Selected applications of metaheuristics to location problems

Michel Gendreau

Département d'informatique et de recherche opérationnelle
and
Interuniversity Research Centre on Enterprise Networks, Logistics and Transportation (CIRRELT)
Université de Montréal

DEIS
University of Bologna
June 5, 2008
PRESENTATION OUTLINE

1. Location/allocation with balancing requirements

2. Locating a transit line

3. Location of inspection stations on a network

4. Locating and relocating ambulances

5. Variable Neighbourhood Search for the $p$-median

6. References
LOCATION/ALLOCATION WITH BALANCING REQUIREMENTS


- Problem occurring in the context of the management of a heterogeneous fleet of maritime containers over a medium to long term planning horizon.

- After delivery to import customers, loaded containers are unloaded and sent back, empty, to some depots waiting for allocation to export customers requesting containers.

- Empty containers can be returned to the port that initiated the loaded movement or repositioned to some other location in anticipation of future requests.

- Containers can typically be stored in or close to port locations or railroad yards at fairly low cost: there are therefore many attractive options for locating depots.

- Regional imbalances in the supply and demand of empty containers must be accounted for and can be corrected by shipping groups of empty containers between depots. Interdepot transportation is assumed to be more efficient than transportation between customers and depots.
MATHEMATICAL MODEL

• Problem data
  – $C = \{\text{customer zones}\}$
  – $J = \{\text{candidate depot locations}\}$
  – $P = \{\text{container types (products)}\}$
  – $f_j$: fixed cost of opening depot $j$
  – $c_{ijp}, c_{jip}$: unit transportation costs for product $p$ between customer zone $i$ and depot $j$
  – $s_{jkp}, s_{kjp}$: unit transportation costs for product $p$ between depots $j$ and $k$
  – $O_{ip}$: supply of product $p$ in customer zone $i$
  – $D_{ip}$: demand of product $p$ in customer zone $i$
MATHEMATICAL MODEL

• Variables
  - $x_{ijp}, x_{jip}$: flows of product $p$ between customer zone $i$ and depot $j$ (in both directions)
  - $w_{jkp}, w_{kjp}$: flows of product $p$ between depots $j$ and $k$ (in both directions)
  - $y_j$: 0–1 variable indicating if depot $j$ is open or not

• Constraints
  - Satisfaction of supply for each customer zone and product
  - Satisfaction of demand for each customer zone and product
  - Linking constraints between $x_{ijp}, x_{jip}$ and $y_j$ for each depot location $j$
  - Flow balance equations for each depot and each product

• Objective: minimize the sum of all fixed and transportation costs
PROPERTIES

• Fairly complex mixed integer programming formulation.

• For any given vector \( \hat{y} \), the problem reduces to a to an uncapacitated multicommodity minimum cost network flow problem \( P(\hat{y}) \), whose objective value can be “easily” computed.

• To prevent infeasible solutions, an artificial depot with very high transportation cost arcs from/to all customer is added. A solution is thus feasible is flows to/from the artificial depot are equal to 0.
TABU SEARCH SOLUTION APPROACH

- Search space: space of location variables.

- Neighbourhoods for regular search: Add/drop, Swap (restricted: for each open depot, only the closest closed depot is considered).
  - Add/drop sequences are followed by swap sequences.
  - Sequences end after some number of iterations without improvement in the value of the “local best” solution.
  - Only a subset of moves is selected randomly in each iteration; the selection probabilities are adjusted dynamically during the search to encourage the search to produce feasible solutions.
  - Short-term tabu lists of fixed length for both add/drop and swap moves; both the reversal and the repetition of recent swap moves are prohibited.
  - Information is transferred from the swap tabu list to the add/drop one, but not vice-versa.
TABU SEARCH SOLUTION APPROACH (2)

• Intensification:
  – “Strict” swap sequences from the best known solution.
  – Moves are only performed if they improve the current solution.

• Neighbour evaluation uses estimates of some costs to avoid solving the costly minimum cost flow problems. Only the best neighbour, according to the estimates, is evaluated exactly using a network optimization software.

