Quadratic Programming Models in Strategic Sourcing Optimization

Daniel Ahlbom
Abstract

Quadratic Programming Models in Strategic Sourcing Optimization

Daniel Ahlbom

Strategic sourcing allows for optimizing purchases on a large scale. Depending on the requirements of the client and the offers provided for them, finding an optimal or even a near-optimal solution can become computationally hard. Mixed integer programming (MIP), where the problem is modeled as a set of linear expressions with an objective function for which an optimal solution results in a minimum objective value, is particularly suitable for finding competitive results. However, given the research and improvements continually being made for quadratic programming (QP), which allows for objective functions with quadratic expressions as well, comparing runtimes and objective values for finding optimal and approximate solutions is advised: for hard problems, applying the correct methods may decrease runtimes by several orders of magnitude. In this report, comparisons between MIP and QP models used in four different problems with three different solvers were made, measuring both optimization and approximation performance in terms of runtimes and objective values. Experiments showed that while QP holds an advantage over MIP in some cases, it is not consistently efficient enough to provide a significant improvement in comparison with, for example, using a different solver.
Acknowledgements

I would like to express my deepest gratitude to the following: Professor Arne Andersson at Trade Extensions, for being my supervisor and introducing me to a project with great depth and learning opportunities; Dr. Fredrik Ygge at Trade Extensions, for being an advisor providing the guidance, discussions and problems necessary for making the project interesting; and Professor Di Yuan at Uppsala University, for providing much needed insights and taking the time and effort to criticize this report into legibility.
# Contents

1 Introduction ................................................................. 4

2 Background ........................................................................ 5
   2.1 Computing Strategic Sourcing ........................................... 5
   2.2 Solving Computationally Hard Problems ........................... 5
   2.3 Mixed Integer Linear Programming .................................. 5
   2.4 Quadratic Programming ................................................ 6
   2.5 Approximation ............................................................ 6
   2.6 Modeling with AMPL ................................................... 6

3 Comparison of MIP and QP in Sourcing Problems ............... 9
   3.1 Seller Discounts .......................................................... 9
   3.2 Supply Restrictions ...................................................... 11
   3.3 Regional Division ......................................................... 16
   3.4 Reserved Price ........................................................... 20

4 Summary ............................................................................. 24
   4.1 Conclusion ................................................................. 24
   4.2 Future Work ............................................................... 24

References ............................................................................. 25

A Prerequisite knowledge ....................................................... 26
   A.1 NP-completeness ......................................................... 26
   A.2 Linear Algebra ............................................................ 26
   A.3 Convex Functions and Sets ............................................. 27

B Experiments ......................................................................... 28
   B.1 Testing platform .......................................................... 28
   B.2 Datasets ....................................................................... 28
   B.3 AMPL Implementations ............................................... 28
   B.4 Tables ......................................................................... 35
Glossary


**Constraint** Boundaries that a solution must satisfy in order to be valid.

**Decision variable** Variables for which the value has to be determined in order to find a solution.

**Gurobi** A solver suitable for MIP and QP optimization. It supports convex and non-convex problems with continuous and non-continuous variables in any combination for MIP, while QP requires that the problem is convex, or that any variable-by-variable multiplication to consist of a binary variable on either side. Official site as of June 27, 2017: [http://www.gurobi.com/](http://www.gurobi.com/).


**LP  *Linear Programming***: A solver technology where models are written using linear relations. Discussed in 2.3: Mixed Integer Linear Programming.


**MIP  *Mixed Integer Programming***: A solver technology where models are written using linear formula. In contrast with *Linear Programming* (LP), integers are allowed, which increases the difficulty of finding solutions. Discussed in 2.3: Mixed Integer Linear Programming.

**Model** A description of a problem suitable for a *solver*.

**Objective function** The function quantifying the quality of a solution. If the optimal solution is found by minimizing the objective function, a solution where the function returns the lowest possible value is optimal.

**Optimization problem** Problems where the goal is to find an optimal solution according to an objective function: this is in contrast with *satisfaction problem*.

**QP  *Quadratic Programming***: A solver technology where models are written using quadratic relations. Discussed in 2.4: Quadratic Programming.

**SAT  *Boolean Satisfiability Problem***: A satisfaction problems where a solution determines whether a boolean expression can be evaluated as true.

**Satisfaction problem** Problems where the goal is to find any solution: the aim of this is usually to prove feasibility, that is, whether a solution exists for a dataset.

**Solver** A program which accepts a model and finds an (optimal) solution: while there are many solver technologies, the ones used in this report are MIP and QP.

1 Introduction

When a business faces the issue of large-scale purchases, an optimization problem arises according to their requirements.

The requirements may involve factors such as lowering costs, reducing environmental impact and increasing production speeds. As the scale of their purchases increase, so does computation time: depending on how the problem is expressed, finding an optimal solution in a naive way may require hundreds of thousands of years, despite state-of-the-art computational power. However, as the scale increases, potential improvements according to requirements may increase as well. In this report, the use of strategic sourcing by expressing the problems for use with advanced algorithms allows for finding optimal solutions within feasible time-frames. The methods involve combinatorial optimization, by use of high-level models that are interpreted by solvers: software implementing sets of algorithms specialized in finding solutions to computationally complex problems. The solver softwares cover a spectrum of different problems and implementations, being continuously researched and improved. This motivates continuous re-evaluation of the most suitable methods of solving specific problems. While strategic sourcing problems allow for efficient use with MIP – expressing the problems in terms of linear relations with integer variables – they can be intuitively expressed in terms of QP, which allows for use of quadratic relations as well.

The aim of this report is to investigate the comparative efficiency of modern versions of MIP and QP solvers for strategic sourcing problems.
2 Background

2.1 Computing Strategic Sourcing

Strategic sourcing considers the purchase of products from a number of sellers. Large companies encounter sourcing problems of different kinds and complexity. In its simplest form, the problem requires no more effort than to find the best price for each product from each seller. This allows for problem solving in linear time: each price of each product only has to be considered once. In real-life situations, this method may prove to be infeasible. While an optimal solution could allow for a small purchase from hundreds of sellers, this may be highly impractical due to increased management complexity. Sellers may provide discounts. Some products may have regional restrictions.

Adding such restrictions to the problem risk making it necessary to consider combinations of sources, in order to find an optimal solution. This quickly increases computation time. For example, with \( s \) sellers and \( p \) products, there are \( p^s \) arrangements – given that each product is assigned at most single seller. At 15 products and 20 sellers, looking at each possible combination requires approximately 219523 years, \(^1\) given \( 4.8 \cdot 10^{10} \) solutions per second.\(^2\) On the other hand, finding an optimal solutions may provide significantly better results, according to a customer’s requirements. A case study – using similar methods to those discussed in this report – shows how Procter & Gamble reduced costs of display manufacturing by 48\% annually, translating to nearly $67 million, along with many other benefits [1].

Using an electronic reverse auction [2], sellers can bid their offers using online services; this provides data suitable for more advanced solution methods, in turn allowing for more clever problem solving methods.

2.2 Solving Computationally Hard Problems

When encountering problems for which naive algorithms lead to exponentially increasing computing times, solutions may be more efficiently found by constructing a model for use with a solver.

When using brute-force algorithms, NP-complete problems – see Appendix A.1 for a refresher – quickly become impossible to solve within useful timeframes in real-life instances. In order to solve such instances, much more efficient methods are required. One such method involves expressing the problem in terms of a different problem, in this context denoted a model. Any NP-complete problem can be reduced to some other NP-complete problem in polynomial time [3]. Modeling languages such as MiniZinc and A Mathematical Programming Language (AMPL) allow for expressing a problem such that it is interpreted by a solver — software specializing in solving specific problem types. There are many different solvers created for different problems by different companies. For example, SAT solvers such as MiniSAT [4] specialize in satisfiability of boolean expressions, while LP solvers, as used in this report, are particularly efficient in satisfying and finding optimal solutions for sets of linear relations. While using a solver allows for utilizing powerful algorithms with a minimal number of declarations, it is not necessarily trivial nor suitable to express a specific problem in terms of another.

The solvers technologies discussed in this report use MIP and QP.

2.3 Mixed Integer Linear Programming

A LP model of a linear optimization problem is expressed as follows [3,5,6]:

\[
\text{Minimize} \quad c^T x \tag{1} \\
\text{Subject to} \quad Ax \leq b \\
\quad x \geq 0 \tag{2}
\]

For a reminder of matrix and vector multiplication, see Appendix A.2. The terms are defined as follows:

- \( x \) is a vector of decision variables: the goal of the problem is achieved by finding the best possible combination of values for these.
- \( A \) is a matrix of known constants.
- \( b \) and \( c \) are vectors of known constants.
- (1) is the objective function.
- (2) are the constraints.

\(^1\) \( 4.8 \cdot 10^{10} \cdot 360 \cdot 24 \cdot 365.25 \approx 219522.9734728223 \)
\(^2\) For example, using a 4 Ghz 16-core processor.
When decision variables are required to be integers, it is denoted a MIP model. As efficient algorithms for LP rely on continuity of variables to find extreme points of polytopes [6], MIP tends to be significantly slower.

2.4 Quadratic Programming

The general form of a quadratic program is stated as follows [5, 6]:

\[
\begin{align*}
\text{Minimize} & \quad \frac{1}{2}x^T A x + x^T c \\
\text{Subject to} & \quad Ax \leq b \\
& \quad x \geq 0
\end{align*}
\]

(3) (4)

For a reminder of matrix and vector multiplication, see Appendix A.2. The terms are defined as follows:

- \( x \) is a vector of decision variables.
- \( A \in \mathbb{R}^{n \times n} \) is a symmetric matrix of known constants, where each side is the size of \( x \).
- \( b \) and \( c \) are vectors of known constants.
- (3) is the objective function.
- (4) are the constraints.

If \( A \) is not a positive definite matrix – discussed in Appendix A.3 – the objective function is non-convex, resulting problem becomes much more complex [5]. For example, solving with the interior point method, used by Artelys Knitro [7], does not guarantee finding the global optimum [8, 9] in this case. Certain solvers, such as Gurobi, allow for finding the global optimum in non-convex QP problems when the problem is formulated such that a multiplication \( x_1 \cdot x_2 \) restricts \( x_1 \) or \( x_2 \) to binary values [9].

2.5 Approximation

In certain cases, optimization problems can be solved such that a near-optimal solution can be found with significantly shorter runtimes.

Depending on the method, approximation can be accomplished within a proven percentage. For example, using a branch-and-bound algorithm, an accepted gap may be set according to the allowed value. Problem-specific approximation can also be applied. The discount problem discussed in 3.1: Seller Discounts, where sellers can provide a discount if a purchase surpasses a certain threshold, can be expressed approximately without having a discount at all. This allows for finding a solution in linear time, as the optimal solution would trivially be achieved by finding the lowest price of each product sold by each seller.

Integer relaxation is an approximation method that can indicate optimal solutions. This is done by considering all integer variables as continuous. In some cases, especially for MIP problems, this may significantly reduce runtimes. There are, however, cases where integer relaxation is not applicable. For example, constraints with binary variables are used in 3.4: Reserved Price in such that making the results of integer relaxation useful would generate a new problem of similar complexity.

2.6 Modeling with AMPL

AMPL [10] is a high-level language for constructing mathematical models for use with solvers.

While writing models according to the restrictions inherent in a solving technology may be difficult, understanding a model is fairly intuitive. It also allows for both MIP and QP models to be interpreted. It is therefore suitable for the purposes of this report.

2.6.1 File structure

AMPL’s file structure consists of a model file, as well as a dataset. This allows for a single model to be reused with any number of compatible sets of data. It also makes the model more manageable, as larger datasets may contain thousands of lines.
2.6.2 Syntax

Models are written using the following syntax:

- **param** defines a *parameter*, i.e., a constant for which a value is usually defined in a dataset.
- **set** defines a range of numbers, this is useful in the context of a for-all statement.
- **var** defines a *decision variable*.

Parameters and variables are defined as arrays by declaring the set it spans within curly brackets. For example, a two-dimensional array sharing sizes with sets *Products* and *Sellers* is declared using `{Products,Sellers}`.

Valid ranges for parameters and variables, as used in this report, are defined using `>`, `>=`, `<`, `<=`, `binary` (strictly greater than, greater than, less than, strictly less than and binary – `{0,1} values` – respectively).

