TacTex: An Adaptive Supply Chain Bidding Agent

David Pardoe
Ronggang Yu
Ali Niaz
Peter Stone

Department of Computer Sciences
The University of Texas at Austin
Starting Point

• Formulate **optimal** solution for each of the 5 decisions

• **Intractable** in all cases

• Identify key necessary **approximations**

• Example: production and delivery
  - Deliver as soon as it’s produced
  - Heuristic production scheduling
Production and Delivery: Optimal LP

- Define $x[i][d]$: customer order $i$ satisfied on day $d$?
  - $d = -1 \implies$ use product inventory
- $Pr[i]$: profit (price) of order $i$
- $Pe[i][d]$: penalties avoided on $i$ for all days after $d$
- $int[d]$: interest earned on profit from day $d$ until end.
- Objective function:
  $$ \sum_{d=-1}^{\text{maxday}} \sum_{i} x[i][d] \times (Pr[i] + Pe[i][d]) \times int[d] $$
- Constraints: cycles, components, inventory, . . .
- Takes too long even with maxday = 0!
Production and Delivery: Approximations

• Consider late orders first (2 pools of orders)
  – Priority Sort: Rank by (price – cost + penalties)
  – Approximate LP:
    ✽ Separate by computer
    ✽ Separate use of inventory from production
    ✽ If taking too long, fall back to priority sort

• With free cycles, partially satisfy large future orders
Comparisons

- Experiments over 36 games
- 1 approximate LP, 4 Priority sort, 1 dummy
- Agents are otherwise identical

<table>
<thead>
<tr>
<th>AGENT NAME</th>
<th>RELATIVE SCORE</th>
<th>RANK</th>
<th>ERROR</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linear Prog.</td>
<td>3,440,916.76</td>
<td>1</td>
<td>1,171,318.3</td>
</tr>
<tr>
<td>Priority1</td>
<td>-295,478.49</td>
<td>2</td>
<td>1,239,094.1</td>
</tr>
<tr>
<td>Priority2</td>
<td>-350,361.10</td>
<td>3</td>
<td>939,827.4</td>
</tr>
<tr>
<td>Priority3</td>
<td>-1,320,184.18</td>
<td>4</td>
<td>1,411,318.9</td>
</tr>
<tr>
<td>Priority4</td>
<td>-1,474,892.99</td>
<td>5</td>
<td>1,106,764.8</td>
</tr>
</tbody>
</table>
Ordering Components

• Components are **cheapest on the first day**
  – Try to stock up
  – But don’t order too many

• The right number depends on the total number of **customer RFQs** in the game
RFQs vs. Orders

- RFQs on **day 2** gives indication of expected total

- Problem: cheapest supplies on **day 1**

- Solution: Send RFQs to **maintain flexibility**
  - RFQs of 8000, 4000, 2000, 1000, and 500 (in order)
  - Never need more than 16000
  - Can be combined to get a wide range of totals
  - Largest first to last until next order arrives
  - Accept a subset based on RFQ prediction

- **Prediction** based on simulation of day 2 RFQ ⇒ Total RFQs
Components after day 1

- Need a different strategy after day 1 (if more needed)
- Prices are determined by due date
  - Supplier has lots of orders before due date ⇒ high price
- Probe price as function of due date with small RFQs
- Request enough to maintain threshold supply 50 days ahead (assuming current rate of use)
- Only accept if expected profit increase > price
  - Marginal value based on assumptions about computer prices and other components’ costs
Offering Computers

- Find the set of offers that **maximizes profit**

- Need to **estimate** $P(\text{winning order } | \text{ offer price})$

- Given RFQ and cost of computer in question, optimal price maximizes $(\text{price} - \text{cost}) \times P(\text{order } | \text{ price})$

- May not be able to produce all orders for optimal prices
  - Then need to raise prices to **reduce demand**
Raising prices

- **Iteratively** raise prices on the least profitable offers.
- Based on *greedy* production scheduler:
  - Order by profit/Prod. cycles (main constraint)
  - Look ahead 15 days (last production for request today)
- Order not produced $\Rightarrow$ increase price (frac. of base price)
- **Repeat** until all orders can be produced.
- Too many orders $\Rightarrow$ less capacity for future orders.
- So we add **predicted** future RFQs to today’s RFQs.
Predicting RFQs

- Generate RFQs for computers that could be produced in the 15 days we are considering
- Only need to predict # of RFQs
- 2 options for using past data:
  - line fitting
  - averaging
- Tested using same model as the server
  - Thousands of simulated games
- **Average of the past four values** gave the best result
- Tried using slope of fitted line to predict the RFQ trend
  - Less accurate than simply assuming zero trend
Predicting the probability of an order

• For deciding offers, need $P(\text{order} \mid \text{price})$

• Useful data from **daily reports** (past 10 days):
  – the lowest price it was ordered at (P=1)
  – the mean low price
  – mean of mean low and mean high
  – the mean high price
  – the highest price (P=0)

• Estimate probabilities at middle 3 points
  – Start with .75, .50, .25
  – Also trying to use past game data

• Linearly interpolate between points for prices in between
Day Factor

- # days to due date also important
  - Learn **day factor**: multiplier for probability
  - Each possible day (3-12) has its own day factor
  - Day factors start at 1
  - Adjust based on expected # orders, actual orders

- Without it, get more orders on earlier days

- With it, more even distribution
Using Past Game Data

- Assume that games are played against constant opponents

- Mining logfiles gives long term $P(\text{offer} \mid \text{price})$

- Turned out that short term information from daily reports is more predictive
Summary

- Very little opportunity for optimal decision-making
- Lots of prediction in our strategy
- Many attempts at learning and adaptation
- So far only a few are useful (e.g. day factor)