# The bakery order scheduling problem

Roberto Meroni  Hugo Sanz González

December 2023

roberto.meroni@estudiantat.upc.edu
hugo.sanz@estudiantat.upc.edu

# Problem statement

A bakery receives $n$ orders for the following day, and it wants to maximize its profits by taking as many orders as it can afford.

Orders are numbered from $1$ to $n$ and the working hours are divided into time slots $1, 2, \ldots, t$.

# Problem statement

An order $i$ is satisfied if and only if it spends exactly $l_i$ slots in the oven and it is ready between slots $m_i$ and $M_i$, inclusive.

In that case the order brings a profit of $p_i$ and requires $a_i$ units of oven surface area while it is being prepared.

# Problem statement

An order $i$ is satisfied if and only if it spends exactly $l_i$ slots in the oven and it is ready between slots $m_i$ and $M_i$, inclusive.

In that case the order brings a profit of $p_i$ and requires $a_i$ units of oven surface area while it is being prepared.

The amount of surface area needed by all orders being baked at any particular time may not exceed the surface capacity $A$ of the bakery's oven.

# An integer linear program

**Decision variables:**

| | |
|---|---|
| $x_{ij}$ | Is the $i$th order is being baked at time $j$? |
| $y_i$ | Do we take the $i$th order? |
| $s_i, e_i$ | Starting and ending baking time of the $i$th order. |

Here $1 \leq i \leq n$ and $1 \leq j \leq t$; adopt the convention $s_i = e_i = 0$ whenever the $i$th order is not taken.

# An integer linear program

**Decision variables:**

| | |
|---|---|
| $x_{ij}$ | Is the $i$th order is being baked at time $j$? |
| $y_i$ | Do we take the $i$th order? |
| $s_i, e_i$ | Starting and ending baking time of the $i$th order. |

Here $1 \leq i \leq n$ and $1 \leq j \leq t$; adopt the convention $s_i = e_i = 0$ whenever the $i$th order is not taken.

**Objective:** maximize the gross profit $\sum_{i=1}^{n} p_i y_i$.

**Constraints:**

- If the $i$th order is taken, then it spends exactly $l_i$ slots in the oven:

$$\sum_{j=1}^{t} x_{ij} = l_i y_i, \quad \text{for } 1 \leq i \leq n, 1 \leq j \leq t.$$

**Constraints:**

- If the $i$th order is taken, then it spends exactly $l_i$ slots in the oven:

$$\sum_{j=1}^{t} x_{ij} = l_i y_i, \quad \text{for } 1 \leq i \leq n, 1 \leq j \leq t.$$

- The end time $e_i$ is $\geq$ to every slot $j$ in which the $i$th order is being baked:

$$e_i \geq j x_{ij} \quad \text{for } 1 \leq i \leq n, 1 \leq j \leq t.$$

- The start time $s_i$ is $\leq$ to every slot $j$ in which $x_{ij} = 1$:

$$t - s_i \geq (t - j) x_{ij} \quad \text{for } 1 \leq i \leq n, 1 \leq j \leq t.$$

**Constraints (cont.):**

- A taken order does not leave the oven until it is ready for delivery:

$$e_i - s_i = y_i(l_i - 1) \quad \text{for } 1 \leq i \leq n.$$

- A taken order is ready during its pick-up window:

$$m_i y_i \leq e_i \leq M_i y_i \quad \text{for } 1 \leq i \leq n.$$

- The capacity of the oven is never be exceeded:

$$\sum_{i=1}^{n} a_i x_{ij} \leq A \quad \text{for } 1 \leq j \leq t.$$

# A binary linear program

**Decision variables:**

$x_{ij}$ Does the $i$th order finish baking at time $j$?

Again $1 \leq i \leq n$ and $1 \leq j \leq t$. The variables $y_i$, $s_i$ and $e_i$ are no longer necessary!

# A binary linear program

**Decision variables:**

$$x_{ij} \quad \text{Does the } i\text{th order finish baking at time } j\text{?}$$

Again $1 \leq i \leq n$ and $1 \leq j \leq t$. The variables $y_i$, $s_i$ and $e_i$ are no longer necessary!