- The *objective function* is written using `minimize` or `maximize`, followed by a function.
- Constraints are declared following the marker `subject to`. A “for all *i* in *S*” statement, is set using `{i in S}: (constraint)`. Both *objective function* and *constraints* can be prefixed by a name, preceding the : symbol, or for-all statement.
- Comments follow the hashtag symbol `#`.

2.6.3 Example

A short example demonstrates a model according to the following:

- **n_products** is the number of products.
- **n_sellers** is the number of sellers.
- **m_sellers** is the allowed number of sellers from which to order.
- **Price[i,j]** is the price of product *i* from seller *j*.
- The objective is to minimize the total cost of purchases.
- The problem is constrained by requiring all products to be purchased and the number of sellers.

To solve this, two sets of variables are used: **Order[i,j]** corresponds to the amount product *i* is ordered from seller *j* and **SellerOrder[j]** is set such that, if there is an order from seller *i*, **SellerOrder[i] = 1**. The resulting model is shown in Listing 1.

Listing 1: An example model written in AMPL

```AMPL
### CONSTANTS ###
param n_products; # Number of products
param n_sellers; # Number of sellers
param m_sellers; # Allowed number of sellers

# Sets allow convenient descriptions
set Products := 1..n_products; # Set of products
set Sellers := 1..n_sellers; # Set of sellers

# Price[i,j] is price of product i from seller j
param Price {Products,Sellers} >= 0;

### VARIABLES ###
var Order {Products,Sellers} >= 0, <= 1;

# SellerOrder[j] == 1 iff there is an order from seller j
var SellerOrder {Sellers} binary;
```
### OBJECTIVE ###

# Sum of all orders subtracted by sum of all discounts

\[
\text{minimize } \text{totalCost} : \sum_{i \in \text{Products}} \text{Order}[i,j] \times \text{Price}[i,j]
\]

### CONSTRAINTS ###

subject to

# Sum of orders for each product must exceed 1

\[
\text{eachProductOrdered } \{i \in \text{Products} \} : \sum_{j \in \text{Sellers}} \text{Order}[i,j] \geq 1
\]

# Satisfy order limit: sum of enlisted sellers must be less than limit.

\[
\text{countSellers} : \sum_{j \in \text{Sellers}} \text{SellerOrder}[j] \leq m_{\text{sellers}}
\]

# Mark sellers with orders: if \text{Order}[i,j] > 0, \text{SellerOrder}[j] must be 1

\[
\text{setSellerOrder } \{i \in \text{Products}, j \in \text{Sellers} \} : \text{SellerOrder}[j] \geq \text{Order}[i,j]
\]
3 Comparison of MIP and QP in Sourcing Problems

A comparison of MIP and QP was done by constructing a problem in 3.1: Seller Discounts, followed by three extensions providing a wider base of testing in a sourcing context.

The sourcing discount problem occurs in a context where a number of sellers offer products at a discounted price following a certain order size. This creates a problem of significantly greater complexity, than one where simply selecting the seller providing the lowest price for each product. Since the three problems following the first share many similarities, 3.1: Seller Discounts contains a more in-depth explanation.

All problems were implemented in AMPL followed by practical experiments calculating both exact and approximate solutions. The solvers used were Knitro (efficient for QP solving [5]), Gurobi (QP and MIP) and FICO Xpress (MIP only, due to solver limitations). Exact solutions were found using Gurobi and Xpress with default parameters. Approximative solutions for the problems were found using the following:

Max cross iterations An option in Knitro allowing for further optimization by use of the solver’s Cross Iteration algorithm. This can provide more exact results at cost of increased computation time.

Increased optimality gap An option in both Gurobi and Xpress setting the relative threshold by which a solution is accepted as optimal using the branch-and-bound algorithm. For example, a 5% optimality gap accepts a solution that is at most within 5% of optimum. The default value, in this context considered the global optimum, is at 0.01% in both solvers. This can also be set to an absolute value. Since Gurobi essentially uses the binary nature of the variables to convert QP models for use with MIP algorithms, the improvements made by increasing the optimality gap is very similar for both models. Therefore, only MIP models in Gurobi were tested using this parameter.

No discount condition A trivial modification of the model, removing all constraints and subtractions allowing for discounting of problems. Disregarding discounts can, in some instances, provide nearly optimal results while immensely simplifying the problem.

Integer relaxation (applicable in discount problem only) A solver option available in all solvers, allowing non-continuous variables to be regarded as continuous. This causes most constraints to be incorrectly fulfilled, which is why results are only applicable to the discount problem. While relaxation significantly improves runtime, most discount conditions are left unfulfilled in practice, leading to a higher actual cost compared to what the solver expects. An advanced improvement for this would require an auxiliary solution to attempt increasing purchases from sellers from which the discount is nearly applied.

3.1 Seller Discounts

Consider a seller discount problem as follows: a number of sellers provide products we have to purchase, where following specific thresholds a company may provide a discount. The objective is to minimize the total cost.

The problem can be defined according to the following constants:

- Let set $P$ represent all products such that $P = \{ i \in \mathbb{N} | 1 \leq i \leq |P| \}$.
- Let set $S$ represent all sellers such that $S = \{ j \in \mathbb{N} | 1 \leq j \leq |S| \}$.
- A two-dimensional array $C \in \mathbb{R}^{P \times |S|}$ describes cost $C_{i,j}$ for product $i$ when provided by seller $j$, where $C_{i,j} > 0$. If seller $i$ does not provide product $i$, the cost is set to $C_{i,j} = -1$.
- A two-dimensional array $D \in \mathbb{R}^{P \times |S|}$ describes the possible discount $D_{i,j}$ for product $i$ when provided by seller $j$, where $D_{i,j} > 0$.
- A two-dimensional array $D' = C - D$ describes the discounted price for each product, such that each element $D'_{i,j} = C_{i,j} - D_{i,j}$.
- A one-dimensional array $T \in \mathbb{R}^{S}$ contains the discount threshold $T_j \in T$ for each seller $j$, such that if the total cost of an order from $j$ exceeds $T_j$ a discount is provided for all purchases from $j$.

3.1.1 MIP Model

A linear formulation of this problem defines three variable arrays:

- A two-dimensional array $X \in \mathbb{R}^{P \times |S|}$ of real values $0 \leq X_{i,j} \leq 1$ where $X_{i,j}$ is the proportional amount of product $i$ ordered from seller $j$.
- A two-dimensional array $Y \in \mathbb{R}^{P \times |S|}$ of real values $0 \leq Y_{i,j} \leq 1$ where $Y_{i,j} = X_{i,j}$ if the discount is applied for seller $j$ and product $i$ is purchased from them.
A one-dimensional array $Z \in \mathbb{N}^{|S|}$ of binary values $Z_j \in \{0, 1\}$ where $Z_j = 1$ if the discount is applied for seller $j$.

This allows for a description of the problem as follows:

\[
\text{Minimize} \quad \sum_{i \in P} \sum_{j \in S} C_{i,j} \cdot X_{i,j} - \sum_{i \in P} \sum_{j \in S} D_{i,j} \cdot Y_{i,j} \tag{5}
\]

\[
\text{Subject to} \quad \forall i \in P \left( \sum_{j \in S} X_{i,j} \geq 1 \right) \tag{6}
\]

\[
\forall j \in S \left( \sum_{i \in P} C_{i,j} \cdot X_{i,j} - Z_j \cdot T_j \geq 0 \right) \tag{7}
\]

\[
\forall i \in P, j \in S \left( X_{i,j} - Y_{i,j} \geq 0 \right) \tag{8}
\]

\[
\forall i \in P, j \in S \left( Z_j - Y_{i,j} \geq 0 \right) \tag{9}
\]

\[
\forall i \in P, j \in S \left( C_{i,j} \cdot X_{i,j} \geq 0 \right) \tag{10}
\]

Where the relations define the following:

(5) is the objective, i.e., total cost, found by the sum of all orders subtracted by the sum of all discounts.

(6) ensures each product is ordered, by ensuring that the sum of all orders of each product is at least 1.

(7) is the discount condition: for each seller $j$, the sum of the order must be at least that of their thresholds multiplied by the binary variable $Z_j$.

(8) sets that discounts are only provided for purchased products: the condition only holds if the discounted amount is at most the ordered amount. The more intuitive statement $X_{i,j} = Y_{i,j}$ would not hold if no discount is provided.

(9) assures discounts are only provided for a product iff the discount condition for the seller is met: if no discount is provided, i.e., $Z_j = 0$, then $Y_{i,j} = 0$ as well.

(10) sets that purchases can only be made when a seller provides the product: if a product $i$ is not provided by seller $j$, then $C_{i,j} = -1$. Therefore, $C_{i,j} \cdot X_{i,j} < 0$ if $X_{i,j} > 0$, breaking the constraint. To satisfy the constraint in such a case, $X_{i,j} = 0$.

### 3.1.2 QP Model

A quadratic formulation of this problem defines two variable arrays:

- A two-dimensional array $X \in \mathbb{R}^{P \times |S|}$ of real values $0 \leq X_{i,j} \leq 1$ where $X_{i,j}$ is the proportional amount of product $i$ ordered from seller $j$. This corresponds to the variable of the same name in the MIP formulation.

- A one-dimensional array $Y \in \mathbb{N}^{|S|}$ of binary values $Y_j \in \{0, 1\}$ where $Y_j = 1$ if the discount is not applied for seller $j$.

This allows us to describe the problem as follows, with many similarities to the 3.1.1: MIP Model:

\[
\text{Minimize} \quad \sum_{i \in P} \sum_{j \in S} D'_{i,j} \cdot X_{i,j} + \sum_{i \in P} \sum_{j \in S} D_{i,j} \cdot X_{i,j} \cdot Y_j \tag{11}
\]

\[
\text{Subject to} \quad \forall i \in P \left( \sum_{j \in S} X_{i,j} \geq 1 \right) \tag{12}
\]

\[
\forall j \in S \left( \sum_{i \in P} D'_{i,j} \cdot X_{i,j} + \sum_{i \in P} D_{i,j} \cdot X_{i,j} - T_j \cdot (1 - Y_j) \geq 0 \right) \tag{13}
\]

\[
\forall i \in P, j \in S \left( D'_{i,j} \cdot X_{i,j} \geq 0 \right) \tag{14}
\]

(11) is the objective, i.e., total cost: this is calculated by the sum of discounted prices and the corresponding discounts (equivalent of difference to the full cost).

(12) ensures each product is ordered, corresponding to (6).

(13) is the discount condition: contrary to (7), if the total order of a seller $i$ does not exceed the threshold, $Y_j$ must be set to 0.

(14) sets that purchases can only be made when a seller provides the product, corresponding to (10).
Comparison of MIP and QP in Sourcing Problems

3.1.3 Implementation

Models for each solution was written in AMPL and are shown in Listing 2 and Listing 3, under Appendix B.3 for the MIP solution and QP solution respectively.

3.1.4 Results

Figure 1 shows the sum of objective values for each solver method, allowing for an overview of how well the approximations compare to exact solutions. Table 1 shows the relative accuracy of each solution, allowing for a similar comparison at greater detail. Figure 2 allows for comparison of runtimes from the various methods.

The entirety of test results are shown in Table 5, under Appendix B.4.

3.1.5 Discussion

The results in Figure 2 show that there is some benefit in finding global optima using the QP model with Gurobi. The runtime improvements do however not differ by orders of magnitude, with the exception of a single tested instance. However, the difference in different solvers can affect runtimes more than the models: specifically, in instances p100-s140-A as well as p100-s140-C, Gurobi finds an optimal solution significantly quicker than Xpress.

Approximate solutions provide a negligible benefit for QP solving with Knitro. In this problem, crossover iterations do not significantly impact results. As shown in Figure 1, accuracy is only slightly better than Xpress solutions with 10% accuracy, while the runtimes do not differ significantly. Table 1 shows a more fine-grained comparison between the two: the difference in accuracy for respective solutions vary by problem instance, suggesting that neither method is uniformly superior.

When relaxing MIP variables, the model allows for mistakenly applying discounts that are invalid. This, in turn, provides highly inaccurate results. A possible solution for this would be to fine-tune the resulting solution by transferring purchases from sellers further from having their discounts applied, to those nearly reaching their discount thresholds.

