Solving Retail Space Optimization Problem using the Randomized Search Algorithm

Kresimir Mihic[1], Andrew Vakhutinsky[2] and David Vengerov[1]

Oracle Labs[1] and Oracle RGBU[2]

April 21st, 2012

Randomized Search Algorithm

Case Study: Shelf Space Optimization Problem

Experimental Results

Summary

Randomized Search Algorithm

Case Study: Shelf Space Optimization Problem

Experimental Results

Summary

Randomized Search (RS) Algorithm

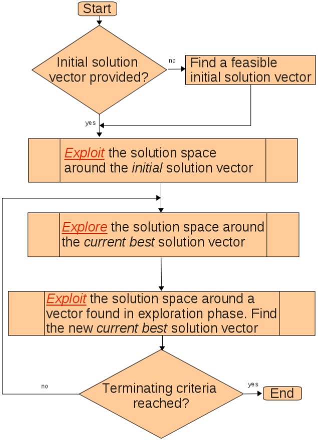
- Applies to hard combinatorial problems (solution space is finite)
- Especially suited for smooth surface problems:
  - That include complex constraints among the function's input and output variables.
  - That are non-linear and non-convex (the optimal solution does not need to be guaranteed)
- It builds on a stochastic nature of the Simulated Annealing (SA) methodology, but:
  - includes a mechanism for structural exploration of the solution space
  - derives its convergence criteria on a quality of the result rather than on a "temperature" schedule
  - does not recognize the concept of "temperature" what makes it easier to implement across a wide range of problems

ORACLE

The Big Picture

- The algorithm consists of two sequential phases, *exploration* and *exploitation* phase, that alternate until RS converges to some locally optimal solution or until maximum run time is reached.
- Each phase consists of repetitive cycles where components of the solution vector are considered in random order using uniform probability distribution.
- In the exploitation phase, the algorithm seeks to improve the current solution vector
- The exploration phase serves as a mean to "escape" locally optimal points.

RS Algorithm: Top View



Exploitation phase

1: let S_0 be the current solution vector.
2: **for** each component $i \in S_0$ (randomly chosen without replacement) **do**
3: among all the values allowed for the component i **find the value that satisfies constraints and maximizes** (minimizes) the objective value with all the other components unchanged. Set i to that value.
4: **end for**
5: repeat steps 2-4 if terminating criteria not reached

Exploration phase

1: let S_0 be the current solution vector.
2: **for** each component $i \in S_0$ (randomly chosen without replacement) **do**
3: **choose** a value from the set of all the values allowed for the component i **at** random.
4: accept the random value if it does not decrease the previously found best objective value by more than a specified percentage number.
5: if the new objective value $>$ the best objective value, return from the exploration phase
6: **end for**
7: return to the step 2 if terminating criteria not reached

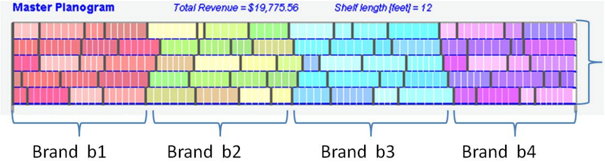
Randomized Search Algorithm

Case Study: Shelf Space Optimization Problem

Experimental Results

Summary

Shelf Space Optimization Problem

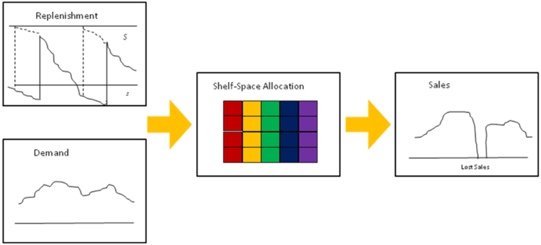


- Objectives:
  1. Determine shelf location and the number of facings for each item that would maximize a business criteria subject to the total shelf capacity, inventory replenishment constraints and adjacency rules.
  2. Minimize the total cost of changing the current layout.
- Constraints:
  - Shelf capacity
  - Category and brand boundaries
  - Item group adjacency
  - Shelf uniqueness