**Objective:** maximize the profits $\sum_{i=1}^{n} \sum_{j=1}^{t} p_i x_{ij}$.

The $i$th order is being baked at time $j$ if and only if one of $x_{ij}, x_{i,j+1}, \ldots, x_{i,j+\min\{j+l_i-1,t\}}$ equals 1.

**Constraints:**

- The capacity of the oven is never exceeded:

$$\sum_{i=1}^{n} \sum_{k=j}^{\min\{j+l_i-1,t\}} a_i x_{ik} \leq A, \quad \text{for } 1 \leq j \leq t.$$

**Constraints (cont.):**

- A taken order is finished during its pick-up window:

$$m_i x_{ij} \leq j x_{ij} \leq M_i x_{ij} \quad \text{for } 1 \leq i \leq n, 1 \leq j \leq t.$$

- An order is completed at most once:

$$\sum_{j=1}^{t} x_{ij} \leq 1 \quad \text{for } i \leq 1 \leq n.$$

- An order can only be delivered if $l_i$ slots have passed since the beginning of the day:

$$x_{ij} = 0 \quad \text{for } 1 \leq i \leq n, 1 \leq j < l_i.$$

# Model comparison

| | Variables | | |
|---|---|---|---|
| Model | Binary | Nonbinary | Constraints |
| ILP | $nt + n$ | $2n$ | $2nt + t + 4n$ |
| BLP | $nt$ | 0 | $2nt + t + \sum_{i=1}^{n} l_i$ |

# Model comparison

| | Variables | | |
|---|---|---|---|
| Model | Binary | Nonbinary | Constraints |
| ILP | $nt + n$ | $2n$ | $2nt + t + 4n$ |
| BLP | $nt$ | 0 | $2nt + t + \sum_{i=1}^{n} l_i$ |

# Greedy search

**Order selection:** Choose the order $i$ having the maximal *normalized profits* $p_i/a_i l_i^{\beta}$, where $0 \leq \beta \leq 1$ is a parameter.

This strategy is advantageous because it

- reduces fragmentation, and
- allows us to schedule smaller orders at a later stage, when less free oven space remains.

# Greedy search

**Assignment selection:** Choose a time slot $j$ that leads to a feasible solution and maximizes the minimum remaining oven surface area over time slots $j, \ldots, j + l_i - 1$.

This evens out the oven usage over the day, because it has the effect of scheduling an order during the least busy time slots.

$S \leftarrow \varnothing$, a set of (order, time slot) assignments.

Candidates $\leftarrow \mathrm{sort}(\text{Orders}, \text{NormalizedProfits}, \leq)$

**forall** $o \in$ Candidates **do**

  FeasibleAssignments $\leftarrow \varnothing$

  **for** $j = \text{MinReadyTimes}_o, \ldots, \text{MaxReadyTimes}_o$ **do**

    $q(o, j) \leftarrow \min_{j - \text{Length}_o + 1 \leq k \leq j} \text{RemainingCapacity}_k$

