RSSolver: A tool for solving large non-linear, non-convex discrete optimization problems

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

Oracle Labs[1] & Oracle Retail Business Unit[2]

October 12, 2012

RS Algorithm

RS Solver

Case Studies: Revenue Management Problems
- Regular Price Optimization Problem
- Shelf Space Optimization Problem

Summary

RS Solver

- A tool for solving hard combinatorial problems (solution space is finite):
  - 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)
- Built around Randomized Search (RS) algorithm
- Implemented in Java
- Supports modern, parallel, multi-threading implementation paradigm.
- Standardized I/O interface

RS Algorithm

RS Solver

Case Studies: Revenue Management Problems

Regular Price Optimization Problem

Shelf Space Optimization Problem

Summary

ORACLE

RS Algorithm

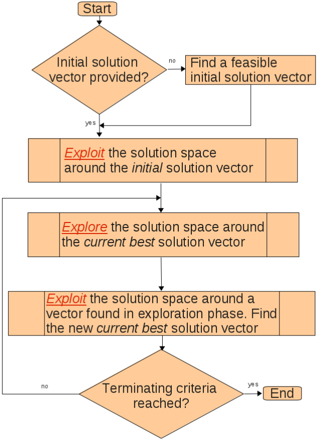
- It builds on a stochastic nature of the Simulated Annealing (SA) methodology, but:
  - includes a mechanism for structural exploration of the solution space
  - derives its convergence criteria on a quality of the result rather than on a "temperature" schedule
  - does not recognize the concept of "temperature" what makes it easier to implement across a wide range of problems
- Can be seen as a generalization of GRASP [?] algorithm.

ORACLE

RS Algorithm: 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

ORACLE

RS Algorithm

RS Solver

Case Studies: Revenue Management Problems

- Regular Price Optimization Problem
- Shelf Space Optimization Problem

Summary

ORACLE

RS Solver

- Implemented in Java
- Provides a standardized input/output interface:
  - RSDecisionVariable()
  - RSFunction()
  - RSObjective()
  - RSConstraint()
  - RSSolve()
- Enables users to extend the interface and define specialized objective and constraint functions
- Provides a set of easy to use run-time parameters that control quality and speed of the tool.

RS Solver: Decision Variables

- x = RSDecisionVariable(descriptor, domain, image)

$s_i = RSDecisionVariable($"shelf position", $\{2, 3, 4, 7, 8\}, \{0.9, 1.1, 1.0, 0.8\ 0.7\})$

- domain $= \{$allowed shelf positions for item $i\}$
- image $= \{$shelf coefficients$\}$

$n_i = RSDecisionVariable($"numberOfFacings", $\{0, 4, 6, 8\}, \{0, 0.8, 1.1, 2, 2.3\})$

- domain $= \{$allowed number of facings for item $i\}$
- image $= \{$demand$\}$

$p_i = RSDecisionVariable($"price", $\{5.09, 5.19, 5.49, 6.09\}, \{\})$

- domain $= \{$price ladder for item $i\}$
- image $=$ empty set

RS Solver: Functions

- Providing primitive algebraic, logic and set functions: sum(), max(), log(),..., ifThen(), or(),..., memberOf(), subsetOf(), atLeastNofM(),...
- Complex functions build using a composition principle
- Open interface: users can modify built-in functions and add their own specialized functions
- f = RSFunction(type,parameters)

  $f(x) = \sum_i c_i x_i$

  $$f(x) = RSFunction(\text{“dot”}, \{c_1, c_2, \ldots c_n\}, \{x_1, x_2, \ldots x_n\})$$

  $f(x,y,z) = g(x) - h(y_i) \cdot \max[l(z_i), k(y), g(x)] * l(y) + n(z)$

  $$m(x,y,z) = RSFunction(\text{“max”}, l(z), k(y), g(x))$$
  $$\tilde{m}(x,y,z) = RSFunction(\text{“times”}, h(y), m(x,y,z), l(y))$$
  $$f(x,y,z) = RSFunction(\text{“sum”}, g(x), \tilde{m}(x,y,z), n(z))$$