3.2 Supply Restrictions

Not all sellers can be expected to provide an infinite supply. This may be reflected in a problem that restricts the size of purchases of a product from a specific seller.
Comparison of MIP and QP in Sourcing Problems

Daniel Ahlbom, Uppsala University 2017

Application runtimes for the seller discount problem

![Graph showing runtimes in seconds for all tested datasets.](image)

Figure 2: Runtimes in seconds for all tested datasets, where pA \times B - C has A products and B sellers for an instance C. Gurobi (QP), Gurobi (MIP), Xpress (MIP) found proven optimality and have filled markers. Lower values are better. The figure is provided in logarithmic scale as different methods provide results that differ in several orders of magnitude. Knitro runtimes are similar, resulting in overlaps in the figure. The same is true for the two different gap settings in Gurobi. Exact results are shown in Table 5.

### Dataset Runtimes (s)

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Knitro, no crossover it.</th>
<th>Knitro, max 1 crossover it.</th>
<th>Knitro, max 2 crossover it.</th>
</tr>
</thead>
<tbody>
<tr>
<td>p100-s140-A</td>
<td>7.24%</td>
<td>8.89%</td>
<td>9.16%</td>
</tr>
<tr>
<td>p100-s140-B</td>
<td>7.24%</td>
<td>8.89%</td>
<td>9.16%</td>
</tr>
<tr>
<td>p100-s140-C</td>
<td>5.33%</td>
<td>8.89%</td>
<td>9.16%</td>
</tr>
<tr>
<td>p100-s80-A</td>
<td>1.70%</td>
<td>0.80%</td>
<td>2.61%</td>
</tr>
<tr>
<td>p100-s80-B</td>
<td>1.70%</td>
<td>0.80%</td>
<td>2.61%</td>
</tr>
<tr>
<td>p100-s80-C</td>
<td>1.70%</td>
<td>0.80%</td>
<td>2.61%</td>
</tr>
<tr>
<td>p140-s100-A</td>
<td>26.97%</td>
<td>26.98%</td>
<td>27.35%</td>
</tr>
<tr>
<td>p140-s100-B</td>
<td>26.97%</td>
<td>26.98%</td>
<td>27.35%</td>
</tr>
<tr>
<td>p140-s100-C</td>
<td>26.97%</td>
<td>26.98%</td>
<td>27.35%</td>
</tr>
<tr>
<td>p140-s120-A</td>
<td>26.97%</td>
<td>26.98%</td>
<td>27.35%</td>
</tr>
<tr>
<td>p140-s120-B</td>
<td>26.97%</td>
<td>26.98%</td>
<td>27.35%</td>
</tr>
<tr>
<td>p140-s120-C</td>
<td>26.97%</td>
<td>26.98%</td>
<td>27.35%</td>
</tr>
</tbody>
</table>

### Table 1: Accuracy of the objective values of the seller discount problem.

Each cell shows how much larger the objective value of a solution is compared to the proven optimality of each dataset. An optimum solution shows exactly 0%, while larger values suggest less accuracy.
Additionally, a seller may restrict their dependency on a specific company: another requirement restricting the total order from each seller is therefore added.

In addition to the constants defined in 3.1: Seller Discounts, we add the following, including the modified array \( C \):

- A two-dimensional array \( A \) describes the possible amount \( A_{i,j} \) of a product \( i \) sold by a seller \( j \), where \( A_{i,j} = 1 \) suggests the full required amount can be purchased, while \( A_{i,j} = 0 \) means that the seller does not provide the product at all.

- A one-dimensional array \( B \) describes the total price \( B_j \) a seller \( j \) will accept.

\subsection{MIP Model}

We use the same decision variables as the 3.1.1: MIP Model for 3.1: Seller Discounts. This allows us to describe the problem as follows:

\[
\text{Minimize } \sum_{i \in P} \sum_{j \in S} C_{i,j} \cdot X_{i,j} - \sum_{i \in P} \sum_{j \in S} D_{i,j} \cdot Y_{i,j} \tag{15}
\]

\[
\text{Subject to } \forall i \in P \left( \sum_{j \in S} X_{i,j} \geq 1 \right) \tag{16}
\]

\[
\forall j \in S \left( \sum_{i \in P} C_{i,j} \cdot X_{i,j} - Z_{j} \cdot T_{j} \geq 0 \right) \tag{17}
\]

\[
\forall i \in P, j \in S \left( X_{i,j} - Y_{i,j} \geq 0 \right) \tag{18}
\]

\[
\forall i \in P, j \in S \left( Z_{j} - Y_{i,j} \geq 0 \right) \tag{19}
\]

\[
\forall i \in P, j \in S \left( A_{i,j} - X_{i,j} \geq 0 \right) \tag{20}
\]

\[
\forall j \in S \left( B_j - \sum_{i \in P} C_{i,j} \cdot X_{i,j} \geq 0 \right) \tag{21}
\]

Where the relations define the following:

(15) is the objective, i.e., total cost.

(16) ensures each product is ordered.

(17) is the discount condition.

(18) sets that discounts are only provided for purchased products.

(19) assures discounts are only provided for a product \( \text{iff} \) the discount condition for the seller is met.

(20) limits purchases according to supply: this replaces the condition whether an item is sold at all.

(21) limits purchases according to permitted level.

\subsection{QP Model}

We use the same decision variables as 3.1.2: QP Model for 3.1: Seller Discounts., which describes the problem using the following statements.

\[
\text{Minimize } \sum_{i \in P} \sum_{j \in S} D_{i,j} \cdot X_{i,j} + \sum_{i \in P} \sum_{j \in S} D_{i,j} \cdot X_{i,j} \cdot Y_{j} \tag{22}
\]

\[
\text{Subject to } \forall i \in P \left( \sum_{j \in S} X_{i,j} \geq 1 \right) \tag{23}
\]

\[
\forall j \in S \left( \sum_{i \in P} D_{i,j} \cdot X_{i,j} + \sum_{i \in P} D_{i,j} \cdot X_{i,j} - T_{j} \cdot (1 - Y_{j}) \geq 0 \right) \tag{24}
\]

\[
\forall i \in P, j \in S \left( A_{i,j} - X_{i,j} \geq 0 \right) \tag{25}
\]

\[
\forall j \in S \left( B_j - \sum_{i \in P} D_{i,j} \cdot X_{i,j} - \sum_{i \in P} D_{i,j} \cdot X_{i,j} \cdot Y_{j} \geq 0 \right) \tag{26}
\]

Where each statement considers the following:
Knitro (QP), mcit1
Knitro (QP), mcit2
Gurobi (QP)
Gurobi (MIP)
Gurobi (MIP), 5% gap
Gurobi (MIP), 10% gap
Gurobi (MIP), no discount
Xpress (MIP)
Xpress (MIP), 5% gap
Xpress (MIP), 10% gap
Xpress (MIP), no discount

Figure 3: Sum of objective values for all tested datasets in the supply problem, where pA-sB-C consists of A products, B sellers in instance C. Gurobi (QP), Gurobi (MIP) and Xpress (MIP) show proven optimality. Lower values are better.

(22) is the objective, i.e. minimizing total cost.
(23) ensures each product is ordered.
(24) is the discount condition.
(25) limits purchases according to supply: this replaces the condition whether an item is sold at all.
(26) limits purchases according to permitted level.

3.2.3 Implementation

Models for each solution were written in AMPL. Models are shown in Listing 4 and Listing 5 for the MIP solution and QP solution respectively. The approximative models differ only by setting the binary variable to continuous.

3.2.4 Results

Figure 3 shows the sum of objective values for each solver method, allowing for an overview of how well the approximations compare to exact solutions. Table 2 shows the relative accuracy of each solution, allowing for a similar comparison at greater detail. Figure 4 allows for comparison of runtimes from the various methods.

The entirety of test results are shown in Table 6, under Appendix B.4.

3.2.5 Discussion

Similar to in 3.1: Seller Discounts, results for this problem show that solving without discounts requires significantly less time than finding optimal solutions, while other approximations are placed somewhere in between.

Figure 3 shows that approximate solutions found with Knitro roughly correspond to objective values between the 5% and 10% gap tolerances tested with MIP solvers. Runtimes using Knitro are consistently slower than for corresponding MIP approximations in datasets with 60 products and 100 sellers, while the opposite is true for datasets with 100 products and 80 sellers. No discernable pattern is apparent for the remaining datasets. QP models in Gurobi appear to be the consistently quickest way to find optimal results, with small margins in most cases.

It appears that neither QP nor MIP can provide a significant advantage in the seller supply problem.
### Application runtimes for the supply problem

![Runtimes in seconds for all tested datasets](image)

**Figure 4:** Runtimes in seconds for all tested datasets, where pAsB-C has A products and B sellers for an instance C. Gurobi (QP), Gurobi (MIP), Xpress (MIP) found proven optimality and have filled markers. Lower values are better. The figure is provided in logarithmic scale as different methods provide results that differ in several orders of magnitude.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>p60-s100-A</th>
<th>p60-s100-B</th>
<th>p60-s100-C</th>
<th>p80-s60-A</th>
<th>p80-s60-B</th>
<th>p80-s60-C</th>
<th>p100-s60-A</th>
<th>p100-s60-B</th>
<th>p100-s60-C</th>
<th>p100-s80-A</th>
<th>p100-s80-B</th>
<th>p100-s80-C</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Knitro</strong></td>
<td>None</td>
<td>1.82% 2.58%</td>
<td>4.17% 3.84%</td>
<td>3.40% 3.03%</td>
<td>1.96% 2.4%</td>
<td>3.99% 1.14%</td>
<td>2.56% 2.1%</td>
<td>2.56% 2.1%</td>
<td>1.83% 1.83%</td>
<td>1.83% 1.83%</td>
<td>1.83% 1.83%</td>
<td>1.83% 1.83%</td>
</tr>
<tr>
<td></td>
<td>Max 1 cross it</td>
<td>1.82% 2.58%</td>
<td>4.17% 3.84%</td>
<td>3.40% 3.03%</td>
<td>1.96% 2.4%</td>
<td>3.99% 1.14%</td>
<td>2.56% 2.1%</td>
<td>2.56% 2.1%</td>
<td>1.83% 1.83%</td>
<td>1.83% 1.83%</td>
<td>1.83% 1.83%</td>
<td>1.83% 1.83%</td>
</tr>
<tr>
<td></td>
<td>Max 2 cross it</td>
<td>1.82% 2.58%</td>
<td>4.17% 3.84%</td>
<td>3.40% 3.03%</td>
<td>1.96% 2.4%</td>
<td>3.99% 1.14%</td>
<td>2.56% 2.1%</td>
<td>2.56% 2.1%</td>
<td>1.83% 1.83%</td>
<td>1.83% 1.83%</td>
<td>1.83% 1.83%</td>
<td>1.83% 1.83%</td>
</tr>
<tr>
<td><strong>Gurobi</strong></td>
<td>QP</td>
<td>0.00% 0.00%</td>
<td>0.00% 0.00%</td>
<td>0.00% 0.00%</td>
<td>0.00% 0.00%</td>
<td>0.00% 0.00%</td>
<td>0.00% 0.00%</td>
<td>0.00% 0.00%</td>
<td>0.00% 0.00%</td>
<td>0.00% 0.00%</td>
<td>0.00% 0.00%</td>
<td>0.00% 0.00%</td>
</tr>
<tr>
<td></td>
<td>MIP, int.</td>
<td>0.00% 0.00%</td>
<td>0.00% 0.00%</td>
<td>0.00% 0.00%</td>
<td>0.00% 0.00%</td>
<td>0.00% 0.00%</td>
<td>0.00% 0.00%</td>
<td>0.00% 0.00%</td>
<td>0.00% 0.00%</td>
<td>0.00% 0.00%</td>
<td>0.00% 0.00%</td>
<td>0.00% 0.00%</td>
</tr>
<tr>
<td></td>
<td>MIP, 5% gap</td>
<td>2.55% 0.70%</td>
<td>0.87% 1.00%</td>
<td>0.26% 0.30%</td>
<td>0.02% 0.03%</td>
<td>0.00% 0.00%</td>
<td>0.18% 0.21%</td>
<td>0.00% 0.00%</td>
<td>0.18% 0.21%</td>
<td>0.00% 0.00%</td>
<td>0.18% 0.21%</td>
<td>0.00% 0.00%</td>
</tr>
<tr>
<td></td>
<td>MIP, 10% gap</td>
<td>2.55% 0.70%</td>
<td>0.87% 1.00%</td>
<td>0.26% 0.30%</td>
<td>0.02% 0.03%</td>
<td>0.00% 0.00%</td>
<td>0.18% 0.21%</td>
<td>0.00% 0.00%</td>
<td>0.18% 0.21%</td>
<td>0.00% 0.00%</td>
<td>0.18% 0.21%</td>
<td>0.00% 0.00%</td>
</tr>
<tr>
<td></td>
<td>MIP, no disc.</td>
<td>2.55% 0.70%</td>
<td>0.87% 1.00%</td>
<td>0.26% 0.30%</td>
<td>0.02% 0.03%</td>
<td>0.00% 0.00%</td>
<td>0.18% 0.21%</td>
<td>0.00% 0.00%</td>
<td>0.18% 0.21%</td>
<td>0.00% 0.00%</td>
<td>0.18% 0.21%</td>
<td>0.00% 0.00%</td>
</tr>
<tr>
<td><strong>Xpress</strong></td>
<td>None</td>
<td>0.00% 0.00%</td>
<td>0.00% 0.00%</td>
<td>0.00% 0.00%</td>
<td>0.00% 0.00%</td>
<td>0.00% 0.00%</td>
<td>0.00% 0.00%</td>
<td>0.00% 0.00%</td>
<td>0.00% 0.00%</td>
<td>0.00% 0.00%</td>
<td>0.00% 0.00%</td>
<td>0.00% 0.00%</td>
</tr>
<tr>
<td></td>
<td>5% gap</td>
<td>2.55% 0.70%</td>
<td>0.87% 1.00%</td>
<td>0.26% 0.30%</td>
<td>0.02% 0.03%</td>
<td>0.00% 0.00%</td>
<td>0.18% 0.21%</td>
<td>0.00% 0.00%</td>
<td>0.18% 0.21%</td>
<td>0.00% 0.00%</td>
<td>0.18% 0.21%</td>
<td>0.00% 0.00%</td>
</tr>
<tr>
<td></td>
<td>10% gap</td>
<td>2.55% 0.70%</td>
<td>0.87% 1.00%</td>
<td>0.26% 0.30%</td>
<td>0.02% 0.03%</td>
<td>0.00% 0.00%</td>
<td>0.18% 0.21%</td>
<td>0.00% 0.00%</td>
<td>0.18% 0.21%</td>
<td>0.00% 0.00%</td>
<td>0.18% 0.21%</td>
<td>0.00% 0.00%</td>
</tr>
<tr>
<td></td>
<td>No disc.</td>
<td>24.42% 23.53%</td>
<td>24.41% 23.69%</td>
<td>22.64% 23.76%</td>
<td>22.89% 22.76%</td>
<td>22.30% 22.66%</td>
<td>22.96% 22.66%</td>
<td>22.96% 22.66%</td>
<td>22.96% 22.66%</td>
<td>22.96% 22.66%</td>
<td>22.96% 22.66%</td>
<td>22.96% 22.66%</td>
</tr>
</tbody>
</table>

