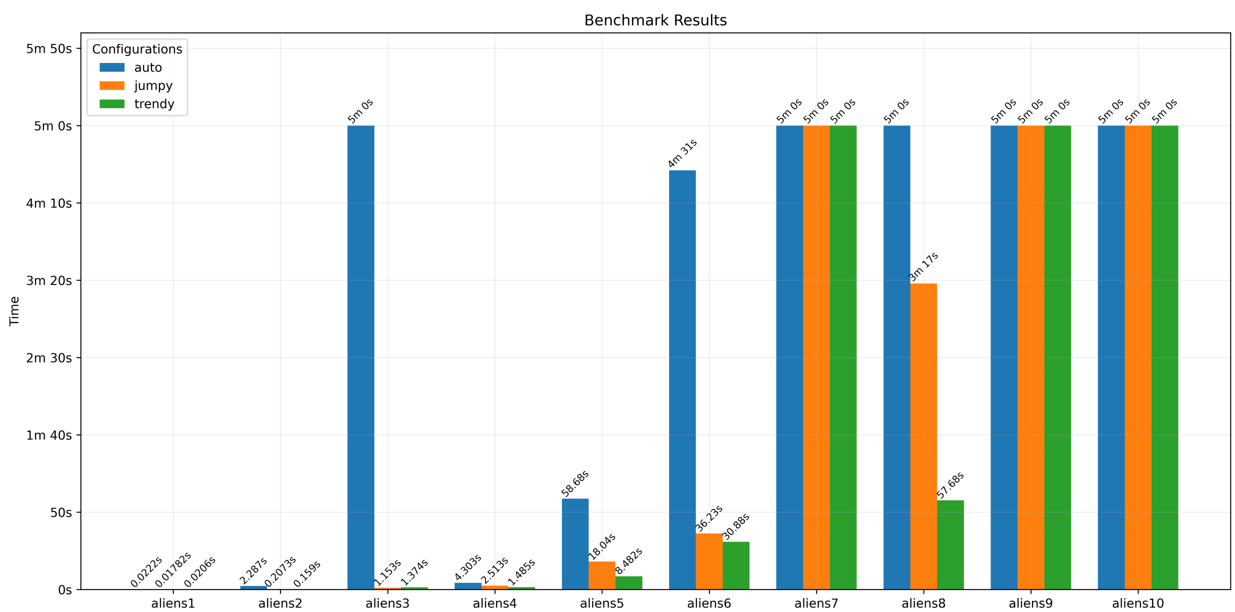

Figure 1: Comparing clingo benchmark results

In Table 3 the list of files is presented together with their respective inputs.

4.5 Minizinc model

In Figure 2 a visualization of the results is presented. The `org.chuffed.chuffed` solver has a huge benefit with `aliens2`. This advantage vanishes starting with `aliens3`. From `aliens3` to `aliens10` the model, ran with the default solver and with `org.chuffed.chuffed`, always reaches the timeout. In Table 4 the full benchmark results are presented.

This means that the model solves the problem, but in an inefficient way. Especially compared to the clingo model.

In Table 5 the list of files is presented together with their respective inputs.