• The costs estimation formulas are complicated and differ for each type of moves. Furthermore, they rely on some unit cost estimates for balancing flows that are derived from previous solution by exponential smoothing.

• Search diversification:
  – Performed after $N$ “inner loops” of regular search and intensification phases.
  – Forces in the solution the $\gamma$ depots the less used up to then; their removal is also made tabu.
  – A “diversification memory” is used to prevent always looking at the same depots.
COMPUTATIONAL RESULTS

• 12 medium instances with about 25 locations and 125 customers and 8 larger ones with 44 depots and 220 customers.

• Instances with low and high fixed-to-variable cost ratios.

• Extensive experimentation required for parameter setting.

• Computational experiments showed the need to add a descent phase without sampling from the best global solution to ensure that a local optimum was returned (this is because sampling is used). This phase takes about 1/8 of the total CPU time and improves the quality of solutions obtained in 6 instances out of 20.

• When compared to an effective dual ascent procedure, the tabu search heuristic finds better solutions for 14 instances out of 20 and equivalent ones in two cases.

• Tabu search is also more robust than the dual ascent procedure, but is also slower (on average 67 vs 17 s).
PARALLEL IMPLEMENTATIONS


- This problem was used to test and compare various strategies for the parallelization of tabu search heuristics.

- The basic model and tabu search approach are left unchanged, but some of the parallelization strategies involve *multi-thread* search.
LOCATING A TRANSIT LINE I


- Objective is to locate on a grid of possible station locations a fixed number of stations making up a transit line to maximize a weighted measure of the population living close to the stations (the population at a given point is multiplied by a weight that depends upon the Manhattan distance between this point and the station).

- Lower and upper bounds on inter-station distances (8 and 16 in our application) are given and ensure that the catchment areas of stations are disjoint.

- Important: origin-destination flows are not considered, only populations “covered” by the stations.
TABU SEARCH SOLUTION APPROACH

• Solution: ordered sequence of station coordinates.

• Search space: space of feasible lines.

• Initial solutions:
  – Generated by performing a random walk along the main or the second diagonals of the grid;
  – Stations are placed after 12 or 13 steps.

• Neighbourhood: corresponds to feasible moves of one of the stations by one unit along coordinate axes.

• Tabu search with random tabu tags to prevent reversal of moves.

• Diversification/intensification:
  – Applied after a number of iterations without improvement from the best known solution.
  – “Shake up”, i.e., moving several stations several units away from their current location.
  – Moving station in opposite direction is made tabu.
TABU SEARCH SOLUTION APPROACH (2)

- False starts:
  - 60 initial configurations generated
  - 100 iterations of TS applied to each
  - retain 30 best and apply full procedure to them to yield 30 solutions.

- Extensive computational experiments on 10 randomly generated instances to assess the impact of parameters of the procedure.

- The calibration of parameters cannot be performed in isolation: strong parameter interactions.

- Results on instances with known optima showed that optimal or near-optimal solutions were obtained for most instances.
LOCATING A RAPID TRANSIT LINE II

• Bruno, Gendreau, and Laporte (2002).

• Similar assumptions to the previous case with respect to population and coverage measures.

• Constraints on the line are defined with respect to the Euclidean norm, rather than the Manhattan one:
  – Maximum separation between consecutive stations
  – Minimum separation between any two stations
  – This ensures that the catchment areas of stations are disjoint.

• The objective is still to locate a fixed number of stations making up a transit line to maximize the same weighted measure of the population living close to the stations.
SOLUTION APPROACH

- Two-phase approach:
  - Greedy construction phase
  - Improvement phase

- In the construction phase, stations are added in a greedy fashion to the alignment on the basis of their population coverage, as long as it possible to link them to the closest station using the remaining number of stations.

- In the improvement phase, one extracts a partial alignment from the best known solution and tries to extend it, using the construction procedure, into several full alignments.

- Computational experiments showed that the heuristic could find optimal or near-optimal solutions in just a few seconds.

- The improvement procedure was also tested with Dufourd’s method for deriving an initial solution and proved to be robust.