|------------|---------|---------|---------|---------|---------|---------|---------|---------|---------|---------|---------|---------|

Table 2: Accuracy of the objective values of the seller supply problem. Each cell shows how much larger the objective value of a solution is compared to the proven optimality of each dataset. An optimum solution shows exactly 0%, while larger values suggest less accuracy.
3.3 Regional Division

Ordering products from a seller brings some overhead: an optimized minimum cost may result in an impractical number of sellers to manage.

This problem is of particular interest when a company requires purchases to be made for a larger amount of regions: for example, a small number of national sellers can be more manageable than a large number of local sellers. This is, however, not always a possibility: for example, there may be a policy that requires a certain class of wares to be locally produced.

Another interpretation may be in a multinational context: global corporations may deliver products to certain countries, but not all.

The problem can be defined according to the following constants:

- Let set $P$ represent all products such that $P = \{ i \in \mathbb{N} \mid 1 \leq i \leq \left| P \right| \}$.
- Let set $S$ represent all sellers such that $S = \{ j \in \mathbb{N} \mid 1 \leq j \leq \left| S \right| \}$.
- Let set $R$ represent all regions such that $R = \{ k \in \mathbb{N} \mid 1 \leq k \leq \left| R \right| \}$.
- A three-dimensional array $C \in \mathbb{R}^{[P] \times [S] \times [R]}$ describes cost $C_{i,j,k}$ for product $i$ when provided by seller $j$, where $C_{i,j,k} > 0$. If seller $i$ does not provide product $i$, the cost is set to $C_{i,j,k} = -1$.
- A three-dimensional array $D \in \mathbb{R}^{[P] \times [S] \times [R]}$ describes the possible discount $D_{i,j,k}$ for product $i$ when provided by seller $j$, where $D_{i,j,k} > 0$.
- A three-dimensional array $D' = C - D$ describes the discounted price for each product, such that each element $D'_{i,j,k} = C_{i,j,k} - D_{i,j,k}$.
- A two-dimensional array $Q$ of size $|P| \times |R|$ describes the number of units $Q_{i,k}$ required of a product $i$ in region $k$. If $Q_{i,k} = 0$, the product is not required in this region.
- A one-dimensional array $L$ of size $|R|$ describes the limit $L_k$ of total sellers serving a single region $k$.
- A constant $l$ describes the total amount of sellers allowed across all regions.

3.3.1 MIP Model

We use decision variables similar to the 3.1.1: MIP Model for 3.1: Seller Discounts.

- A three-dimensional array $X \in \mathbb{R}^{[P] \times [S] \times [R]}$ of real values $0 \leq X_{i,j}$ where $X_{i,j,k}$ is the ordered amount of product $i$ from seller $j$ in region $k$.
- A three-dimensional array $Y \in \mathbb{R}^{[P] \times [S] \times [R]}$ of real values $0 \leq Y_{i,j}$ where $Y_{i,j,k} = X_{i,j,k}$ if the discount is applied for seller $j$ and product $i$ is purchased from them in region $k$.
- A one-dimensional array $Z \in \mathbb{N}^{[S]}$ of binary values $Z_j \in \{ 0, 1 \}$ where $Z_j = 1$ if the discount is applied for seller $j$.
- A two-dimensional array $U \in \mathbb{N}^{[S] \times [R]}$ of binary values $U_{j,k} \in \{ 0, 1 \}$ where $U_{j,k} = 1$ if seller $j$ has an order in region $k$.
- A one-dimensional array $V \in \mathbb{N}^{[S]}$ of binary values $V_j \in \{ 0, 1 \}$ where $V_j = 1$ if seller $j$ has any order.

This allows us to describe the problem as follows:
Minimize \[ \sum_{i \in P} \sum_{j \in S} \sum_{k \in R} C_{i,j,k} \cdot X_{i,j,k} - \sum_{i \in P} \sum_{j \in S} \sum_{k \in R} D_{i,j,k} \cdot Y_{i,j,k} \]  

Subject to \[ \forall i \in P, \forall k \in R \left( \sum_{j \in S} X_{i,j,k} - Q_{i,k} \geq 0 \right) \]  
\[ \forall j \in S \left( \sum_{i \in P} \sum_{k \in R} C_{i,j,k} \cdot X_{i,j,k} - Z_j \cdot T_j \geq 0 \right) \]  
\[ \forall i \in P, j \in S, k \in R \left( X_{i,j,k} - Y_{i,j,k} \geq 0 \right) \]  
\[ \forall i \in P, j \in S, k \in R \left( C_{i,j,k} \cdot X_{i,j,k} \geq 0 \right) \]  
\[ \forall i \in P, j \in S, k \in R \left( U_{j,k} \cdot Q_{i,k} - X_{i,j,k} \geq 0 \right) \]  
\[ \forall k \in R \left( \sum_{j \in S} U_{j,k} \cdot Q_{i,k} - X_{i,j,k} \geq 0 \right) \]  
\[ \forall j \in S, k \in R \left( V_j - U_{j,k} \geq 0 \right) \]  
\[ \forall j \in S \left( V_j - l \leq 0 \right) \]

Where the relations define the following:

(27) is the objective, i.e., total cost.
(28) ensures each product is ordered.
(29) is the discount condition.
(30) sets that discounts are only provided for purchased products.
(31) assures discounts are only provided for a product if the discount condition for the seller is met.
(32) sets that purchases can only be made when a seller provides the product.
(33) sets whether a seller has an order in a region.
(34) limits sellers in a region.
(35) sets whether a seller has any order.
(36) limits total number of sellers.

### 3.3.2 QP Model

We use decision variables similar to the 3.1.2: QP Model for 3.1: Seller Discounts.

- A three-dimensional array \( X \in \mathbb{R}^{[P] \times [S] \times [R]} \) of real values \( 0 \leq X_{i,j,k} \) where \( X_{i,j,k} \) is the ordered amount of product \( i \) from seller \( j \) in region \( k \).

- A one-dimensional array \( Y \in \mathbb{N}^{[S]} \) of binary values \( Y_j \in \{0, 1\} \) where \( Y_j = 1 \) if the discount is not applied for seller \( j \).

- A two-dimensional array \( U \in \mathbb{N}^{[S] \times [R]} \) of binary values \( U_{j,k} \in \{0, 1\} \) where \( U_{j,k} = 1 \) if seller \( j \) has an order in region \( k \).

- A one-dimensional array \( V \in \mathbb{N}^{[S]} \) of binary values \( V_j \in \{0, 1\} \) where \( V_j = 1 \) if seller \( j \) has any order.
Minimize
\[ \sum_{i \in P} \sum_{j \in S} \sum_{k \in R} D'_{i,j,k} \cdot X_{i,j,k} + \sum_{i \in P} \sum_{j \in S} \sum_{k \in R} D_{i,j,k} \cdot X_{i,j,k} \cdot Y_j \] (37)

Subject to
\( \forall i \in P, \forall k \in R \left( \sum_{j \in S} X_{i,j,k} - Q_{i,k} \geq 0 \right) \) (38)
\( \forall j \in S \left( \sum_{i \in P} \sum_{k \in R} (D'_{i,j,k} + D_{i,j,k}) \cdot X_{i,j,k} - T_j \cdot (1 - Y_j) \geq 0 \right) \) (39)
\( \forall i \in P, j \in S, k \in R \left( D'_{i,j,k} \cdot X_{i,j,k} \geq 0 \right) \) (40)
\( \forall i \in P, j \in S, k \in R \left( U_{j,k} \cdot Q_{i,k} - X_{i,j,k} \geq 0 \right) \) (41)
\( \forall k \in R \left( \sum_{j \in S} U_{j,k} - L_k \leq 0 \right) \) (42)
\( \forall j \in S, k \in R \left( V_j - U_{j,k} \geq 0 \right) \) (43)
\( \forall j \in S \left( V_j - l \leq 0 \right) \) (44)

(37) is the objective, i.e. total cost.

(38) ensures each product is ordered.

(39) is the discount condition.

(40) sets that purchases can only be made when a seller provides the product.

(41) sets whether a seller has an order in a region.

(42) limits sellers in a region.

(43) sets whether a seller has any order.

(44) limits total number of sellers.

3.3.3 Implementation

Models for each solution were written in AMPL. The models are shown in Listing 6 and Listing 7 for the MIP solution and QP solution respectively. The approximative models differ only by setting the binary variable to continuous.

3.3.4 Results

Figure 5 show the sum of objective values for each solver method, allowing for an overview of how well the approximations compare to exact solutions. Figure 6 allows for comparison of runtimes from the various methods.

The entirety of test results are shown in Table 7, under Appendix B.4.

3.3.5 Discussion

In this subproblem, the QP model performs unevenly with Gurobi, in comparison with the other tested methods for finding optimal solutions, while the QP model performs significantly slower with Knitro than all other configurations, including all three set-ups finding proven optimality.

In contrast with experiments in both 3.1: Seller Discounts and 3.2: Supply Restrictions, Knitro does find solutions very close to optimality, even when compared to MIP configurations with 5% and 10% gap tolerances. Since QP algorithms have a tendency to become inefficient for a large number of variables [5], it is possible that the three-dimensional arrays constructed for this problem cause the longer runtimes. Meanwhile, Gurobi and Xpress are likely able to use conventional MIP algorithms, handling larger variable arrays with less issues. Further, they are likely to exploit the restricted amounts of sellers, which further cuts down the search space. The QP model used with Gurobi still shows comparable, but not superior, runtimes for optimal solutions.

