Efficiently completing partial configurations

Toward automatically learned search heuristics for CSP-encoded configuration problems

Results from an initial experimental analysis

Dietmar Jannach

TU Dortmund, Germany

dietmar.jannach@tu-dortmund.de

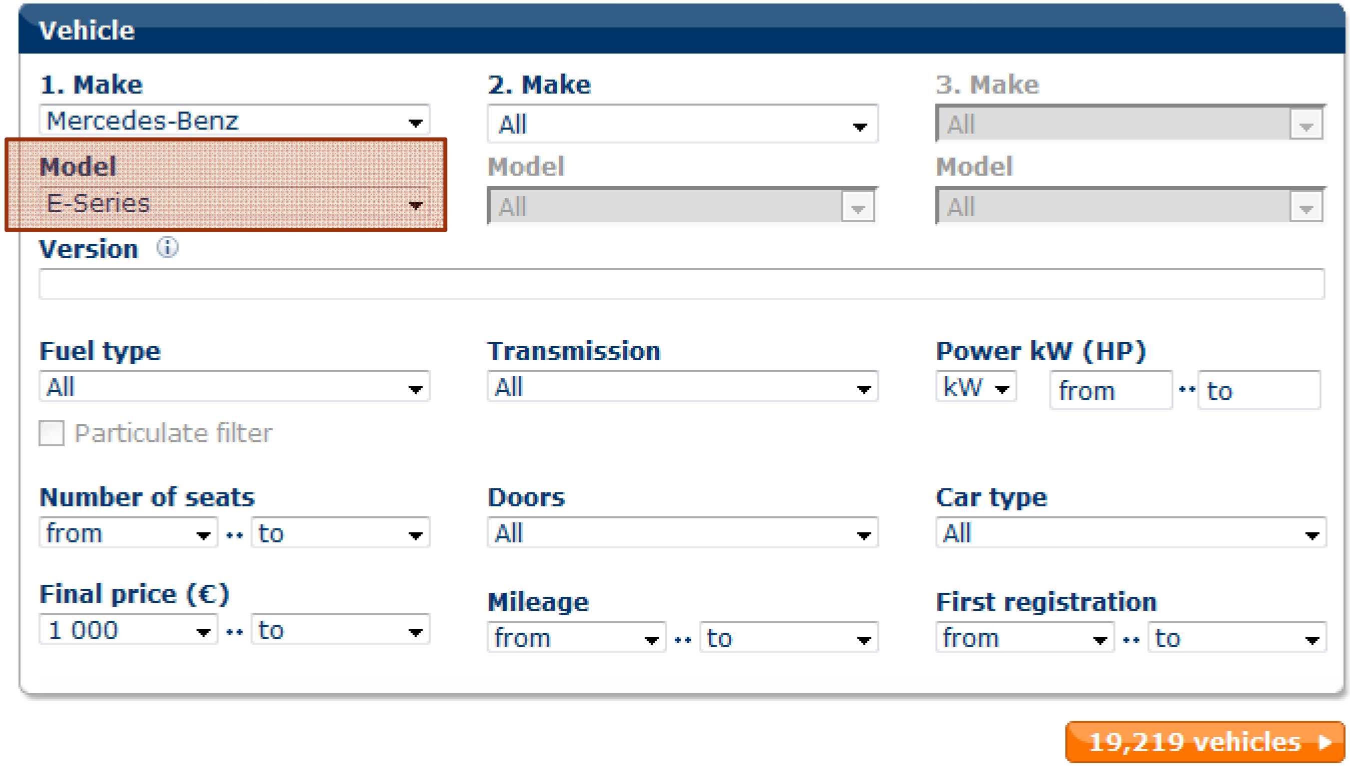
Background

- Not all configurations are created equal

- Looking for an E-series Mercedes?