- Additional tests were performed on real data from Milan (130 x 130 grid with squares of 150-meter side). The solution was found in just 10 s.
LOCATION OF INSPECTION STATIONS ON A NETWORK


• Location of a fixed number of inspection stations at vertices of a network to reduce risk (preventive policy).

• The problem is stated in terms of flows along paths $p$ in a set $P$. For each path $p$ and each vertex $v_i$ along $p$, we are provided with the reduction of risk obtained if the first facility encountered along that path is located at $v_i$.

• The problem can be formulated as integer program with $y_i$ location variables and $x_{ip}$ “interception” variables (i.e., $x_{ip} = 1$, if the flow of path $p$ is intercepted at vertex $v_i$), with constraints:
  
  – on the number of available facilities,

  – linking interception and location variables for each vertex $v_i$

  – allowing interception in at most one vertex along each path.
SOLUTION APPROACH

• Search space: space of subsets of \( m \) vertices.

• Initial solution obtained by an existing Greedy Search heuristic.

• Neighbourhood: corresponds to moving a single station (five rules for choosing facilities candidate to move are proposed) to its “best” location.

• Simple local search and tabu search with fixed-length tabu list preventing repetition or reversal of moves.

• The two search heuristic were tested with the 5 rules on a set of randomly generated instances with up to 200 instances.

• Computational results show that the greedy heuristic already provides very good solutions, but these can be improved by the other procedures (especially by TS).
LOCATING AND RELOCATING AMBULANCES

• Gendreau, Laporte, and Semet (1998)

• The overall objective is to be able to manage a fleet of ambulances to cover emergency calls over some time horizon.

• Covering calls implies meeting some service targets with respect to response times to calls (e.g., answering urgent calls within 7 minutes).

• As a first step, a static problem is considered: given a set of given potential sites $W$ (with known coverage attributes w.r.t. population zones), $p$ ambulances are to be located among these sites to maximize the population covered by two ambulances, subject to other coverage constraints.

• This problem is modeled as integer program with $y_j$ (integer, not binary!) location variables and $x_i^1$ and $x_i^2$ “coverage” variables for each population zone.
TABU SEARCH SOLUTION APPROACH

- Search space: space of possible assignments of the $p$ ambulances to the locations in $W$.

- Solutions not satisfying the covering constraints are allowed, but penalized.

- The initial solution is generated by progressive rounding of the continuous relaxation of the IP formulation.

- Complex neighbourhood structure defined by a multi-step procedure that may yield moves in which several ambulances are moved: start by moving an ambulance to one the 5 closest neighbouring sites, then move other ambulances to restore coverage constraints.

- Random tabu tags prevent the reversal of moves.

- Diversification based on moving ambulances to sites that are not near.

- Stopping criterion: the procedure is stopped after a number of iterations without improvement or when the a feasible solution with value no less than 99% of the value of the continuous relaxation is found.

- Computational results showed the method to be very effective on randomly generated and actual data.
DYNAMIC VARIANT AND PARALLEL IMPLEMENTATION


• The allocation of ambulances to incoming calls is governed by fairly straightforward rules (no optimization).

• However, after each call, it might be necessary to improve the deployment of the fleet by relocating some ambulances.

• This amounts to solving a modified version of the static problem previously described.

• To save reaction time the procedure uses the time between calls to precompute relocation decisions associated with the possible assignment of each available ambulance to the next incoming call.

• Because of this anticipation, the computational burden is substantial and parallel processing necessary.

• The method was tested in simulations based on actual data from Montreal.
Parallel Variable Neighbourhood Search for the $p$-Median

Teodor Gabriel Crainic
Michel Gendreau
Pierre Hansen
Nenad Mladenović

Centre for Research on Transportation
GERAD
Université de Montréal
École des Hautes Études Commerciales
Université du Québec à Montréal
Matematički Institut, SANU

2002
Outline

♣ The $p$-Median problem: Central in Location

♣ Many works
   ◇ Exact
   ◇ Heuristics and Meta-heuristics

♣ Variable Neighbourhood Search
   ◇ One of the most successful meta-heuristics for the $p$-median problem
   ◇ A number of variants