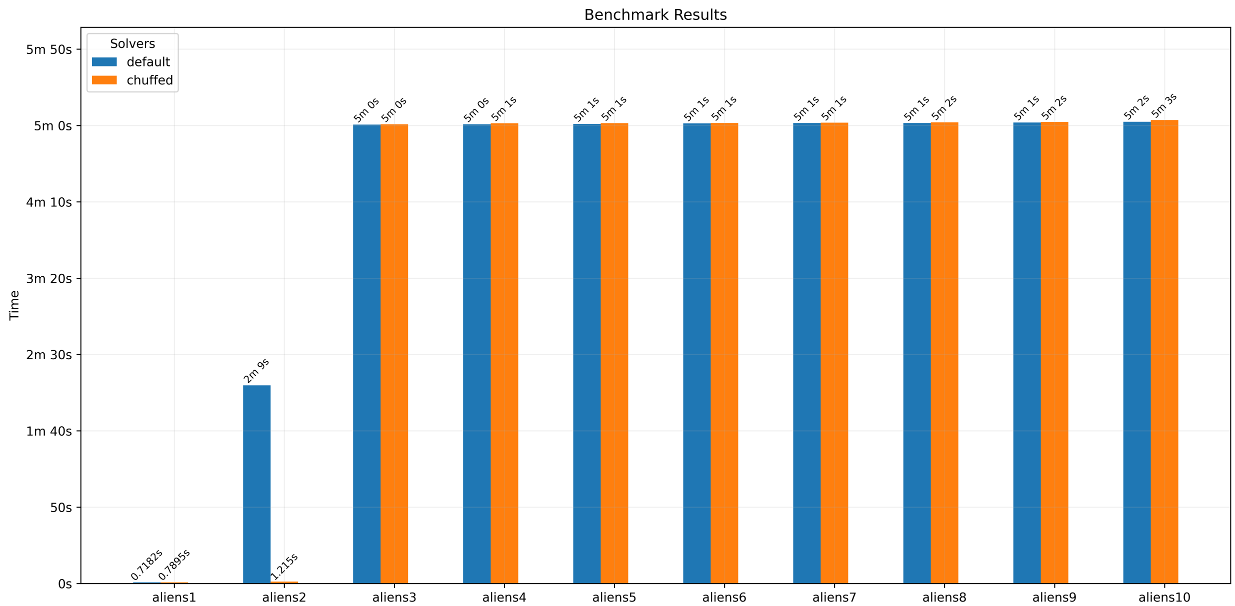

Figure 2: Comparing minizinc benchmark results

5 Conclusion

The clingo model performs better than the clingo model, especially when the latter is ran with the *trendy* configuration. Some improvements can be done to speed up the solving of the models. Both the clingo and Minizinc models do not have constraints on how the cannon should move or shoot. The cannon can be constrained to move to the closest alien or to shoot to all the aliens in the same x coordinate, therefore reducing the search space and solving times.

References

[1] Martin Gebser et al. *Clingo Guide*. v2.2.0. Available at https://github.com/potassco/guide/releases/tag/v2.2.0. University of Postdam. 2019.

6 Appendix: tables

Aliens	Configuration	Mean	Std Dev	Min	Max	Times faster	Percentage
aliens1	auto	0.022 s	0.008 s	0.014 s	0.039 s	1.0x	baseline
aliens1	jumpy	0.018 s	0.0016 s	0.015 s	0.021 s	1.2x	-19.76%
aliens1	trendy	0.021 s	0.0016 s	0.019 s	0.024 s	1.1x	-7.21%
aliens2	auto	2.29 s	0.6 s	1.84 s	3.48 s	1.0x	baseline
aliens2	jumpy	0.21 s	0.0093 s	0.19 s	0.22 s	11.0x	-90.93%
aliens2	trendy	0.16 s	0.015 s	0.15 s	0.2 s	14.4x	-93.05%
aliens3	auto	300.01 s	0.0012 s	300.00 s	300.01 s	1.0x	baseline
aliens3	jumpy	1.15 s	0.044 s	1.12 s	1.27 s	260.2x	-99.62%
aliens3	trendy	1.37 s	0.017 s	1.35 s	1.40 s	218.3x	-99.54%
aliens4	auto	4.30 s	0.049 s	4.24 s	4.39 s	1.0x	baseline
aliens4	jumpy	2.51 s	0.026 s	2.48 s	2.55 s	1.7x	-41.60%
aliens4	trendy	1.48 s	0.022 s	1.45 s	1.51 s	2.9x	-65.50%
aliens5	auto	58.68 s	0.38 s	58.16 s	59.30 s	1.0x	baseline
aliens5	jumpy	18.04 s	0.18 s	17.83 s	18.46 s	3.3x	-69.26%
aliens5	trendy	8.48 s	0.13 s	8.35 s	8.80 s	6.9x	-85.54%
aliens6	auto	271.09 s	0.52 s	270.06 s	271.75 s	1.0x	baseline
aliens6	jumpy	36.23 s	0.34 s	35.89 s	36.80 s	7.5x	-86.63%
aliens6	trendy	30.88 s	0.45 s	30.43 s	31.79 s	8.8x	-88.61%
aliens7	auto	300.01 s	0.0025 s	300.00 s	300.01 s	1.0x	baseline
aliens7	jumpy	300.01 s	0.00071 s	300.00 s	300.01 s	1.0x	-0.00%
aliens7	trendy	300.01 s	0.00055 s	300.00 s	300.01 s	1.0x	-0.00%
aliens8	auto	300.01 s	0.0014 s	300.01 s	300.01 s	1.0x	baseline
aliens8	jumpy	197.89 s	1.78 s	196.43 s	202.73 s	1.5x	-34.04%
aliens8	trendy	57.68 s	0.29 s	57.30 s	58.29 s	5.2x	-80.77%
aliens9	auto	300.01 s	0.0018 s	300.01 s	300.01 s	1.0x	baseline
aliens9	jumpy	300.01 s	0.0064 s	300.01 s	300.03 s	1.0x	+0.00%
aliens9	trendy	300.01 s	0.0011 s	300.00 s	300.01 s	1.0x	-0.00%
aliens10	auto	300.01 s	0.0016 s	300.01 s	300.01 s	1.0x	baseline
aliens10	jumpy	300.01 s	0.0022 s	300.01 s	300.02 s	1.0x	+0.00%
aliens10	trendy	300.01 s	0.0011 s	300.01 s	300.01 s	1.0x	-0.00%