For this subproblem, there is no compelling argument to be made in favor of QP, based on the experiments.
Figure 5: Sum of objective values for all tested datasets in the regions problem, where A-B-C consists of A products, B sellers in instance C. Gurobi (QP), Gurobi (MIP) and Xpress (MIP) show proven optimality. Lower values are better.

Figure 6: Runtimes in seconds for all tested datasets, where A-B-C has A products and B sellers for an instance C. Gurobi (QP), Gurobi (MIP), Xpress (MIP) found proven optimality and have filled markers. Lower values are better. The figure is provided in logarithmic scale as different methods provide results that differ in several orders of magnitude.
### 3.4 Reserved Price

A special case appears where the number of sellers a customer will order from is highly limited.

Consider a situation where a customer will purchase a number of products from one of two sellers. The first provides highly competitive prices for all products but one, which may be unavailable or sold at a very high cost. The other seller provides all products, but all are at a significantly higher price. In this case, a customer may want to consider ordering the competitively priced products from the first seller, while ordering the very last product by some other means.

To implement this problem, a *reserved price* is introduced. For each product such that, if the total cost will be lowered by purchasing a product of this price, it is applied without increasing the number of sellers from which an order is made.

In addition to the constants defined in Section 3.1: Seller Discounts, the following constants are defined.

- A seller limit $l$ that sets the highest total number of sellers.
- A one-dimensional array $H$ of size $P$, where each entry $H_i$ is the penalty, or reserved price, for not ordering a product from any of the defined sellers.

#### 3.4.1 MIP Model

We use the same decision variables as the 3.1.1: MIP Model for Section 3.1: Seller Discounts, with the following two additions.

- A one-dimensional array $V \in \mathbb{N}^{[S]}$ of binary values $V_j \in \{0, 1\}$ where $V_j = 1$ if seller $j$ has any order.
- A one-dimensional array $W \in \mathbb{R}^{[P]}$ of real values $0 \leq W_i \leq 1$ where $W_j \geq 0$ if a product is ordered from a different source.

This allows for a description of the problem as follows:

---

**Table 3:** Accuracy of the objective values of the regions problem. Each cell shows how much larger the objective value of a solution is compared to the proven optimality of each dataset. An optimum solution shows exactly 0%, while larger values suggest less accuracy.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Knitro</td>
<td>Max 1 cross it.</td>
<td>0.00%</td>
<td>0.01%</td>
<td>0.00%</td>
<td>0.71%</td>
<td>0.73%</td>
<td>0.66%</td>
<td>0.00%</td>
<td>0.00%</td>
<td>0.07%</td>
<td>0.01%</td>
<td>0.05%</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Max 2 cross it.</td>
<td>0.01%</td>
<td>0.02%</td>
<td>0.00%</td>
<td>1.18%</td>
<td>0.21%</td>
<td>0.84%</td>
<td>0.37%</td>
<td>0.00%</td>
<td>0.00%</td>
<td>0.07%</td>
<td>0.01%</td>
<td>0.05%</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Gurobi</td>
<td>QP</td>
<td>0.00%</td>
<td>0.00%</td>
<td>0.00%</td>
<td>0.00%</td>
<td>0.00%</td>
<td>0.00%</td>
<td>0.00%</td>
<td>0.00%</td>
<td>0.00%</td>
<td>0.00%</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>MIP, int.</td>
<td>3.71%</td>
<td>3.57%</td>
<td>2.71%</td>
<td>0.00%</td>
<td>2.07%</td>
<td>2.89%</td>
<td>2.57%</td>
<td>3.33%</td>
<td>1.31%</td>
<td>2.77%</td>
<td>1.84%</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>MIP, 5% gap</td>
<td>3.71%</td>
<td>3.57%</td>
<td>2.71%</td>
<td>0.00%</td>
<td>2.07%</td>
<td>2.89%</td>
<td>2.57%</td>
<td>3.33%</td>
<td>1.31%</td>
<td>2.77%</td>
<td>1.84%</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>MIP, no disc.</td>
<td>10.19%</td>
<td>11.50%</td>
<td>11.25%</td>
<td>21.70%</td>
<td>20.71%</td>
<td>20.21%</td>
<td>17.92%</td>
<td>18.43%</td>
<td>17.00%</td>
<td>17.48%</td>
<td>16.16%</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Xpress</td>
<td>None</td>
<td>0.00%</td>
<td>0.00%</td>
<td>0.00%</td>
<td>0.00%</td>
<td>0.00%</td>
<td>0.00%</td>
<td>0.00%</td>
<td>0.00%</td>
<td>0.00%</td>
<td>0.00%</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>5% gap</td>
<td>0.93%</td>
<td>0.90%</td>
<td>1.66%</td>
<td>0.00%</td>
<td>2.33%</td>
<td>2.13%</td>
<td>1.11%</td>
<td>1.31%</td>
<td>1.19%</td>
<td>1.11%</td>
<td>1.68%</td>
<td>1.22%</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>10% gap</td>
<td>0.93%</td>
<td>0.90%</td>
<td>1.66%</td>
<td>0.00%</td>
<td>2.33%</td>
<td>2.13%</td>
<td>1.11%</td>
<td>1.31%</td>
<td>1.19%</td>
<td>1.11%</td>
<td>1.68%</td>
<td>1.22%</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>No disc.</td>
<td>10.18%</td>
<td>11.50%</td>
<td>11.25%</td>
<td>21.70%</td>
<td>20.71%</td>
<td>20.21%</td>
<td>17.92%</td>
<td>17.43%</td>
<td>18.42%</td>
<td>17.00%</td>
<td>17.43%</td>
<td>16.20%</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Minimize \[ \sum_{i \in P} W_i \cdot H_i + \sum_{i \in P} \sum_{j \in S} C_{i,j} \cdot X_{i,j} - \sum_{i \in P} \sum_{j \in S} D_{i,j} \cdot Y_{i,j} \] (45)

Subject to \[ \forall i \in P \left( W_i + \sum_{j \in S} X_{i,j} \geq 1 \right) \] (46)

\[ \forall j \in S \left( \sum_{i \in P} C_{i,j} \cdot X_{i,j} - T_j \cdot Z_j \geq 0 \right) \] (47)

\[ \forall i \in P, j \in S \left( X_{i,j} - Y_{i,j} \geq 0 \right) \] (48)

\[ \forall i \in P, j \in S \left( Z_j - Y_{i,j} \geq 0 \right) \] (49)

\[ \forall i \in P, j \in S \left( C_{i,j} \cdot X_{i,j} \geq 0 \right) \] (50)

\[ \forall i \in P, \forall j \in S \left( V_j - X_{i,j} \geq 0 \right) \] (51)

\[ \sum_{j \in S} V_j - l \leq 0 \] (52)

Where the relations define the following:

(45) is the objective, i.e., total cost.

(46) ensures each product is ordered.

(47) is the discount condition.

(48) sets that discounts are only provided for purchased products.

(49) assures discounts are only provided for a product if the discount condition for the seller is met.

(50) sets that purchases can only be made when a seller provides the product.

(51) sets whether a seller has any order.

(52) limits total number of sellers.

3.4.2 QP Model

We use the same decision variables as the 3.1.2: QP Model for 3.1: Seller Discounts, with the following two additions.

- A one-dimensional array \( V \in \mathbb{N}^{\left| S \right|} \) of binary values \( V_j \in \{0, 1\} \) where \( V_j = 1 \) if seller \( j \) has any order.

- A one-dimensional array \( W \in \mathbb{R}^{\left| P \right|} \) of real values \( 0 \leq W_i \leq 1 \) where \( W_j \geq 0 \) if a product is ordered from a different source.

Minimize \[ \sum_{i \in P} H_i \cdot W_i + \sum_{i \in P} \sum_{j \in S} D'_{i,j} \cdot X_{i,j} + \sum_{i \in P} \sum_{j \in S} D_{i,j} \cdot X_{i,j} \cdot Y_j \] (53)

Subject to \[ \forall i \in P \left( W_i \sum_{j \in S} X_{i,j} \geq 1 \right) \] (54)

\[ \forall j \in S \left( \sum_{i \in P} D'_{i,j} \cdot X_{i,j} + \sum_{i \in P} D_{i,j} \cdot X_{i,j} - T_j \cdot (1 - Y_j) \geq 0 \right) \] (55)

\[ \forall i \in P, j \in S \left( D'_{i,j} \cdot X_{i,j} \geq 0 \right) \] (56)

\[ \forall i \in P, j \in S \left( V_j - X_{i,j} \geq 0 \right) \] (57)

\[ \sum_{j \in S} V_j - l \leq 0 \] (58)

(53) is the objective, i.e., total cost.

(54) ensures each product is ordered.

(55) is the discount condition.

(56) sets that purchases can only be made when a seller provides the product.

(57) sets whether a seller has any order.

(58) limits total number of sellers.
Figure 7: Sum of objective values for all tested datasets in the reserved price problem, where pA-sB-C consists of A products, B sellers in instance C. Gurobi (QP), Gurobi (MIP) and Xpress (MIP) show proven optimality. Lower values are better.

3.4.3 Implementation

Models for each solution were written in AMPL. The models are shown in Listing 8 and Listing 9 for the MIP solution and QP solution respectively.

3.4.4 Results

Figure 7 shows the sum of objective values for each solver method, allowing for an overview of how well the approximations compare to exact solutions. Table 4 shows the relative accuracy of each solution, allowing for a similar comparison at greater detail. Figure 8 allows for comparison of runtimes from the various methods. Full results are shown in Table 8.

3.4.5 Discussion

Similarly to observations in 3.3.5: Discussion, results for Gurobi and Xpress are favorable to those of Knitro. While QP with Knitro configurations consistently provides results near optimality, runtimes exceed those of Gurobi and Xpress by orders of magnitude in all tested datasets. Notably, allowing for 2 crossover iterations provides worse objective values – this may be explained by a floating point error. Further, approximations used with MIP configurations provide similarly promising results at even more improved runtimes. QP and MIP configurations in Gurobi provide generally similar runtimes, where the advantage varies by each dataset.

Overall, this problem suggests that QP and MIP configurations are similarly powerful using Gurobi, while Knitro provides no proven optimality at worse runtimes.
Comparison of MIP and QP in Sourcing Problems

Figure 8: Runtimes in seconds for all tested datasets, where pAsB-C has A products and B sellers for an instance C. Gurobi (QP), Gurobi (MIP), Xpress (MIP) found proven optimality and have filled markers. Lower values are better. The figure is provided in logarithmic scale as different methods provide results that differ in several orders of magnitude.

Table 4: Accuracy of the objective values of the reserved price problem. Each cell shows how much larger the objective value of a solution is compared to the proven optimality of each dataset. An optimum solution shows exactly 0%, while larger values suggest less accuracy.
4 Summary

4.1 Conclusion

Test results show that QP provides no significant benefit over MIP in solving the presented problems, with the used parameters.

Solvers exploiting the binary nature of variables, allowing for finding optimal solutions in non-convex problems, have lead to slightly increased efficiency in the experiments outlined in 3.1: Seller Discounts as well as 3.2: Supply Restrictions. In the following two experiments, results have been less consistent.

Approximate solutions provide several considerations. Gurobi and Xpress allow for relaxing integer variables in MIP models, while relaxed integer variables in QP models were only allowed in Knitro for the tested problems. Further, the non-convex nature of the discussed problems do not allow for globally optimal solutions in QP models in Knitro, such that the solutions become locally optimal—essentially becoming approximations. These solutions have generally been comparable to applying an optimality gap between 5% and 10% in Gurobi and Xpress. The significant caveat in this case, however, is that no such gap can be guaranteed for general, non-convex QP problems. Considering runtimes, QP models in Knitro appear to perform similarly to comparable MIP approximations at best, while being consistently and significantly slower at worst. The latter is certainly the case in experiments discussed in 3.3: Regional Division and 3.4: Reserved Price.

There are, however, many nuances to consider. For example, a model written in terms of QP may be more intuitively formulated and understood. If, for example, multiplication of binary variables is performed, a QP model appears to be likely to perform similarly to MIP model in finding a solution that is optimal, as shown in all four discussed problems, or within an increased global optimality gap. It may however be cumbersome to consider the limitations of what a solver allows for: at the time of writing, Xpress only accepts convex QP unless all variables are restricted to binary values [9].

To summarize, it appears that for the discussed problems and solvers, it is possible to write efficient MIP models that allow for finding solutions that are, at worst, nearly as good in terms of runtimes and optimality as QP.