    **if** $S \cup \{(o, j)\}$ *is feasible* **then**

      FeasibleAssignments $\leftarrow$ FeasibleAssignments $\cup \{j\}$

  **if** FeasibleAssignments $\neq \varnothing$ **then**

    $j^\star \leftarrow \arg\max_{j \in \text{FeasibleAssignments}} q(o, j)$

    $S \leftarrow S \cup \{(o, j^\star)\}$

**return** $S$

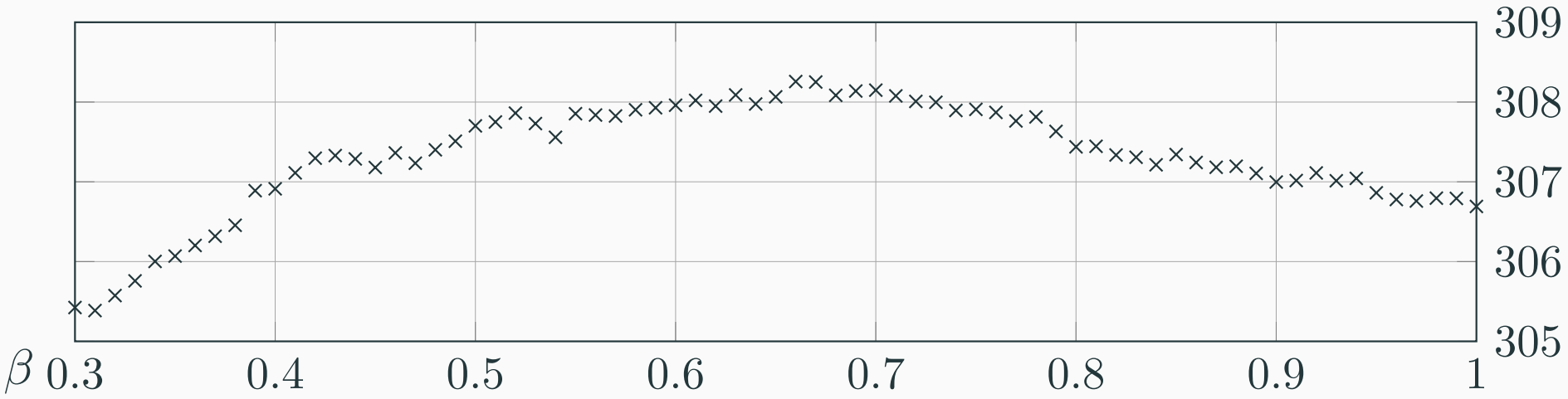


**Figure:** The average profit of the solutions found by the greedy algorithm with parameter $\beta$ on 1000 random instances of various sizes.

# Greedy search

| | Objective | | |
|---|---|---|---|
| n | Original ($\beta = 1$) | Tuned ($\beta = \frac{2}{3}$) | Improvement (%) |
| 50 | 182 | 189 | $4.00$ |
| 60 | 285 | 295 | $3.53$ |
| 70 | 381 | 394 | $3.51$ |
| 80 | 459 | 479 | $4.38$ |
| 90 | 401 | 398 | $-0.84$ |
| 100 | 638 | 641 | $0.48$ |

**Table:** The improvement in the objective after tuning the profit normalization parameter $\beta$.

# Local search

**Idea:** explore the neighborhood of a solution by shifting the assigned start and end time slots of a taken order, and trying to fit a new order in the freed oven space.

# Local search

**Idea:** explore the neighborhood of a solution by shifting the assigned start and end time slots of a taken order, and trying to fit a new order in the freed oven space.

Two policies:

a) Best improvement: do this for every assigned order $i$, ending time slot $m_i \leq j \leq M_i$, and overlapping unassigned order $k$. Keep the pair of reassignments that lead to the greatest increase in profits.

b) First improvement: take the first pair of reassignments found.

# GRASP

**Idea:** construct a solution by iteratively adding a random feasible assignment that leads to a *minimum remaining oven surface area* at or above the $\alpha$th percentile.

# GRASP

**Idea:** construct a solution by iteratively adding a random feasible assignment that leads to a *minimum remaining oven surface area* at or above the $\alpha$th percentile.

- Similar to the greedy approach, but introduces randomization.
- This allows us to explore the solution space more widely, and to escape from local maxima.
- Heuristic search is fast; we can execute multiple runs to obtain better results.