Table 2: Complete clingo benchmark results

File	Number of aliens	max_x	max_y	t
aliens1.lp	10	15	15	5
aliens2.lp	20	30	30	10
aliens3.lp	30	45	45	15
aliens4.lp	40	60	60	20
aliens5.lp	50	75	75	25
aliens6.lp	60	90	90	30
aliens7.lp	70	105	105	35
aliens8.lp	80	120	120	40
aliens9.lp	90	135	135	45
aliens10.lp	100	150	150	50

Table 3: Instances used for benchmarking the clingo model

Aliens	Solver	Mean	Std Dev	Min	Max	Times faster	Percentage
aliens1	default	0.72 s	0.034 s	0.68 s	0.8 s	1.0x	baseline
aliens1	chuffed	0.79 s	0.048 s	0.74 s	0.86 s	0.9x	+9.94%
aliens2	default	129.84 s	3.59 s	123.64 s	136.12 s	1.0x	baseline
aliens2	chuffed	1.22 s	0.041 s	1.17 s	1.28 s	106.8x	-99.06%
aliens3	default	300.58 s	0.027 s	300.55 s	300.64 s	1.0x	baseline
aliens3	chuffed	300.77 s	0.041 s	300.73 s	300.86 s	1.0x	+0.06%
aliens4	default	300.80 s	0.025 s	300.75 s	300.84 s	1.0x	baseline
aliens4	chuffed	301.44 s	0.042 s	301.40 s	301.54 s	1.0x	+0.22%
aliens5	default	301.09 s	0.036 s	301.02 s	301.13 s	1.0x	baseline
aliens5	chuffed	301.60 s	0.024 s	301.55 s	301.65 s	1.0x	+0.17%
aliens6	default	301.37 s	0.02 s	301.34 s	301.41 s	1.0x	baseline
aliens6	chuffed	301.70 s	0.027 s	301.65 s	301.74 s	1.0x	+0.11%
aliens7	default	301.66 s	0.038 s	301.60 s	301.72 s	1.0x	baseline
aliens7	chuffed	301.86 s	0.031 s	301.81 s	301.91 s	1.0x	+0.07%
aliens8	default	301.62 s	0.065 s	301.57 s	301.79 s	1.0x	baseline
aliens8	chuffed	302.06 s	0.044 s	302.01 s	302.15 s	1.0x	+0.15%
aliens9	default	301.97 s	0.063 s	301.89 s	302.08 s	1.0x	baseline
aliens9	chuffed	302.31 s	0.032 s	302.27 s	302.38 s	1.0x	+0.11%
aliens10	default	302.41 s	0.065 s	302.33 s	302.57 s	1.0x	baseline
aliens10	chuffed	303.61 s	0.89 s	302.86 s	305.96 s	1.0x	+0.40%

Table 4: Complete Minizinc benchmark results

File	Number of aliens	max_x	max_y	t
aliens1.lp	10	15	16	5
aliens2.lp	20	30	31	10
aliens3.lp	30	45	46	15
aliens4.lp	40	60	61	20
aliens5.lp	50	75	76	25
aliens6.lp	60	90	91	30
aliens7.lp	70	105	106	35
aliens8.lp	80	120	121	40
aliens9.lp	90	135	136	45
aliens10.lp	100	150	151	50

Table 5: Instances used for benchmarking the Minizinc model