4.2 Future Work

Since there are a large amount of untested solvers, testing them all would be far beyond the scope of this report.

Research in QP algorithms is being done continuously, which may quickly provide an advantage to solvers implementing the latest discoveries, while the same is true for MIP solvers. As such, the results found in this report may drastically differ in future versions of the same software. Generally, all tested solvers are considered state-of-the-art, suggesting that they incorporate what may be considered leading edge research. Since the software is proprietary, the details of their implementations are difficult to assess. Further, entirely different technologies such as constraint programming and local search allow for finding approximations and optimal solutions as well: since these are being continuously researched in a similar manner, considering the quality of these solvers could provide interesting results as well.

Further experimentation with models and parameters could also provide significant benefits. It is difficult to prove whether a specific implementation of a problem is most efficient, in any aspect, as unknown approaches may exist. The impact of setting different parameters may also provide interesting results: the tested solvers allow for a multitude of settings that can be used in a large number of combinations.

The problems discussed in this report do not cover the entirety of strategic sourcing. Any combination of the applied considerations may be used to create new sets of constraints. Further, entirely different constraints may be applied for penalizing factors such as environmental impact.

In conclusion, while this report shows that QP does not appear to provide any significant benefit for the discussed problems, it is by no means to be considered exhaustive.
References


A Prerequisite knowledge

A.1 NP-completeness

Problems with search spaces that grow exponentially (or worse) – in current solutions – can sometimes be classified as NP-complete.

NP stands for non-deterministic polynomial time: this is a class of problems that may be verified in polynomial time, i.e. \( O(n^c) \) for some constant \( c \). If a problem can be solved in polynomial time, it is considered to be a part of the problem class \( P \). Since problem solutions in \( P \) can be verified in polynomial time, \( P \subseteq NP \). Another class, NP-hard, consists of problems that are at least as hard as the hardest problems in NP. If a problem is both in \( NP \) and \( NP-hard \), it is considered \( NP-complete \). [3]

It is currently unknown whether \( P = NP \); that is, whether NP-problems can be solved in polynomial time [3]. Until then, large instances of NP-complete problems can take lifetimes to solve using brute-force algorithms, even with high-end hardware. Considering the problem discussed in 3.1: Seller Discounts, the smallest datasets consist of 100 products and 80 sellers, generating arrays with 8000 variables: assuming only binary assignments and that a generous instance would allow for 90% of the variables to be pre-set, the resulting search space of a naive algorithm would still consider \( 2^{800} \approx 6.67 \cdot 10^{240} \) combinations to exhaust all possible solutions. To put this in a real-life perspective, a case study of display sourcing for Procter and Gamble consisted of some 500 components and 40 sellers.

Using the methods discussed in the report, these problems can be modelled such that advanced programs can interpret the problem in ways that allow for it to be solved in near-polynomial time, in turn reducing runtimes by many orders of magnitude at large instances.

A.2 Linear Algebra

Consider the following:

- Constant matrix \( A \in \mathbb{R}^{i \times j} \) (size \( i \) by \( j \) with real numbers)
- Constant vector \( c \in \mathbb{R}^j \) (contains \( i \) real numbers)
- Variable vector \( x \in \mathbb{R}^j \) (contains \( i \) real numbers)
- Variable vector \( y \in \mathbb{R}^i \) (contains \( j \) real numbers)

Let them contain the following values:

\[
A = \begin{pmatrix}
a_{1,1} & \cdots & a_{i,1} \\
\vdots & \ddots & \vdots \\
a_{1,j} & \cdots & a_{i,j}
\end{pmatrix}, ~
\begin{pmatrix}
c_1 \\
\vdots \\
c_i
\end{pmatrix}, ~
\begin{pmatrix}
x_1 \\
\vdots \\
x_i
\end{pmatrix}, ~
\begin{pmatrix}
y_1 \\
\vdots \\
y_j
\end{pmatrix}
\]

This allows us to express inequalities of the form \( ax \leq y \) with the following:

\[
Ax \leq y \iff \begin{cases}
a_{1,1} \cdot x_1 + a_{2,1} \cdot x_2 + \ldots + a_{i,1} \cdot x_i \leq y_1 \\
\vdots \\
a_{1,j} \cdot x_1 + a_{2,j} \cdot x_2 + \ldots + a_{i,j} \cdot x_i \leq y_j
\end{cases}
\]

A transposed matrix or vector essentially mirrors it diagonally:

\[
x = \begin{pmatrix}
x_1 \\
\vdots \\
x_i
\end{pmatrix} \iff x^T = (x_1, x_2, \ldots, x_i)
\]

[3] For example, if the sellers do not provide a product, the variable setting an order can be set to 0.
[4] For comparison, albeit one that is difficult to reference: at the time of writing, the observable universe is commonly estimated to contain approximately \( 10^{80} \) protons.
To find the dot product of two vectors, one can simply transpose the first vector:

\[ \mathbf{c} \cdot \mathbf{x} = \mathbf{c}^T \mathbf{x} = (c_1, c_2, \ldots, c_i) \left( \begin{array}{c} x_1 \\ \vdots \\ x_i \\ x_1 \\ \vdots \end{array} \right) = c_1 \cdot x_1 + c_2 \cdot x_2 + \ldots + c_i \cdot x_i \]

In QP, as shown in 2.4: Quadratic Programming, we encounter multiplications of the form \( \frac{1}{2} \mathbf{x}^T \mathbf{A} \mathbf{x} \). In order for such a multiplication to hold, both sides of \( \mathbf{A} \) must be equal to the number of elements in \( \mathbf{x} \); for this case, let \( i = j = k \).

\[
\mathbf{x}^T \mathbf{A} \mathbf{x} = (x_1, x_2, \ldots, x_k) \left( \begin{array}{cccc} a_{1,1} & \cdots & a_{k,1} \\ \vdots & \ddots & \vdots \\ a_{1,k} & \cdots & a_{k,k} \end{array} \right) \left( \begin{array}{c} x_1 \\ \vdots \\ x_k \\ x_1 \\ \vdots \end{array} \right) = x_1(a_{1,1} \cdot x_1 + a_{2,1} \cdot x_2 + \ldots + a_{k,1} \cdot x_k) \\
+ x_2(a_{1,2} \cdot x_1 + a_{2,2} \cdot x_2 + \ldots + a_{k,2} \cdot x_k) \\
+ \ldots \\
+ x_k(a_{1,k} \cdot x_1 + a_{2,k} \cdot x_2 + \ldots + a_{k,k} \cdot x_k) \\
= a_{1,1} \cdot x_1^2 + a_{2,1} \cdot x_1 \cdot x_2 + \ldots + a_{k,1} \cdot x_1 \cdot x_k \\
+ a_{1,2} \cdot x_1 \cdot x_2 + a_{2,2} \cdot x_2^2 + \ldots + a_{k,2} \cdot x_2 \cdot x_k \\
+ \ldots \\
+ a_{1,k} \cdot x_1 \cdot x_k + a_{2,k} \cdot x_2 \cdot x_k + \ldots + a_{k,k} \cdot x_k^2 \\
= x_1^2 \cdot a_{1,1} + x_1 \cdot x_2(a_{1,2} + a_{1,1}) + \ldots + x_k \cdot x_k(a_{k,k} + a_{k,1}) \\
+ x_2^2 \cdot a_{2,2} + x_2 \cdot x_3(a_{2,3} + a_{2,2}) + \ldots + x_k \cdot x_k(a_{k,k} + a_{k,2}) \\
+ \ldots \\
+ x_k^2 \cdot a_{k,k}
\]

If \( \mathbf{A} \) is symmetric – mirrored across the upper-left to lower-right diagonal, such that \( a_{p,q} = a_{q,p} \) – then the result is as follows:

\[
\mathbf{x}^T \mathbf{A} \mathbf{x} = x_1^2 \cdot a_{1,1} + 2 \cdot x_1 \cdot x_2 \cdot a_{1,2} + \ldots + 2 \cdot x_1 \cdot x_k \cdot a_{1,k} \\
+ x_2^2 \cdot a_{2,2} + 2 \cdot x_2 \cdot x_3 \cdot a_{2,3} + \ldots + 2 \cdot x_2 \cdot x_k \cdot a_{2,k} \\
+ \ldots \\
+ x_k^2 \cdot a_{k,k}
\]

This relation also shows that a quadratic function can be expressed in the form \( \mathbf{x}^T \mathbf{A} \mathbf{x} \).

### A.3 Convex Functions and Sets

A convex set contains its entire line segment between any two points. For example, if a set is defined by the (filled) shape of a circle, the line between any two points on that circle is entirely within that same set, i.e., inside that circle. If a quarter were to be removed from that set – into a Pac-Man shape – any point on a line between the two corners of its rim would be outside of the set: therefore it would not be convex.

A convex function requires that, given values \( x_1 < x_2 < x_3 \) and a function \( f, f(x_2) \) is less than or equal to any point on the line between coordinates \((x_1, f(x_1))\) and \((x_3, f(x_3))\). This is the case if the second derivative of \( f \) is always non-negative. On a graph, this essentially becomes the shape of a cone, pointing downwards. If \( f(x_2) \) is strictly less than any point formed by the line, the function is strictly convex. For quadratic functions expressed in the form \( \mathbf{x}^T \mathbf{A} \mathbf{x} \), the function is strictly convex if and only if \( \mathbf{A} \) is positive definite. A consequence of a function being convex is that any local optimum must also be a global optimum.
B Experiments

B.1 Testing platform
Experiments where run on an Intel Core i5 6600K CPU 3.50 Ghz with 16 GB DDR4 RAM 3200 Mhz. The following software versions were used:

- **OS** x86 64-bit Funtoo Linux using kernel version 4.9.10
- **AMPL** Version 20170412 for Linux x86/64
- **Knitro** Version 10.2.0 for AMPL/Linux
- **Gurobi** Version 7.0.2 for AMPL/Linux
- **FICO Xpress** Version 8.0.4 (Xpress-Optimizer 29.01.10) for AMPL/Linux

B.2 Datasets
Datasets were generated randomly using parameters allowing for making reasonably competitive sellers. For example, a seller whose products have a higher price would provide a higher discount, while sellers whose base prices are low provide a less significant discount. Further, each product in a dataset was set to a standard price a seller would not diverge from by a significant degree, in order to avoid obvious solutions.

B.3 AMPL Implementations
The AMPL implementations are listed as follows:

- Listing 2 is the AMPL model for the MIP implementation of the discount problem.
- Listing 3 is the AMPL model for the QP implementation of the discount problem.
- Listing 4 is the AMPL model for the MIP implementation of the supply problem.
- Listing 5 is the AMPL model for the QP implementation of the supply problem.
- Listing 6 is the AMPL model for the MIP implementation of the regions problem.
- Listing 7 is the AMPL model for the QP implementation of the regions problem.
- Listing 8 is the AMPL model for the MIP implementation of the reserved price problem.
- Listing 9 is the AMPL model for the QP implementation of the reserved price problem.