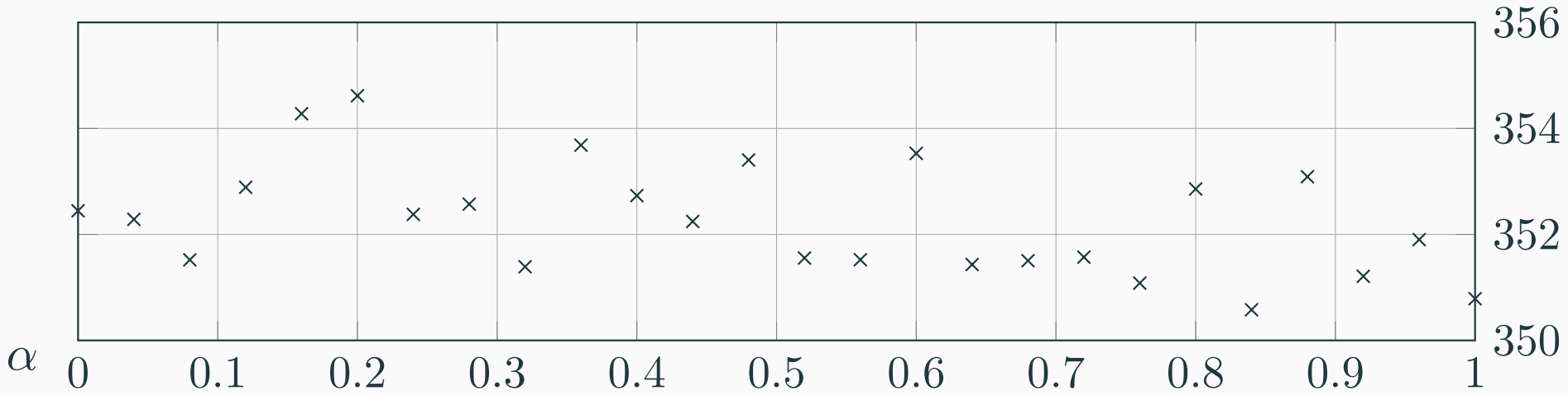


**Figure:** The average profit of the solutions found by the GRASP algorithm with parameter $\alpha$ on twenty random instances with $15 \leq n \leq 30$.