♣ Parallel Meta-heuristics

♣ Parallel Strategies for VNS

♣ Experimental results

♣ Perspectives
The \( p\)–Median

\[ L = \{ \text{Facilities (location points), } j = 1, \ldots , m \} \]

\[ U = \{ \text{Users (customers, demand points), } i = 1, \ldots , n \} \]

\[ D = (d_{ij}) : \text{Travel (cost) matrix for all } j \in L \text{ and } i \in U \]

Objective: Minimize total distance:

\[ \min \sum_{i \in U} \min_{j \in L} d_{ij} \]
The combinatorial optimization formulation

- \( y_j = 1 \): Facility is located at \( j \) (0, otherwise)
- \( x_{ij} = 1 \): User \( i \) is assigned to facility \( j \) (0, otherwise)

\[
\min \sum_i \sum_j d_{ij} x_{ij}
\]

s.t. \( \sum_j x_{ij} = 1, \forall i \)

\( x_{ij} \leq y_j, \forall i, j \)

\( \sum_j y_j = p \)

\( x_{ij}, y_j \in \{0, 1\} \)
Variable Neighbourhood Search

♣ Change neighbourhoods in search of better solutions

♣ Descend to a local minimum

♣ Explore increasingly distant neighbourhoods of this solution (systematic or randomly)

♣ Jump from current solution to a new one if and only if a better solution has been found

♣ Otherwise, change neighbourhood
Simple VNS

- $\mathcal{N}_k$: Neighbourhoods ($k = 1, \ldots, k_{\text{max}}$)

- $\mathcal{N}_k(x)$: $k^{th}$ neighbourhood of $x$

Initialization: Select $\mathcal{N}_k$; Initial solution $x$; Stopping condition

Repeat until stopping condition is met:
(1) $k \leftarrow 1$;
(2) Repeat until $k = k_{\text{max}}$:
   (a) Shaking: Generate at random $x' \in \mathcal{N}_k(x)$
   (b) Local search: From $x$ to new local optimum $x''$
   (c) Move if $\text{Cost}(x'') < \text{Cost}(x)$;
       Continue the search with $\mathcal{N}_1$;
       $k \leftarrow k + 1$, otherwise
Reduced VNS

♣ Simple VNS without the local search

♣ Other variants exist
Parallel Computation

♣ Faster - for similar solution quality

♣ Better - for similar time (effort?)

♣ Larger problem instances

♣ Robustness ("industrial strength")
Parallel Meta-heuristics

♣ **Type 1 (low level):** Parallelization of operations within an iteration of the solution method

♦ Gain in time

♦ Same search trajectory

♣ **Type 2:** Decomposition of problem domain or search space

♣ **Type 3:** Multi-search with various degrees of synchronization and co-operation

♦ Independent search

♦ Co-operative search
  Better if well done ...
Previous Work

♣ García-López et al. 2001

♣ Type 1 (SPVNS): Parallelize Local Search

♣ Type 3 (RSVNS): Independent searches

♣ Type 3 (RPVNS): Rigid synchronization: Execute Shake and Local Search in parallel

♣ Only on 1440 customer problems

♣ Nice speedups for SPVNS

♣ Somewhat better solutions for RSVNS (and RPVNS)

♣ Author suggestion: RSVNS + SPVNS
Our Approach

♠ We aim to stay “close” to the VNS spirit when designing parallel strategies

♠ Distribute neighbourhoods

♠ Implemented in a master-slave scheme

♠ SNVNS (Simple Neighbourhood distribution)
  ◊ Each process receives a neighbourhood $N_k(x)$
  ◊ Shakes and (local) Searches
  ◊ Best solution is compared to incumbent

♠ It may be viewed as a Type 2 approach
♣ **CNVNS** (Co-ordinated Neighbourhood distribution)

◊ Initial solution: Parallel Reduced VNS

◊ Each process receives an initial solution (local incumbent) and a neighbourhood