Listing 2: An AMPL model for the MIP implementation of the seller discount problem

```AMPL
### PARAMETERS ###
param n_products; # Number of products
param n_sellers; # Number of sellers

set Products := 1..n_products; # Set of products
set Sellers := 1..n_sellers; # Set of sellers

param Prices {Products,Sellers} >= -1;
param Discounts {Products,Sellers} >= 0;
param DiscountThreshold {Sellers} >= 0;

### VARIABLES ###
# Order[i,j] > 0 iff product i ordered from seller j
var Order {Products,Sellers} >= 0, <= 1;
# OrderDiscount[i,j] > 0 iff product i ordered from seller j with discount
var OrderDiscount {Products,Sellers} >= 0, <= 1;
# SellerDiscount[j] == 1 iff seller j gives discount
var SellerDiscount {Sellers} binary;

### OBJECTIVE ###
```
# Sum of all orders subtracted by sum of all discounts
minimize totalCost:
\[
\left(\sum_{i \in \text{Products}, \ j \in \text{Sellers}} \text{Prices}[i,j] \cdot \text{Order}[i,j]\right) - \\
\left(\sum_{i \in \text{Products}, \ j \in \text{Sellers}} \text{Discounts}[i,j] \cdot \text{OrderDiscount}[i,j]\right);
\]

### CONSTRAINTS ###
subject to

# Each product must be ordered such that total ratio for all sellers is at least 1
\[
\left(\sum_{j \in \text{Sellers}} \text{Order}[i,j]\right) > 1;
\]

# Discount variable true iff order exceeds threshold
\[
\left(\sum_{j \in \text{Sellers}} \text{Order}[i,j]\right) \geq 1;
\]

# Product discount ratio cannot exceed that of product order ratio
\[
\text{discountPurchased}[i \in \text{Products}, \ j \in \text{Sellers}]:
\]

# Each product must be ordered such that total ratio for all sellers is at least 1
\[
\left(\sum_{j \in \text{Sellers}} \text{Order}[i,j]\right) > 1;
\]

### VARIABLES ###

# Order[i,j] > 0 iff product i ordered from seller j
\[
\text{Order}[\text{Products}, \ \text{Sellers}] > 0, \leq 1;
\]

# DiscountPrices[i,j] = 1 iff seller j gives discount
\[
\text{DiscountPrices}[\text{Products}, \ \text{Sellers}] = -1;
\]

# NoDiscount[j] = 1 iff seller j gives discount
\[
\text{NoDiscount}[\text{Sellers}] = 0;
\]

### OBJECTIVE ###

# Sum of all orders subtracted by sum of all discounts
minimize totalCost:
\[
\left(\sum_{i \in \text{Products}, \ j \in \text{Sellers}} \text{Order}[i,j] \cdot \text{DiscountPrices}[i,j]\right) + \\
\left(\sum_{i \in \text{Products}, \ j \in \text{Sellers}} \text{Discounts}[i,j] \cdot \text{NoDiscount}[j]\right);
\]

### CONSTRAINTS ###
subject to

# Each product must be ordered such that total ratio for all sellers is at least 1
\[
\left(\sum_{j \in \text{Sellers}} \text{Order}[i,j]\right) > 1;
\]

# Discount variable true iff order exceeds threshold
\[
\left(\sum_{j \in \text{Sellers}} \text{Order}[i,j]\right) \geq 1;
\]

# Product discount ratio cannot exceed that of product order ratio
\[
\text{discountPurchased}[i \in \text{Products}, \ j \in \text{Sellers}]:
\]

### AMPL code ###

```
# PARAMETERS
param n_products; # Number of products
param n_sellers; # Number of sellers

set Products := 1..n_products; # Set of products
set Sellers := 1..n_sellers; # Set of sellers

param DiscountPrices {Products,Sellers} >= -1;
param Discounts {Products,Sellers} >= 0;
param DiscountThreshold {Sellers} >= 0;

# VARIABLES

# Order[i,j] > 0 iff product i ordered from seller j
var Order {Products,Sellers} >= 0, <= 1;

# NoDiscount[j] = 1 iff seller j gives discount
var NoDiscount {Sellers} binary;

# OBJECTIVE

# Sum of all orders subtracted by sum of all discounts
minimize totalCost:
\[
\left(\sum_{i \in \text{Products}, \ j \in \text{Sellers}} \text{Order}[i,j] \cdot \text{DiscountPrices}[i,j]\right) + \\
\left(\sum_{i \in \text{Products}, \ j \in \text{Sellers}} \text{Discounts}[i,j] \cdot \text{NoDiscount}[j]\right);
\]

# CONSTRAINTS
subject to

# Each product must be ordered such that total ratio for all sellers is at least 1
\[
\left(\sum_{j \in \text{Sellers}} \text{Order}[i,j]\right) > 1;
\]

# Discount variable true iff order exceeds threshold
\[
\left(\sum_{j \in \text{Sellers}} \text{Order}[i,j]\right) \geq 1;
\]

# Product discount ratio cannot exceed that of product order ratio
\[
\text{discountPurchased}[i \in \text{Products}, \ j \in \text{Sellers}]:
\]

Listing 3: An AMPL model for the QP implementation of the seller discount problem.
### PARAMETERS ###

```ampl
param n_products; # Number of products
param n_sellers; # Number of sellers
set Products := 1..n_products; # Set of products
set Sellers := 1..n_sellers; # Set of sellers
param Prices {Products,Sellers} >= 0;
param Discounts {Products,Sellers} >= 0;
param DiscountThreshold {Sellers} >= 0;
param Supplies {Products,Sellers} >= 0;
param Allowance {Sellers} >= 0;
```

### VARIABLES ###

```ampl
# Order[i,j] > 0 iff product i ordered from seller j
var Order {Products,Sellers} >= 0, <= 1;
# OrderDiscount[i,j] > 0 iff product i ordered from seller j with discount
var OrderDiscount {Products,Sellers} >= 0, <= 1;
# SellerDiscount[j] == 1 ifff seller j gives discount
var SellerDiscount {Sellers} binary;
```

### OBJECTIVE ###

```ampl
minimize totalCost:

\[
\begin{align*}
&\text{sum} \{ i \text{ in Products}, j \text{ in Sellers} \} \{ \text{Prices}[i,j] \times \text{Order}[i,j] \} - \\
&\text{sum} \{ i \text{ in Products}, j \text{ in Sellers} \} \{ \text{Discounts}[i,j] \times \text{OrderDiscount}[i,j] \} \\
\end{align*}
\]

### CONSTRAINTS ###

```ampl
subject to

eachProductOrdered { i in Products }:

\[
\begin{align*}
&\text{sum} \{ j \text{ in Sellers} \} \{ \text{Order}[i,j] \} >= 1; \\
&\text{Discount variable true iff order exceeds threshold}
\end{align*}
\]

discountCondition { j in Sellers }:

\[
\begin{align*}
&\text{sum} \{ i \text{ in Products} \} \{ \text{Prices}[i,j] \times \text{Order}[i,j] \} - \\
&\text{DiscountThreshold}[j] \times \text{SellerDiscount}[j] >= 0; \\
&\text{Product discount ratio cannot exceed that of product order ratio}
\end{align*}
\]

discountPurchased { i in Products, j in Sellers }:

\[
\begin{align*}
&\text{Order}[i,j] - \text{OrderDiscount}[i,j] >= 0; \\
&\text{Product discount requires applied discount}
\end{align*}
\]

discountEnable { i in Products, j in Sellers }:

\[
\begin{align*}
&\text{SellerDiscount}[j] - \text{OrderDiscount}[i,j] >= 0; \\
&\text{Order at most the available amount}
\end{align*}
\]

supplyLimit { i in Products, j in Sellers }:

\[
\begin{align*}
&\text{Supplies}[i,j] - \text{Order}[i,j] >= 0; \\
&\text{Order at most the allowed amount from a seller}
\end{align*}
\]

sellerLimit { j in Sellers }:

\[
\begin{align*}
&\text{Allowance}[j] - \text{sum} \{ i \text{ in Products} \} \{ \text{Prices}[i,j] \times \text{Order}[i,j] \} >= 0; \\
&\text{Product discount requires applied discount}
\end{align*}
\]

### PARAMETERS ###

```ampl
param n_products; # Number of products
param n_sellers; # Number of sellers
set Products := 1..n_products; # Set of products
set Sellers := 1..n_sellers; # Set of sellers
param DiscountPrices {Products,Sellers} >= 0;
param Discounts {Products,Sellers} >= 0;
param DiscountThreshold {Sellers} >= 0;
param Supplies {Products,Sellers} >= 0;
param Allowance {Sellers} >= 0;
```
### VARIABLES ###

- \( \text{Order}[i,j] > 0 \text{ iff product } i \text{ ordered from seller } j \)
- \( \text{Var Order}[\text{Products}, \text{Sellers}] \geq 0, \leq 1 \)
- \( \text{NoDiscount}[j] = 1 \text{ iff seller } j \text{ gives discount} \)
- \( \text{Var NoDiscount} \{\text{Sellers}\} \text{ binary} \)

### OBJECTIVE ###

- \( \text{Sum of orders with discounted prices and discounts where it is not applied} \)
- \( \text{(discount is the difference between full price and discounted price)} \)
- \( \text{minimize totalCost} \)
- \( (\text{sum} \{ i \text{ in Products}, j \text{ in Sellers} \} \text{Order}[i,j] \ast \text{DiscountPrices}[i,j]) + \)
- \( (\text{sum} \{ i \text{ in Products}, j \text{ in Sellers} \} \text{Order}[i,j] \ast \text{Discounts}[i,j] \ast \text{NoDiscount}[j]) \)

### CONSTRAINTS ###

- \( \text{subject to} \)
- \( \text{Each product must be ordered such that total ratio for all sellers is at least 1} \)
- \( (\text{sum} \{ j \text{ in Sellers} \} \text{Order}[i,j]) \geq 1 \)
- \( \text{Discount variable true iff order exceeds threshold} \)
- \( \text{DiscountCondition} \{ j \text{ in Sellers} \} \)
- \( (\text{sum} \{ i \text{ in Products} \} \text{DiscountPrices}[i,j] \ast \text{Order}[i,j]) + \)
- \( (\text{sum} \{ i \text{ in Products} \} \text{Discounts}[i,j] \ast \text{Order}[i,j]) - \)
- \( \text{DiscountThreshold}[j] \ast (1 - \text{NoDiscount}[j]) \geq 0 \)
- \( \text{Order at most the available amount} \)
- \( \text{SupplyLimit} \{ i \text{ in Products}, j \text{ in Sellers} \} \)
- \( \text{Supplies}[i,j] - \text{Order}[i,j] \geq 0 \)
- \( \text{Order at most the allowed amount from a seller} \)
- \( \text{SellerLimit} \{ j \text{ in Sellers} \} \)
- \( \text{Allowance}[j] - \)
- \( (\text{sum} \{ i \text{ in Products} \} \text{DiscountPrices}[i,j] \ast \text{Order}[i,j]) - \)
- \( (\text{sum} \{ i \text{ in Products} \} \text{Discounts}[i,j] \ast \text{Order}[i,j]) \geq 0 \)

Listing 6: An AMPL model for the MIP implementation of the regions problem

### PARAMETERS ###

- \( \text{param n_products} \); \# Number of products
- \( \text{param n_sellers} \); \# Number of sellers
- \( \text{param n_regions} \); \# Number of regions
- \( \text{param globalSellerLimit} \); \# Maximum global number of sellers

### VARIABLES ###

- \( \text{var Order}[\text{Products, Sellers, Regions}] \geq 0 \)
- \( \text{var OrderDiscount}[i,j,k] = 0 \text{ iff product } i \text{ ordered from seller } j \text{ with discount} \)
- \( \text{var SellerDiscount}[\text{Sellers}] \text{ binary} \)
- \( \text{var RegionSellerOrder}[\text{Sellers, Regions}] \text{ binary} \)
- \( \text{var GlobalSellerOrder}[j] = 1 \text{ iff seller } j \text{ has order in region } k \)

### OBJECTIVE ###

- \( \text{Sum of all orders subtracted by sum of all discounts} \)
- \( \text{minimize totalCost} \)
Listing 7: An AMPL model for the QP implementation of the regions problem

```AMPL
# NUMBER OF UNITS ORDERED MUST SATISFY DEMAND IN CORRESPONDING REGION
subject to
   ProductOrdered {i in Products, k in Regions}:
      \sum {j in Sellers} Order[i,j,k] = ProductDemand[i,k];

# DISCOUNT VARIABLE TRUE IFF ORDER EXCEEDS THRESHOLD
subject to
discountCondition {j in Sellers}:
   \sum {i in Products, k in Regions} Prices[i,j,k] * Order[i,j,k] - DiscountThreshold[j] * SellerDiscount[j] >= 0;

# PRODUCT DISCOUNT RATIO CANNOT EXCEED THAT OF PRODUCT ORDER RATIO
subject to
discountPurchased {i in Products, j in Sellers, k in Regions}:
   Order[i,j,k] - OrderDiscount[i,j,k] >= 0;

# PRODUCT DISCOUNT REQUIRES APPLIED DISCOUNT
subject to
discountEnable {i in Products, j in Sellers, k in Regions}:
   ProductDemand[i,k] * SellerDiscount[j] - OrderDiscount[i,j,k] >= 0;

# ORDER ONLY EXISTING PRODUCTS; ASSUMES ALL PRICES ARE POSITIVE INTEGERS
subject to
   validProductOrder {i in Products, j in Sellers, k in Regions}:
      Prices[i,j,k] * Order[i,j,k] >= 0;

# SET SELLER REGIONAL ORDERS.
subject to
   setRegionSellerOrders {i in Products, j in Sellers, k in Regions}:
      RegionSellerOrder[j,k] + ProductDemand[i,k] - Order[i,j,k] >= 0;