Constraints

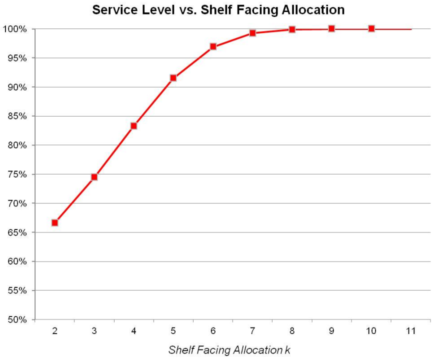
- Shelf capacity
  - Hard constraint. Specifies an upper bound on the total number of fixtures occupied by the items on the shelves
- Brand boundaries
  - Soft constraint. There is a certain tolerance associated with violating vertical brand alignment.
- Category boundaries
  - Categories can occupy only integer number of shelf fixtures. Any shelf space unoccupied by a category is wasted.
- Item adjacency
  - A group of items can be requested to be placed on the same shelf.
- Shelf uniqueness
  - An item can be assigned only to a single shelf.

Sales volume as the function of number of facings



- Given the replenishment policy and demand forecast, compute sales volume as a function of the number of facings lost sales are due to insufficient storage space
- Demand may depend on:
  - shelf position (e.g. eye level vs. bottom)
  - number of facings
- The volume as a function of facings increases with diminishing return

Service Level vs. Shelf Facing Allocation



- Space-aware assortment:
  - Start with some initial assortment for the store type
  - Drop items from the assortmen

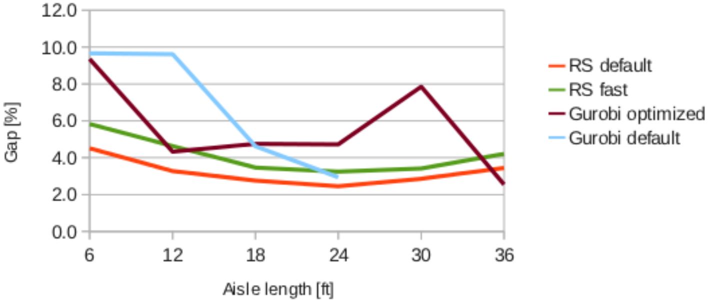
Randomized Search Algorithm

Case Study: Shelf Space Optimization Problem

Experimental Results

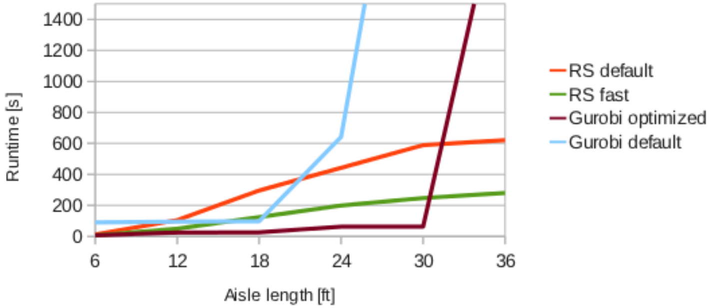
Summary

Experimental Results: Quality of the results



- Brand vertical alignment tolerance = 35 mm
- RS experiments repeated for 10 times each. Reporting mean values.
- “RS fast“ - Exploitation phase only. Exploration turned off
- "Gurobi optimized“ - Runtime parameters optimized for the ShelfSpace problem
- "Gurobi default“ - no solution found in 3600 seconds for aisle lengths $>$ 24 ft

Experimental Results: Runtime



- Brand vertical alignment tolerance = 35 mm
- RS experiments repeated for 10 times each. Reporting mean values.
- "RS fast" - Exploitation phase only. Exploration turned off
- "Gurobi optimized" - Runtime parameters optimized for the ShelfSpace problem
- "Gurobi default" - no solution found in 3600 seconds for aisle lengths > 24 ft

ORACLE

Experimental Results: Memory footprint

- RS has the same memory footprint across the different modes of operation (default, fast):
  - Max heap size (estimated from the JVM garbage collection) is below 300 KB
- Gurobi memory requirements are between 2.9 and 3.2 GB (found by using unix command "pmap -x “)
- Memory footprint measured across different aisle lengths and vertical aligment constraints

ORACLE

Randomized Search Algorithm

Case Study: Shelf Space Optimization Problem

Experimental Results

Summary

Summary

- RS is an algorithm for solving complex multi-dimensional combinatorial problems.
- There is no guarantee that the solution is the global optimum
- The algorithm uses internal structure of a problem to explore the search space and finds good solutions very quickly
- Shelf space optimizaton problem:
  - Implementation done in Java.
  - RS produces results of a better quality than the commercial solver (Gurobi) within comparable or shorter runtime.
  - Yields optimal solutions across a wide range of problem configurations without tuning.