Optaplanner

Jiri Vahala

February 26, 2016

Planning problems

- planning problems in general
- sequence of decisions leading to solution
- in general, planning problems are very interesting
- there's a little catch - they are very hard to solve :)
- NP-complete complexity

Little "math" behind

- complexity of problems
- P, NP, ... what is NP-complete
- planning problems belong into NP-complete
- for all NP-hard problems there is correct exponential algorithm
- one million dollar question: P == NP ?

Little "math" behind

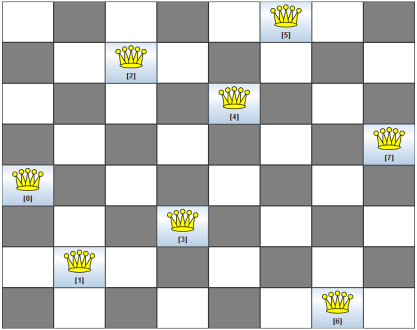
- Difference between deterministic and non-deterministic Turing machine
- state-space
- what is going on when DTM is solving problem with exponential time complexity
- all NP-complete problems can be reduced on each other
- SAT - finding satisfying boolean values for all variables in logical formula
- ... all NP-complete problems are about finding right values for variables in problem
- what is NP-complete? (SAT, TSP) list of NP-C problems

Scoring states

- we need to know how good are picked values in state
- scoring states is needed
- scoring function ties state with it's score
- this allows us to have order over states
- some problems can have only True/False evaluation

NQueens

- problem of placing N chess queens on an N x N chessboard so that no two queens threaten each other
- restricting queens to be movable only in column
- values we want to find are indexes of rows for each queen
- scoring queens - how many of them are threatened



Optaplanner comes to save the day!

- planner is fast, stable, robust, generic, awesome, sweet with rainbow and unicorns
- NP-complete problem solver (constraint satisfaction solver)
- implements multiple different heuristics performing local-search
- goal is to find solution which is near to optimal solution

State space

State space

- from now, state space is everywhere!
- how do you imagine distance of two states in state space?
- it's crucial to understand states, state space and it's features
- local optima are created when score is used
- landscape of states (demonstration)

Steps through sate-space

- to move through state-space, we need ... well ... moves :)
- we can theoretically generate exponential count of moves ... from current state ... but one of them must be optimal!
- restricting generated moves
- scoring moves
- step is equal to move, which was picked from all proposed moves
- (demonstration!)

What is local-search

- general, math based, approach to solving problems
- moving though problem's state-space (demonstration)
- requires initialized problem (well ...)
- example of local search on Nqueens state-space ...

Why brute-force or branch and bound isn't enough?

- brute force is practically depth-first search
- all NP-complete problems are transferable into DFS
- brute force (113)
- branch and bound - smart brute-forcing
  - hits the performance wall anyway

Local search algorithms

- general, math based, algorithms
- Hill climbing
- Tabu search
- Simulated annealing
- Late acceptance
- (137)
- few others...
- infinite fight with local optima!

Hill climbing

- greedy search
- takes the best state
- worse state then current state is not allowed
- highly dependable on initialization of problem
- (138)
- local optima are huge problem

Tabu seach

- queue of used entities/values
- we don't change what is in queue!
- queue has specified size - all entities are freed sooner or later
- (141)

Simulated annealing

- probabilistic technique
- based on metallurgy (reheating steel for stronger connections between molecules)
- decreasing "temperature"
- calculates probability(from temperature) of picking worse state
- the worse state, the smaller probability
- https://en.wikipedia.org/wiki/Simulated_annealing nice gif!
- (143)

Late Acceptance

- queue of scores (for each accepted step one)
- we look at the end of the queue
- ... and allow to accept all states that are equal or better than the last score in queue
- (143)

How we get problem initialized

- construction heuristics is the answer
- greedy algorithms for initializing problem better than random walk
- few different approaches, depending on domain