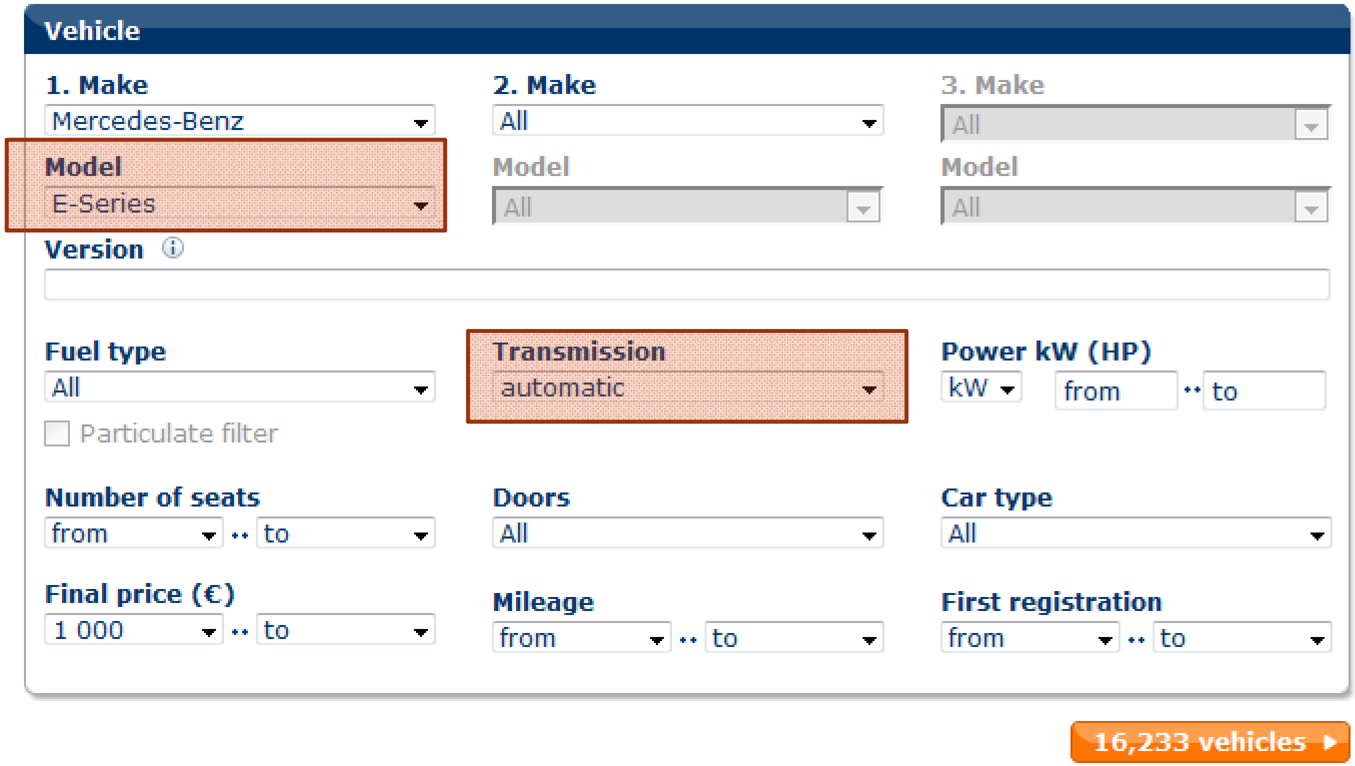
Common combinations

- 19,219 used cars online
  - Customer requirement: "E-Series"



Common combinations

- 16,233 (~84%) with automatic transmission
  - Customer requirement: "E-Series", "automatic transmission"



Main hypothesis & approach

- Configuration problem solving can be hard
  - Configurations can comprise thousands of parameter settings
  - Despite the use of high-performance solvers, domain-specific heuristics might be required for efficient problem solving

- Observations:
  - Some configurations are much more likely (popular) than others
  - The majority of customers might have very similar requirements
    - See yesterday's talk on customer demanded variety

- Therefore:
  - It might be good to explore the "popular" part of the search space first
  - Where to search first, can be learned from past configurations

A CSP-based approach

- Constraint Satisfaction
  - Long tradition of modeling configuration problems as Constraint Satisfaction Problems
  - Basic form, given
    - V – set of variables with defined domains (D)
    - C – a set of constraints on legal, simultaneous value assignments
  - Find:
    - An assignment of a value to each variable in V such that all constraints from C are satisfied

- Advanced CSP models
    - Partially based on requirements from the configuration domain
      - Dynamic CSPs – some variables are only relevant in certain situations
      - Generative CSPs – variables can be added dynamically to the problem

In this work

- Goal
  - Demonstrate the general plausibility and feasibility of a learning-based approach
- What has been done?
  - A simulation-based experiment using CSP benchmark problems
  - Compare problem solving time for different search (branching) heuristics
    - A) Default strategy of the solver
    - B) A learning-based strategy that uses statistics about previous successful configurations
    - **Idea:** If the user chose an E-Series model, try the option "automatic transmission" before the "manual" transmission. (even simpler, in fact)