# LIMITING NUMBER OF REGIONAL SELLERS
subject to
   limitRegionSellers {k in Regions}:
      RegionSellerOrder[j,k] - RegionSellerLimit[k] <= 0;

# SET GLOBAL SELLER ORDERS
subject to
   setGlobalSellerOrders {j in Sellers, k in Regions}:
      GlobalSellerOrder[j] - RegionSellerOrder[j,k] >= 0;

# LIMITING TOTAL NUMBER OF SELLERS
subject to
   limitGlobalSellers:
      \sum {j in Sellers} GlobalSellerOrder[j] - globalSellerLimit <= 0;
```

### PARAMETERS ###
- `param n_products; # Number of products`
- `param n_sellers; # Number of sellers`
- `param n_regions; # Number of regions`
- `param globalSellerLimit; # Maximum global number of sellers`

### VARIABLES ###
- `var Order {Products,Sellers,Regions} >= 0;`
- `var DiscountPrices {Products,Sellers,Regions} >= -1;`
- `var Discounts {Products,Sellers,Regions} >= 0;`
- `var DiscountThreshold {Sellers} >= 0;`
- `var ProductDemand {Products, Regions} >= 0;`
- `var RegionSellerOrder {Regions} >= 0;`
- `var DiscountInActive {Sellers} binary;`
- `var RegionSellerOrder {Regions} binary;`
- `var GlobalSellerOrder {Sellers} binary;`

### OBJECTIVE ###
# Sum of all orders subtracted by sum of all discounts
minimize totalCost:

\[
\sum \{ \text{i in Products, j in Sellers, k in Regions} \} \text{Order[i,j,k] * DiscountPrices[i,j,k]} + \\
\sum \{ \text{i in Products, j in Sellers, k in Regions} \} \text{Order[i,j,k] * Discounts[i,j,k] * DiscountInActive[j]};
\]

### CONSTRAINTS ###

subject to

# Each product must be ordered such that total ratio for all sellers is at least 1
eachProductOrdered \{ \text{i in Products, k in Regions} \}:

\[
\sum \{ \text{j in Sellers} \} \text{Order[i,j,k]} = \text{ProductDemand[i,k] >= 0};
\]

# Discount variable true iff order exceeds threshold
discountCondition \{ \text{j in Sellers} \}:

\[
\sum \{ \text{i in Products, k in Regions} \} \text{DiscountPrices[i,j,k] * Order[i,j,k]} + \\
\sum \{ \text{i in Products, k in Regions} \} \text{Discounts[i,j,k] * Order[i,j,k]} - \\
\text{DiscountThreshold[j]} * (1 - \text{DiscountInActive[j]}) >= 0;
\]

# Order only existing products, assume all prices are positive integers
validProductOrder \{ \text{i in Products, j in Sellers, k in Regions} \}:

\[
\text{DiscountPrices[i,j,k] * Order[i,j,k]} >= 0;
\]

# Regional limits
setRegionSellers \{ \text{i in Products, j in Sellers, k in Regions} \}:

\[
\text{RegionSellerOrder[j,k]} * \text{ProductDemand[i,k]} - \text{Order[i,j,k]} >= 0;
\]

limitRegionSellers \{ \text{k in Regions} \}:

\[
\sum \{ \text{j in Sellers} \} \text{RegionSellerOrder[j,k]} - \text{RegionSellerLimit[k]} <= 0;
\]

# Set global seller orders
setGlobalSellerOrders \{ \text{j in Sellers, k in Regions} \}:

\[
\text{GlobalSellerOrder[j]} - \text{RegionSellerOrder[j,k]} >= 0;
\]

# Limiting total number of sellers
limitGlobalSellers:

\[
\sum \{ \text{j in Sellers} \} \text{GlobalSellerOrder[j]} = \text{globalSellerLimit} <= 0;
\]

Listing 8: An AMPL model for the MIP implementation of the reserved price problem

### PARAMETERS ###

param n_products; # Number of products
param n_sellers; # Number of sellers
param sellerLimit; # Highest allowed number of sellers

set Products := 1..n_products; # Set of products
set Sellers := 1..n_sellers; # Set of sellers

param Prices {Products,Sellers} >= -1;
param Discounts {Products,Sellers} >= 0;
param DiscountThreshold {Sellers} >= 0;
param Penalty {Products} >= 0;

### VARIABLES ###

# Order[i,j] > 0 iff product i ordered from seller j
var Order {Products,Sellers} >= 0, <= 1;
# OrderDiscount[i,j] > 0 iff product i ordered from seller j with discount
var OrderDiscount {Products,Sellers} >= 0, <= 1;
# SellerDiscount[j] = 1 iff seller j gives discount
var SellerDiscount {Sellers} binary;
# SellerOrder[j] = 1 iff there is an order from seller j
var SellerOrder {Sellers} binary;
# ExternalOrder[i] > 0 if product order i is done externally
var ExternalOrder {Products} >= 0;

### OBJECTIVE ###

# Sum of all orders subtracted by sum of all discounts
minimize totalCost:

\[
\sum \{ \text{i in Products} \} \text{ExternalOrder[i] * Penalty[i]} + \\
\sum \{ \text{i in Products, j in Sellers} \} \text{Prices[i,j] * Order[i,j]} - \\
\sum \{ \text{i in Products, j in Sellers} \} \text{DiscountPrices[i,j,k] * Order[i,j,k] * DiscountInActive[j]};
\]
Listing 9: An AMPL model for the QP implementation of the reserved price problem

### PARAMETERS ###

```AMPL
param n_products; # Number of products
param n_sellers; # Number of sellers
param sellerLimit; # Highest allowed number of sellers

set Products := 1..n_products; # Set of products
set Sellers := 1..n_sellers; # Set of sellers
```

### VARIABLES ###

```AMPL
var Order {Products,Sellers} >= 0, <= 1; # Order[i,j] > 0 iff product i ordered from seller j
var NoDiscount {Sellers} binary; # NoDiscount[j] == 1 iff seller j gives discount
var SellerOrder {Sellers} binary; # SellerOrder[j] == 1 iff there is an order from seller j
var ExternalOrder {Products} >= 0; # ExternalOrder[i] > 0 if product order i is done externally
```

### OBJECTIVE ###

```AMPL
minimize totalCost:
    (sum {i in Products} ExternalOrder[i]*Penalty[i]) +
    (sum {i in Products, j in Sellers} Order[i,j]*DiscountPrices[i,j]) +
    (sum {i in Products, j in Sellers} Order[i,j]*Discounts[i,j]*NoDiscount[j]);
```

### CONSTRAINTS ###

```AMPL
subject to
eachProductOrdered {i in Products}:
    ExternalOrder[i] +
    (sum {j in Sellers} Order[i,j]) >= 1;
# Discount variable true iff order exceeds threshold
discountCondition {j in Sellers}:
    (sum {i in Products} Prices[i,j] * Order[i,j]) -
    DiscountThreshold[j] * SellerDiscount[j] >= 0;
# Product discount ratio cannot exceed that of product order ratio
discountPurchased {i in Products, j in Sellers}:
    Order[i,j] - OrderDiscount[i,j] >= 0;
# Product discount requires applied discount
discountEnable {i in Products, j in Sellers}:
    SellerDiscount[j] - OrderDiscount[i,j] >= 0;
# Order only existing products
validProductOrder {i in Products, j in Sellers}:
    Prices[i,j] * Order[i,j] >= 0;
# Mark sellers with orders
setSellerOrder {i in Products, j in Sellers}:
    SellerOrder[j] - Order[i,j] >= 0;
# Satisfy order limit
countSellers:
    (sum {j in Sellers} SellerOrder[j]) - sellerLimit <= 0;
```

### CONSTRAINTS ###

```AMPL
subject to
eachProductOrdered {i in Products}:
    ExternalOrder[i] +
    (sum {j in Sellers} Order[i,j]) >= 1;
# Discount variable true iff order exceeds threshold
discountCondition {j in Sellers}:
    (sum {i in Products} Prices[i,j] * Order[i,j]) -
    DiscountThreshold[j] * SellerDiscount[j] >= 0;
# Product discount ratio cannot exceed that of product order ratio
discountPurchased {i in Products, j in Sellers}:
    Order[i,j] - OrderDiscount[i,j] >= 0;
# Product discount requires applied discount
discountEnable {i in Products, j in Sellers}:
    SellerDiscount[j] - OrderDiscount[i,j] >= 0;
# Order only existing products
validProductOrder {i in Products, j in Sellers}:
    Prices[i,j] * Order[i,j] >= 0;
# Mark sellers with orders
setSellerOrder {i in Products, j in Sellers}:
    SellerOrder[j] - Order[i,j] >= 0;
# Satisfy order limit
countSellers:
    (sum {j in Sellers} SellerOrder[j]) - sellerLimit <= 0;
```
eachProductOrdered {i in Products}:  
  ExternalOrder[i] +  
  (sum {j in Sellers} Order[i,j]) >= 1;  
# Discount variable true iff order exceeds threshold  
discountCondition {j in Sellers}:  
  (sum {i in Products} DiscountPrices[i,j] * Order[i,j]) +  
  (sum {i in Products} Discounts[i,j] * Order[i,j]) -  
  DiscountThreshold[j] * (1 - NoDiscount[j]) >= 0;  
# Order only existing products, Assumes all Discountprices are positive integers  
validProductOrder {i in Products, j in Sellers}:  
  DiscountPrices[i,j] * Order[i,j] >= 0;  
# Mark sellers with orders  
setSellerOrder {i in Products, j in Sellers}:  
  SellerOrder[j] - Order[i,j] >= 0;  
# Satisfy order limit  
countSellers:  
  (sum {j in Sellers} SellerOrder[j]) <= sellerLimit;

**B.4 Tables**

The tables are listed as follows:

- **Table 5** contains the full results for the *discount* problem.
- **Table 6** contains the full results for the *supply* problem.
- **Table 7** contains the full results for the *regions* problem.
- **Table 8** contains the full results for the *reserved price* problem.
<table>
<thead>
<tr>
<th>Solver</th>
<th>Settings</th>
<th>Max 1 cross H</th>
<th>Max 2 cross H</th>
<th>No disc.</th>
<th>None</th>
<th>5% gap</th>
<th>10% gap</th>
</tr>
</thead>
<tbody>
<tr>
<td>Xpress</td>
<td>MIP, 5% gap</td>
<td>5% gap</td>
<td>5% gap</td>
<td>5% gap</td>
<td>5% gap</td>
<td>5% gap</td>
<td>5% gap</td>
</tr>
<tr>
<td>Xpress</td>
<td>MIP, rel.</td>
<td>5% gap</td>
<td>5% gap</td>
<td>5% gap</td>
<td>5% gap</td>
<td>5% gap</td>
<td>5% gap</td>
</tr>
</tbody>
</table>

**Table 5: Full results for the seller’s discount problem.**

<table>
<thead>
<tr>
<th>Solver</th>
<th>Settings</th>
<th>Max 1 cross H</th>
<th>Max 2 cross H</th>
<th>No disc.</th>
<th>None</th>
<th>5% gap</th>
<th>10% gap</th>
</tr>
</thead>
<tbody>
<tr>
<td>Xpress</td>
<td>MIP, 5% gap</td>
<td>5% gap</td>
<td>5% gap</td>
<td>5% gap</td>
<td>5% gap</td>
<td>5% gap</td>
<td>5% gap</td>
</tr>
<tr>
<td>Xpress</td>
<td>MIP, rel.</td>
<td>5% gap</td>
<td>5% gap</td>
<td>5% gap</td>
<td>5% gap</td>
<td>5% gap</td>
<td>5% gap</td>
</tr>
</tbody>
</table>

**Table 6: Full results for the supply problem.**

---

Experiments
Daniel Ahlbom, Uppsala University 2017

Quadratic Programming Models in Strategic Sourcing Optimization  Page 36 of 37
Table 7: Full results for the regions problem.

<table>
<thead>
<tr>
<th>Solver Settings</th>
<th>Data</th>
<th>Runtime</th>
<th>Obj.</th>
<th>MIP, int.</th>
<th>MIP, 5% gap</th>
<th>Obj.</th>
<th>MIP, no disc.</th>
<th>Obj.</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Knitro</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Max 1 cross it.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>None</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Xpress</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Max 10% gap</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>No disc.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 8: Full results for the reserved price problem.