◊ Shakes and (local) Searches

◊ If it improves its local incumbent
  ♥ Continues the search with \( \mathcal{N}_1 \)
  ♥ Until no further improvement is possible
  ♥ When it sends his best solution to master

◊ Otherwise, receives new initial solution and continues from current neighbourhood

♣ It may be viewed as an asynchronous co-operative approach
Implementation

♣ In Fortran 77

♣ MPI

♣ SUN E10000, 64 Gb of RAM memory, 64 400 Mhz processors, Solaris 2.7

♣ On large problem instances from the TSP-Lib

♣ Local search

◊ To gain time, *first improvement* (rather than *best improvement*)

◊ Fast interchange move
Experimental Framework

♣ Problem dimensions

♦ n=1400, p=10,100, $t_{max} = 1400$ sec CPU
♦ n=3038, p=50,500, $t_{max} = 3000$ sec CPU
♦ n=5934, p=50,500, $t_{max} = 5000$ sec CPU
♦ n=11948, p=100,1000, $t_{max} = 10000$ sec CPU

♣ 10 repetitions for each instance

♣ Number of processors $np = 1, 5, 10, 15$

♣ Two sets of experiments

♦ All problems, all $np$, for $t = t_{max}/np$
♦ n=3038 instances, all $np$ for 3000 sec CPU
<table>
<thead>
<tr>
<th>$n$</th>
<th>$p$</th>
<th>% Dev. of the best</th>
<th>% Dev. of the average</th>
<th>Standard deviation</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>1 5 10 15</td>
<td>1 5 10 15</td>
<td>1 5 10 15</td>
</tr>
<tr>
<td>1400</td>
<td>10</td>
<td>0.00 0.00 0.00 0.00</td>
<td>0.00 0.00 0.00 0.00</td>
<td>0.2 0.0 0.0 0.0</td>
</tr>
<tr>
<td>20</td>
<td></td>
<td>0.00 0.00 0.00 0.00</td>
<td>0.00 0.00 0.00 0.02</td>
<td>0.2 0.0 0.0 19.6</td>
</tr>
<tr>
<td>30</td>
<td></td>
<td>0.01 0.00 0.01 0.02</td>
<td>0.01 0.00 0.01 0.02</td>
<td>14.1 3.2 16.7 16.6</td>
</tr>
<tr>
<td>40</td>
<td></td>
<td>0.00 0.01 0.01 0.01</td>
<td>0.00 0.01 0.01 0.01</td>
<td>2.1 5.3 5.4 5.5</td>
</tr>
<tr>
<td>50</td>
<td></td>
<td>0.05 0.00 0.10 0.05</td>
<td>0.05 0.00 0.10 0.05</td>
<td>40.2 0.0 53.1 40.0</td>
</tr>
<tr>
<td>60</td>
<td></td>
<td>0.03 0.04 0.03 0.03</td>
<td>0.03 0.04 0.03 0.03</td>
<td>4.3 6.7 6.5 7.5</td>
</tr>
<tr>
<td>70</td>
<td></td>
<td>0.01 0.00 0.01 0.05</td>
<td>0.01 0.00 0.01 0.05</td>
<td>4.4 0.0 8.2 23.5</td>
</tr>
<tr>
<td>80</td>
<td></td>
<td>0.03 0.07 0.07 0.06</td>
<td>0.03 0.07 0.07 0.06</td>
<td>6.2 14.7 13.5 16.0</td>
</tr>
<tr>
<td>90</td>
<td></td>
<td>0.02 0.02 0.06 0.19</td>
<td>0.02 0.02 0.06 0.19</td>
<td>10.7 6.8 12.0 28.6</td>
</tr>
<tr>
<td>100</td>
<td></td>
<td>0.07 0.10 0.09 0.07</td>
<td>0.07 0.10 0.09 0.07</td>
<td>5.9 9.2 8.2 6.3</td>
</tr>
</tbody>
</table>