Protocol details

1. Find a set of suitable CSP benchmark problems
    - Used CSP problems from the CP'08 solver competition
    - Both standard problems (N-Queens) and a true configuration problem (Renault)
    - Problems should be easily solvable (below 1 sec)
2. Simulate configuration problem instances for learning
    - Determine some variables to be input variables
        - e.g., 5 variables with domain size 10 (leading to 1000 possible inputs)
    - Search for valid solutions given some random or biased inputs
        - Record the solutions using the default strategy
3. Learn a good strategy (a trivial one in our case)
4. Re-solve the same problems using the learned strategy
5. Compare the running times

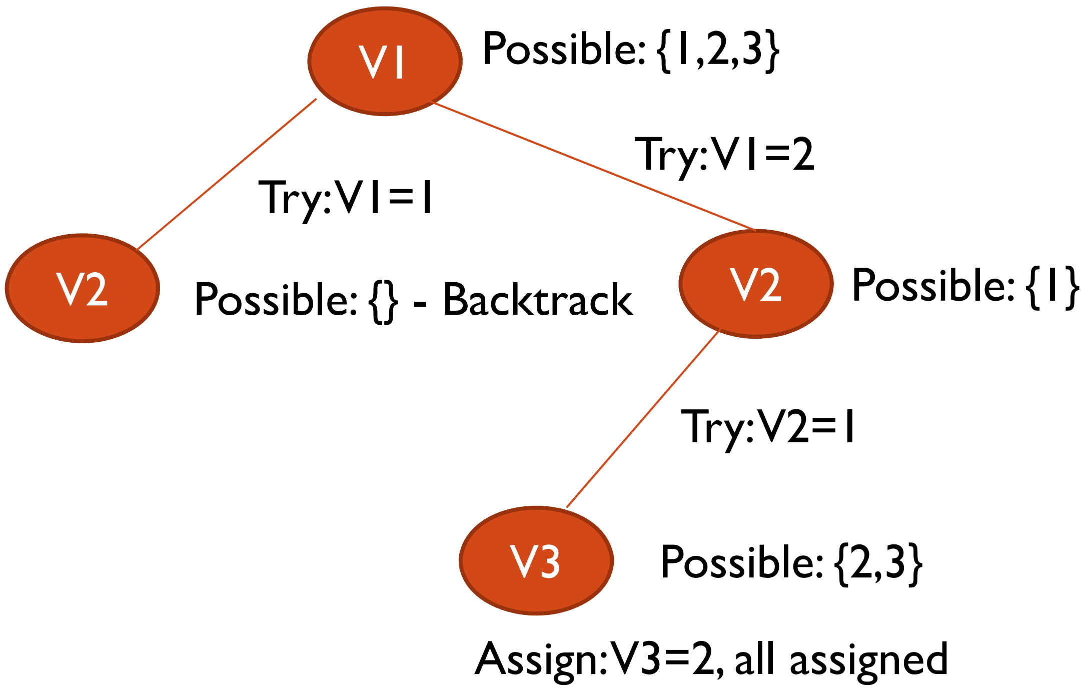
Statistics-based search space exploration

- Simple learning strategy applied as proof-of-concept
  - When "trying out" different value assignments, try the one that was part of the most solutions so far
    - Not depending on inputs
    - Not depending on other variable assignments

- More advanced strategies are of course possible
    - Make choice dependent on other assignments so far
    - Learn more complex rules,
      - e.g., based on Association Rule Mining
    - Perform a static analysis, induce additional "constraints"

Technically – Adapt the branching strategy

- A basic CSP search strategy
  - V={V1,V2,V3}, C={V2<V1,V2<V3}
  - Domains = {1,2,3}



- Standard backtracking
- Constraint propagation omitted here

Choice points – Variables and Values

- Two decision points:
  - Which variable to try next?
    - e.g., based on Fail-First principle (minimum domain)
  - Which value to try first?
    - e.g., based on the order (increasing domain)

- Choice strategy depends on problem structure
  - Solving a standard benchmark with Choco (Java-based solver)
    - Default strategy: 1 minute (!)
    - Impact-based branching: 800ms
    - Increasing domain: 500ms
    - Decreasing domain: 30ms

Statistics-based branching

- Implementation of a trivial "ValueSelection" class
  - Extension mechanism of Java-based constraint solver Choco used
  - Strategy is based on a static ordering of values for each variable determined in the learning phase
    - If no ordering exists or some values were never part of a solution, use a typical default strategy (Increasing Domain)

```
public class StatisticBasedValueSelection
        implements ValIterator<IntDomainVar>   {
        ...
    }
```