- Functions of different types can be combined

RS Solver: Objective

- f_0 = RSObjective(type,objective function)

$f_0 = RSObjective(\text{“maximize”}, revenueFunct)$

- $revenueFunct(price, demand) = RSFunction(....)$

$f_0 = RSObjective(\text{“mimimize”}, costFunct)$

- $costFunct(..) = RSFunction(....)$

ORACLE

RS Solver: Constraints

- c: RSConstraint(algebraic function, comparison operator, rhs value)
- l: RSConstraint(logic or set function, rhs boolean)

$c1 : f(x) \leq n$

- $c1 = RSConstraint(f(x), \text{“}<= \text{”}, n)$

$c2 : f(x) = k$

- $c1 = RSConstraint(f(x), \text{“}=\text{”}, k)$

$l1 : s(x) = true$

- $l1 = RSConstraint(s(x), true)$

ORACLE

RS Algorithm

RS Solver

Case Studies: Revenue Management Problems

- Regular Price Optimization Problem
- Shelf Space Optimization Problem

Summary

Regular Price Optimization (RPO) Problem

- Objective: Given a set of product items I we want to find a price for each item such that the objective function (margin, sales volume, revenue) is maximized.
  - Develop customized pricing strategies that address demographics and competitive characteristics of the store's trading area
  - Hold the retailer's image constant while adapting to neighborhood differences on demand
- Constraints:
  - Price constraints
  - Business constraints
  - Maximum number of items allowed to have their prices changed

ORACLE

Objective Value

- Supporting the multi-objective optimization by using the weights:

$$f_0 = W_v \cdot Volume + W_r \cdot Revenue + W_m \cdot Margin$$

- Revenue and Margin are functions of sales volume and the price vector
- Sales volume is a function of the price vector and "elasticity" matrix
- "Elasticity" matrix γ correlates prices along the items and defines how much a change in pricing of item i affects volume of item j:

$$V_i = V_i^0 \cdot \prod_{j \in I} \left(\frac{p_j}{p_j^0} \right)^{\gamma_{i,j}}, \forall i \in I$$

ORACLE

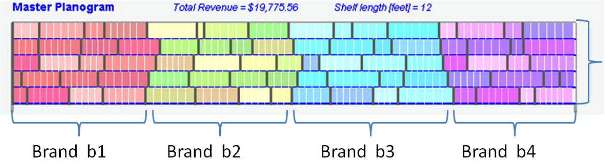
Experimental Results: Maximizing Margin

Test Set	Num. Items	Num. Constraints	Price Ladder Type
628-item-L	628	0	linearized
628-item-M	628	0	"magic" numbers
7D-hypercube	128	448	linearized
9D-hypercube	512	2304	linearized

Test Set	Gurobi		Randomized Search	
	Runtime (s)	Improvement (%)	Runtime (s)	Improvement (%)
628-item-L	5.4	6.4	5.1	6.4
628-item-M	243.6	9.0	5.7	17.7
7D-hypercube	1.1	5.6	7.5	4.5
9D-hypercube	21	5.4	34	4.7

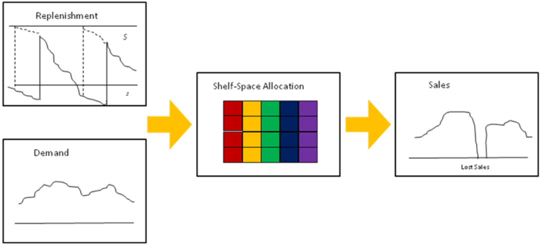
- Linearized price ladder example: $\{0.9p_0, 0.92p_0, ..., p_0, 1.02p_0, ..., 1.1p_0\}$
- "Magic" number price ladder example: {5.09,5.19,5.29,...,5.49}

Shelf Space Optimization Problem



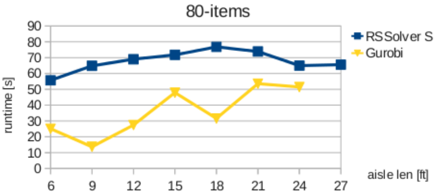
- Objectives:
    1. Determine shelf location and the number of facings for each item that would maximize a business criteria subject to the total shelf capacity, inventory replenishment constraints and adjacency rules.
    2. Minimize the total cost of changing the current layout.
- Constraints:
  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