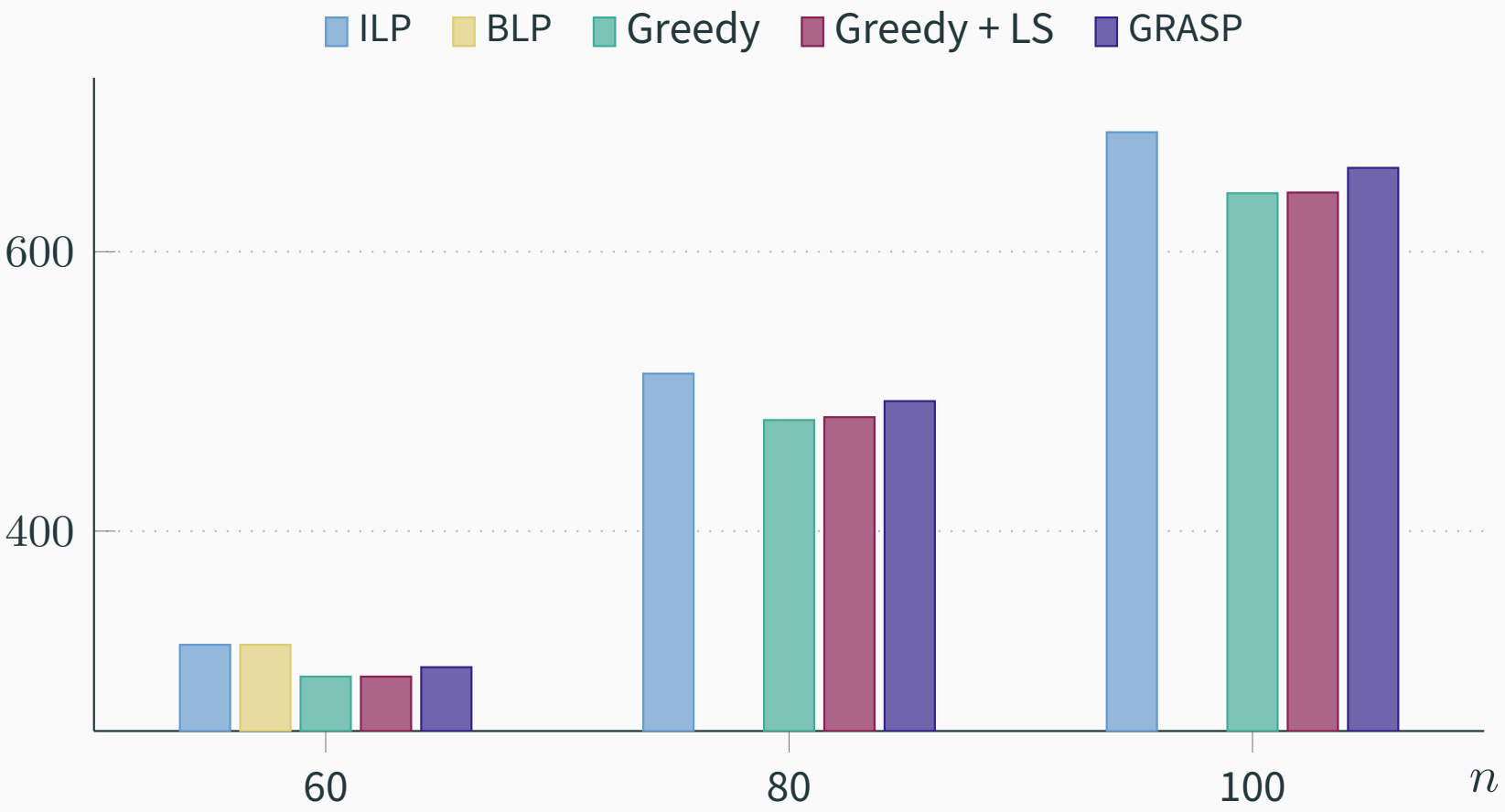


**Figure:** Profits of the solution found by different search methods when given a random instance with $n$ available orders. For $n > 60$, the CPLEX solver finds no solution to the binary linear program after ninety minutes when $\epsilon = 10^{-4}$.

| | CPLEX | | Heuristic methods | | | Optimality gap (%) | | |
|---|---|---|---|---|---|---|---|---|
| $n$ | ILP | BLP | Greedy | Greedy + LS | GRASP | Greedy | Greedy + LS | GRASP |
| 50 | 203 | 203 | 189 | 189 | 190 | 6.76 | 6.76 | 6.52 |
| 60 | 318 | 318 | 295 | 295 | 302 | 7.15 | 7.15 | 5.04 |
| 70 | 437 | - | 394 | 403 | 420 | 9.73 | 7.70 | 3.84 |
| 80 | 512 | - | 479 | 481 | 493 | 6.48 | 6.08 | 3.84 |
| 90 | 421 | - | 398 | 398 | 405 | 5.65 | 5.65 | 3.79 |
| 100 | 685 | - | 641 | 642 | 660 | 6.37 | 6.29 | 3.72 |

**Table:** Objective of the solutions found by different solvers on random instances with $n$ orders.

| | CPLEX | | Heuristic methods | | | Optimality gap (%) | | |
|---|---|---|---|---|---|---|---|---|
| $n$ | ILP | BLP | Greedy | Greedy + LS | GRASP | Greedy | Greedy + LS | GRASP |
| 50 | 203 | 203 | 189 | 189 | 190 | 6.76 | 6.76 | 6.52 |
| 60 | 318 | 318 | 295 | 295 | 302 | 7.15 | 7.15 | 5.04 |
| 70 | 437 | - | 394 | 403 | 420 | 9.73 | 7.70 | 3.84 |
| 80 | 512 | - | 479 | 481 | 493 | 6.48 | 6.08 | 3.84 |
| 90 | 421 | - | 398 | 398 | 405 | 5.65 | 5.65 | 3.79 |
| 100 | 685 | - | 641 | 642 | 660 | 6.37 | 6.29 | 3.72 |