More protocol details

- For each benchmark problem …
- Statistics collection phase
  - Randomly determine "input" variables

```
For (i = 1 to 300)
    Create random inputs using Gaussian distribution
        as not all inputs are equally frequent
    Search for a solution
    IF solution exists
        increase the "successful value" counters for the variables
        remember the required solution time
    IF (i = 30 or i = 50 or … i = 300)
        save a snapshot of the statistics so far
```

  - Results:
    - Average running times with default strategy (300 runs)
    - Statistics of the form V1 = [4,2,3,5,1], V2 = [3,2,4,1,5]

More protocol details

- Measuring the effects (for each benchmark problem)
  - For each snapshot (30, 50, 100, 150, 200, 300)

    For (i = 1 to 300)

      Create random input values for the input variables used in the collection phase; do not use exact same inputs (solution caching)

      Search for a solution

      If solution exists

        record the required running times

- Results
  - Required running times for different learning levels

Measurements (CPU time): initial results

	Problem name	Default	30	50	100	150	200	300	Diff.
1	normalized-renault-mod-0_ext.xml	143,44	37,03	26,20	26,60	28,72	28,70	30,77	-82%
2	normalized-bibd-8-14-7-4-3_glb.xml	11,71	7,65	6,94	7,71	7,64	8,38	6,48	-41%
3	normalized-squares-9-9.xml	53,75	41,88	44,30	41,72	42,40	43,32	44,90	-22%
4	normalized-geo50-20-d4-75-29_ext.xml (less inputs)	431,25	353,07	366,06	327,03	354,16	342,25	352,29	-24%
5	normalized-geo50-20-d4-75-24_ext.xml	95,11	99,64	91,66	87,19	93,27	100,32	99,39	-8%
6	normalized-costasArray-13.xml	53,36	48,33	47,70	47,18	47,13	46,96	46,49	-12%
7	normalized-air05.xml	856,51	847,22	859,04	850,81	854,80	848,71	853,36	-1%
8	normalized-magicSquare-5_glb.xml (GAUSSIAN)	112,59	142,86	149,65	145,17	140,57	147,66	145,05	25%
9	normalized-magicSquare-5_glb.xml (RANDOM)	122,81	154,49	144,64	137,90	141,06	131,88	139,49	7%

- Strongest effect on real configuration problem
  - 111 variables, average domain size = 5, 6 input variables, > 15.000 poss. input comb.
  - up to 82% decrease in search times
- Good effect also on other problems
- Running times can slightly increase again when more data exist
  - No statistical significance tests made so far
- Results get worse when problem structure is symmetric
  - Magic squares (e.g., assign each number from 1 to 9 on a 3-by-3 field)
  - Also experimented with using uniform distribution

Observations

- Already trivial strategies can lead to significant reductions in search time
  - Assumption is a non-uniform distribution of customer requirements / configurations
  - Achievable improvements depend on the problem structure
- Looking at standard deviations (Renault problem)
  - Default strategy: 220ms, Statistics-based strategy: around 110ms
  - Standard deviation also gets lower
    - But is larger when compared to overall running times
  - Interpretation
    - Statistics-based search in many cases very fast
    - But there are more cases where the solver is guided to wrong area of search space

Previous works

- Not many papers found
  - Pointers to corresponding literature welcome
- "Online learning" approaches
  - Try to adapt the strategy during one search process
    - e.g., determining the likelihood of the existence of at least one solution in the search graph to be explored
      - based on static analysis and simplification of the graph
- In Answer Set Programming
  - Learning a "policy" based on past solution runs
- On other domains
  - Instruction scheduling on modern processors

Summary & Future works

- In product configuration,
  - problems are solved many times
  - solutions are not uniformly distributed in the search space
- Our proposal
  - Learn from past solver runs to find solutions more quickly
- Experiments
  - Conducted experiments with benchmark problems and a trivial value selection strategy
  - Results indicate the general feasibility
- Future work
  - Use more advanced strategies
  - However: consider cost of strategy application at run time

Announcement

- Upcoming Dagstuhl seminar on unifying Software and Product configuration
  - To take place in April 2014
  - Commonalities and differences
    - Feature models vs. configuration models, expressivity, reasoning, re-inventing the wheel
    - http://www.dagstuhl.de/14172
- See also
  - Arnaud Hubaux, Dietmar Jannach, Conrad Drescher, Leonardo Murta, Tomi Mannistö Krzysztof Czarnecki, Patrick Heymans, Tien Nguyen and Markus Zanker. **Unifying Software and Product Configuration: A Research Roadmap**. Configuration Workshop 2012

Thank you for your attention!

Questions?