Sales volume as the function of number of facings

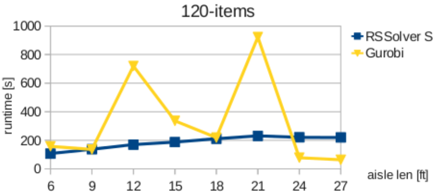


- Given the replenishment policy and demand forecast, compute sales volume as a function of the number of facings (lost sales are due to insufficient storage space)
- Demand may depend on:
  - shelf position (e.g. eye level vs. bottom)
  - number of facings
- The volume as a function of facings increases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

Experimental Results: Run-time

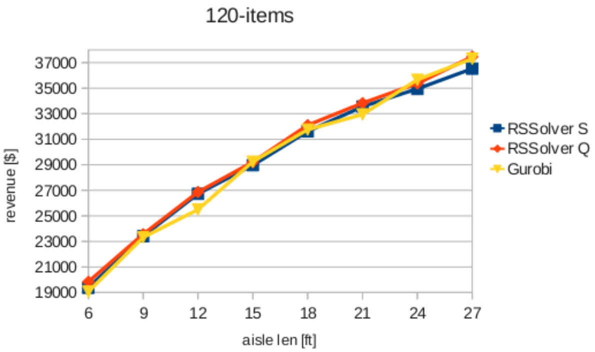


(a)



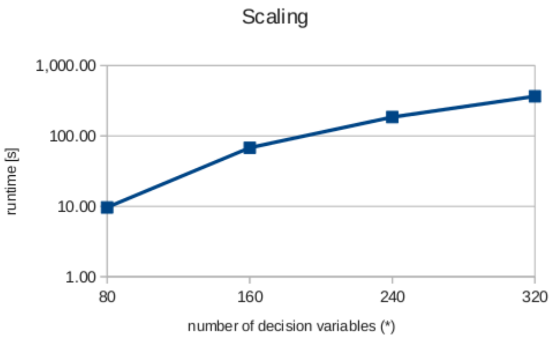
(b)

Experimental Results: Quality



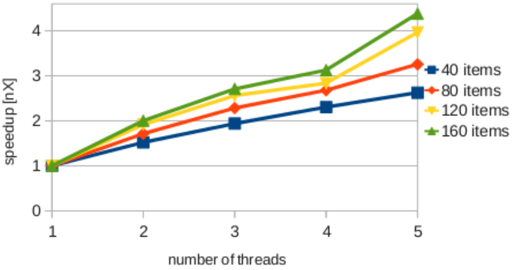
- RSSolver S: run-time parameters set for speed
- RSSolver Q: run-time parameters set for quality of results

Experimental Results: Scaling



- Number of decisions variables = 2xnumber of items (shelf position & number of facings per item)
- RSSolver S, single thread run, average runtime over different aisle lengths

Experimental Results: Multi-threading



- Max number of parallel tasks in this experiment is 5
- A single thread run needs 5 “loops” to execute 5 tasks
- For 2-thread run, RS executes 3 loops and for 3-thread and 4-thread 2 “loops“. 5-thread run is done in a single “loop”
- The difference in nThread=3 and nThread=4 speedup is the consequence of scheduling - individual tasks require different amount of time to be processed
- Difference in speedup with respect to number of items is due parallelization overhead: having more items reults in longer “loop” processing time, and the overhead becomes less significant.

ORACLE

Summary

- RSSolver is tool for solving complex multi-dimensional combinatorial problems.
- The tool implements RS algorithm that uses internal structure of a problem to explore the search space and finds good solutions very quickly
- Implementation done in Java programming language
- For reasonable run-time parameter settings, RS does not guarantee that the solution is the global optimum. The global optimum can be reached in time $t \to \inf$
- Execution time speedup scales almost linearly with number of threads
- Execution time scales polinomialy with number of variables
- Case studies show that RSSolver produces results of a simmilar or better quality then the commercial solver (Gurobi) within comparable or shorter run-time