Construction heuristics

- different strategies
- some knowledge about problem can help us here
- first fit, first fit decreasing, cheapest insertion, strongest fit atc....
- (124)

Deeper look on solving

- phases, steps and moves
- phases executes construction/local-search algorithms
- steps are winning moves for particular state and phase
- moves are the deepest part of whole puzzle. huge amount of moves is generated while planner is solving, some are not even doable
- (131)

Move selectors

- in planner machinery, move selectors are generators of moves
- two types: simple and chaining
- change, swap, pillar change, pillar swap, tail
- move selectors can be combined

Theoretical recap.

- planning problems belong into NP-complete complexity and NP-complete problems are hard
- solving problems can be transformed into state-space search while using scoring function
- Optaplanner implements heuristics for performing such search
- we can combine heuristics by setting up multiple phases
- moving through search is done by generating restricted set of moves and picking the best one with respect to used heuristic
- if we are lucky, system converges to local optima :)

Optaplanner - practical overview

- Optaplanner is 100% pure java engine
- http://www.optaplanner.org/
- great documentation and examples!
- opensource
  - https://github.com/droolsjbpm/optaplanner
  - Apache Software License 2.0
- typically, user must care only about: domain model, score function, config file

Domain model

- from theoretical model - variables and their values
- variable - planning entity
- value - planning value :)
- planning solution - class representing the whole problem (set of all variables and all values available to use)
- using annotations

Planning variable

- simple java class
- represents variable domain
- usually has some attributes
- nothing special here
- see domain example ...

Planning entity

- annotation @PlanningEntity
- represents changing variable in problem
- ... so getting various values over time
- value can be different entity - chaining
- annotated getter for planning variables as @PlanningVariable(valueRangeProviderRefs = "label")

Planning solution

- annotation @PlanningSolution
- implements interface Solution
- list of planning entities
- @PlanningEntityCollectionProperty - annotated getter returning all entities
- @ValueRangeProvider(id = "label") - annotated getter returning values for entities
- score
- see example

Score

- in theoretical view, score function evaluates states with some order.
- ... in practice, we need multiple numbers :)
- score levels - some levels are more important than others
- when problem is unsolvable, it's worse than problem which cost one million
- different scores: Simple, Hard/Soft, Hard/Medium/Soft, Bendable ... with different types (Integer, Long, Double, BigDecimal)

Scoring functions

- we need scoring functions!
- when you think about it ... (with domain model) it defines our problem!
- there are three possible implementations
  - easy score calculation
  - drools score calculation
  - incremental score calculation

Easy score calculation

- easiest implementation
- slowest one - score function is calculated from the scratch
- good proof of concept
- can be used to control other score functions
- (example)

Drools score calculation

- faster than easy score function
- based on drools engine
- score constraints are calculated via rules

Incremental score calculation

- the fastest score calculation
- it calculates score diff for only changed entities (after move)
- can be challenging to write it correctly

Putting it all together

- what we have so far ...
- we have pure engine which takes domain model and realizes abstract local search algorithm with respect to defined scoring
- we know, how to create domain model
- ... and how to score it
- last configuration is needed!

Configuration file

- xml format
- mentioned domain classes
- if needed, specification of construction heuristics
- if needed, specification of local search heuristics
  - acceptor - defines type of local search algorithm and it's parameters
  - forager - specification of how many moves are accepted each step
- see example (multiple)

Environment

- asserting score corruption, checking validity
- PRODUCTION - random seed
- REPRODUCIBLE - same as production but with seed equals to zero
- FULL_ASSERT, NON_INTRUSIVE_FULL_ASSERT, FAST_ASSERT

Termination

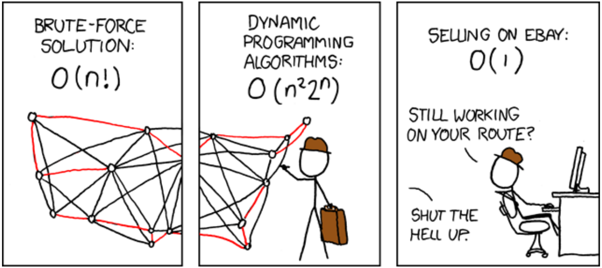
- dealing with huge problems - when to stop?
- different approaches:
- terminations based on:
    - time
    - steps count
    - best score
    - feasible score
    - non-improving score (time, steps)
    - combinations ...