|      | 3038 | 0.00 0.02 0.04 0.03 | 0.03 0.07 0.12 0.21  | 60.1 294.2 349.3 706.4  |
|      | 100  | 0.12 0.21 0.38 0.54 | 123.4 239.7 582.8 498.8  |
|      | 150  | 0.07 0.21 0.45 0.74 | 56.4 250.9 389.2 386.1  |
|      | 200  | 0.04 0.27 0.47 0.61 | 83.1 237.2 256.4 287.0  |
|      | 250  | 0.07 0.26 0.44 0.47 | 113.8 209.1 186.3 255.4  |
|      | 300  | 0.07 0.25 0.45 0.59 | 116.7 137.8 149.0 399.2  |
|      | 350  | 0.12 0.33 0.63 0.77 | 146.3 271.1 467.2 605.8  |
|      | 400  | 0.18 0.44 0.74 0.98 | 113.7 231.2 278.8 386.7  |
|      | 450  | 0.16 0.48 0.70 0.90 | 3435.8 7812.0 6500.0 9337.2  |
|      | 500  | 0.33 0.43 0.45 0.50 | 2792.1 3674.6 372.8 580.7  |
|      | 5934 | 0.12 0.33 0.63 0.71 | 2252.3 3813.3 5903.2 6756.7  |
|      | 100  | 0.11 0.35 0.56 0.58 | 1575.8 2217.0 3968.5 6647.7  |
|      | 150  | 0.08 0.32 0.48 0.64 | 757.2 1416.5 2712.0 4957.5  |
|      | 200  | 0.17 0.55 0.73 0.91 | 1072.3 1344.9 3052.4 4094.7  |
|      | 250  | 0.10 0.40 0.47 0.63 | 934.8 1540.7 1508.1 3814.1  |
|      | 300  | 0.11 0.30 0.39 0.51 | 890.3 1339.6 2146.4 3569.4  |
|      | 350  | 0.11 0.32 0.50 0.66 | 915.3 996.2 2521.2 2605.1  |
|      | 400  | 0.16 0.33 0.36 0.33 | 725.1 841.1 1149.9 1109.9  |
|      | 450  | 0.18 0.33 0.36 0.33 | 725.1 841.1 1149.9 1109.9  |
|      | 500  | 0.19 0.33 0.36 0.33 | 725.1 841.1 1149.9 1109.9  |

Table 1: Deviation from best over 40 of best in 10 and average of 10, and standard deviation for CNVNS with $np = 1, 5, 10$ and 15; maximum cpu time set to $t_{max}/np$, where $t_{max}$ is 1400 sec (for $n=1400$), 3000 ($n=3038$), 5000 ($n=5934$) and 10,000 $n=11948$)
### Table 2: Deviation from best over 40 of best over 10, and standard deviation for CNVNS with \( np = 1, 5, 10 \) and 15; for \( n=1400 \) and 3000 sec CPU time

<table>
<thead>
<tr>
<th>( p )</th>
<th>Best in line</th>
<th>( % ) Dev. of the best</th>
<th>Standard deviation</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1 5 10 15</td>
<td>1 5 10 15</td>
<td></td>
</tr>
<tr>
<td>50</td>
<td>507557.0</td>
<td>0.00 0.00 0.00 0.00</td>
<td>60.1 76.8 74.3 62.1</td>
</tr>
<tr>
<td>100</td>
<td>352609.0</td>
<td>0.02 0.00 0.00 0.00</td>
<td>234.3 168.2 236.5 143.3</td>
</tr>
<tr>
<td>150</td>
<td>281227.0</td>
<td>0.06 0.02 0.00 0.00</td>
<td>212.7 72.2 118.2 80.0</td>
</tr>
<tr>
<td>200</td>
<td>238436.0</td>
<td>0.10 0.06 0.02 0.00</td>
<td>56.4 121.7 92.9 91.0</td>
</tr>
<tr>
<td>250</td>
<td>209391.0</td>
<td>0.04 0.01 0.00 0.01</td>
<td>123.3 88.2 52.6 43.3</td>
</tr>
<tr>
<td>300</td>
<td>187734.0</td>
<td>0.09 0.06 0.02 0.00</td>
<td>83.1 47.1 90.9 67.4</td>
</tr>
<tr>
<td>350</td>
<td>171053.0</td>
<td>0.13 0.04 0.04 0.00</td>
<td>113.8 92.1 44.5 70.2</td>
</tr>
<tr>
<td>400</td>
<td>157144.0</td>
<td>0.12 0.07 0.00 0.02</td>
<td>116.7 46.8 80.1 56.9</td>
</tr>
<tr>
<td>450</td>
<td>145517.0</td>
<td>0.08 0.07 0.02 0.00</td>
<td>146.3 122.3 103.3 257.7</td>
</tr>
<tr>
<td>500</td>
<td>135656.0</td>
<td>0.13 0.04 0.00 0.00</td>
<td>113.7 222.0 84.5 84.7</td>
</tr>
</tbody>
</table>