**Table:** Objective of the solutions found by different solvers on random instances with $n$ orders.

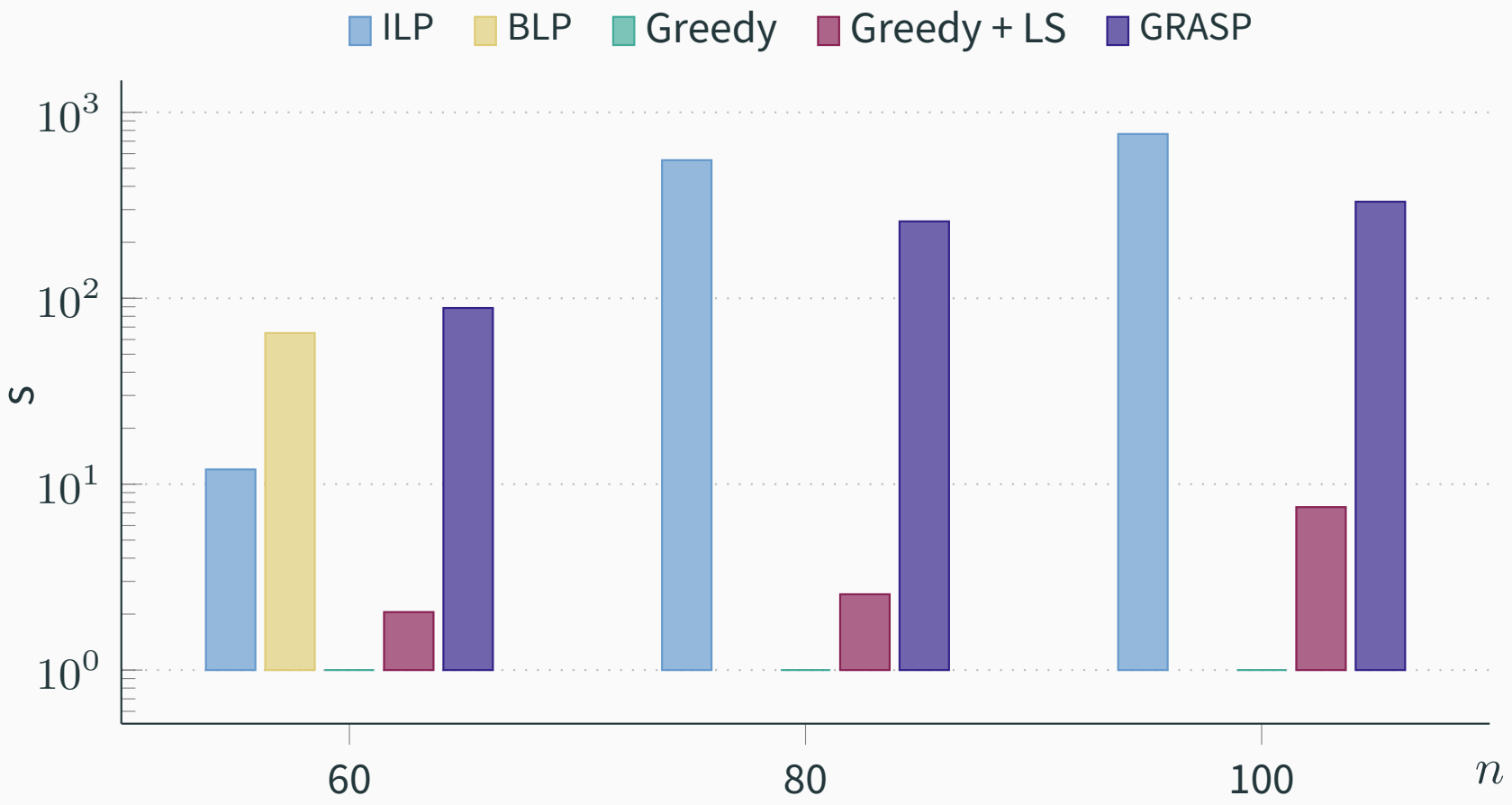


**Figure:** The running time of different search methods when given a random instance with $n$ available orders. For $n > 60$, the CPLEX solver finds no solution to the binary linear program after ninety minutes when $\epsilon = 10^{-4}$.

# Conclusions

- ILP models, greedy constructive algorithms, local search and GRASP help to solve the bakery order scheduling problem.
- CPLEX prefers the nonbinary model, and is best suited to instances with $n \leq 40$ orders.
- The greedy approach finds solutions of acceptable quality almost immediately. Subsequent local search only leads to small improvements in profit and is rather slow.
- GRASP, in turn, returns solutions with $2.5\%$ more profit when $n = 100$. Also, its efficiency is considerably better than that of CPLEX.
- Parameter tuning improves the quality of the solutions by up to $4\%$.