Phases

- construction heuristics - phase n. one
- phases defines different approaches to solution
- sequential! (weakness?)
- why do we need multiple phases. (Do we really need them?)

User is part of the heuristic!

- general selectors features (combining selectors, caching, filtered selection, probabilistic selection, limited selection ...)
- initializing trends, early picking
- is better smaller count of accepted moves? or bigger? what should we chose?
- planner is very general ... so understanding of specific problem is important!



credit: https://xkcd.com/399/

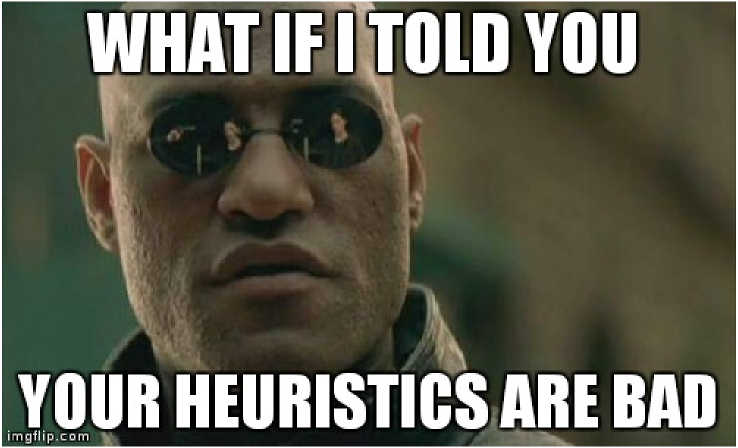
Examples

- cloud balance easy model
- cloud balance sophisticated model
- implement your own bin(bag) packing!
  - each bag has it's own volume
  - each brick has it's own volume
  - goals: pack all bricks in as small amount of bags as possible

Chaining

- when entities(variables) are values
- huge switch in planner for "chaining"
- different planner strategies are used when chaining mode is on
- we are really talking about chain and not general graph!
- solution can have multiple chains
- each chain needs to have anchor
  - anchor is immovable, it's head of the chain
  - every other node is movable entity
  - there must be interface which is implemented by both - so first entity can point to anchor
- chain corrections - not doable moves are detected
- (see TSP example)

Benchmarking



Benchmarking

- which heuristics is better then others (and when)
- effective comparison over many aspects of heuristics
- benchmarker automatically generates html report
- benchmark config extends solver config
- can be parameterized over many datasets and solver instances
- (see benchmarker example)

Other interesting topics

- multiple entities - local search needs more detailed configuration
- shadow variables - variables which are deduced from planning variables(values)
- immovable entities - we can set entity immovable, so planner can't change it's variable
- realtime planning - we can change problem facts asynchronously during planning - CH is automatically used - then local search phase continues
- value range - variables can be defined by interval of values instead (numeric)
- nearby selection - probabilistic local search based on comparator of entities
- custom phases, moves ...

What can be solved using planner

- everything? how fast? and is it optimal?
- what about function optimization?

imgshow.com/image/13/nhsDb

Some tips

- ONLY_DOWN is usually free performance gain
- incremental score calculation is gold, but use easy-score checker, at least at start
- when you are trying to solve black-box, you end up with poor solution
  - even little knowledge can cut off huge part of state-space and lead to performance gain
- investigate different heuristics (even at different steps of solving)
- do not rape optaplanner when not needed