### Table 3: Deviation from best over 40 of average of 10, and standard deviation for CNVNS with \( np = 1, 5, 10 \) and 15; for \( n=1400 \) and 3000 sec CPU time

<table>
<thead>
<tr>
<th>( p )</th>
<th>Best in line</th>
<th>( % ) Dev. of the average</th>
<th>Standard deviation</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1 5 10 15</td>
<td>1 5 10 15</td>
<td></td>
</tr>
<tr>
<td>50</td>
<td>507557.0</td>
<td>0.03 0.02 0.02 0.01</td>
<td>60.1 76.8 74.3 62.1</td>
</tr>
<tr>
<td>100</td>
<td>352609.0</td>
<td>0.15 0.05 0.06 0.05</td>
<td>234.3 168.2 236.5 143.3</td>
</tr>
<tr>
<td>150</td>
<td>281227.0</td>
<td>0.13 0.06 0.05 0.05</td>
<td>212.7 72.2 118.2 80.0</td>
</tr>
<tr>
<td>200</td>
<td>238436.0</td>
<td>0.14 0.10 0.05 0.05</td>
<td>56.4 121.7 92.9 91.0</td>
</tr>
<tr>
<td>250</td>
<td>209391.0</td>
<td>0.11 0.07 0.03 0.04</td>
<td>123.3 88.2 52.6 43.3</td>
</tr>
<tr>
<td>300</td>
<td>187734.0</td>
<td>0.16 0.11 0.09 0.06</td>
<td>83.1 47.1 90.9 67.4</td>
</tr>
<tr>
<td>350</td>
<td>171053.0</td>
<td>0.20 0.12 0.07 0.05</td>
<td>113.8 92.1 44.5 70.2</td>
</tr>
<tr>
<td>400</td>
<td>157144.0</td>
<td>0.24 0.12 0.10 0.07</td>
<td>116.7 46.8 80.1 56.9</td>
</tr>
<tr>
<td>450</td>
<td>145517.0</td>
<td>0.27 0.15 0.10 0.14</td>
<td>146.3 122.3 103.3 257.7</td>
</tr>
<tr>
<td>500</td>
<td>135656.0</td>
<td>0.29 0.18 0.10 0.05</td>
<td>113.7 222.0 84.5 84.7</td>
</tr>
</tbody>
</table>
The graph shows the performance of different processor counts. The x-axis represents the time (in units), and the y-axis represents the performance metric. The lines indicate the performance for 1 processor, 5 processors, 10 processors, and 15 processors.

Key:
- 1 processor: Dotted line
- 5 processors: Solid line
- 10 processors: Dashed line
- 15 processors: Dotted-dashed line

Legend:
- n=3038
- p=500
Summary Results

♣ Parallel procedures give very good solutions

♣ Simple Neighbourhood distribution
  ◇ Very good on medium-size problems
  ◇ Better solutions than the competition
  ◇ Too slow for larger problems
    Similar to sequential and Type 1 approaches

♣ Co-ordinated Neighbourhood distribution
  ◇ Excellent performance
  ◇ Excellent speedups
  ◇ Excellent solution quality
  ◇ Appears scalable
  ◇ Very good solutions obtained early in the parallel exploration
Perspectives

♣ What is *a priori* an appropriate number of processors?

♣ Are all neighbourhoods created equal?

♣ Other parallel strategies

♣ Larger problem instances
REFERENCES


