Performance Measures and Transformations

Computational effort indicators

- CPU time (real time as measured by OS functions)
- number of elementary operations/algorithmic iterations (e.g., search steps, objective function evaluations, number of visited nodes in the search tree, consistency checks, etc.)

Solution quality indicators

- value returned by the cost function (or error from optimum/reference value)
- ranks

Outline

1. Implementation Contest
   Results Task 1

2. Local Search
   Computational Complexity
   Beyond Local Optima

3. Fundamental Search Space Properties
Performance Measures and Transformation

Solution quality indicators
Different instances implies different scales ⇒ need for an invariant measure
  ▶ Distance or error from a reference value:
    
    \[ e_1(x, \pi) = \frac{x(\pi) - \bar{x}(\pi)}{\sigma(\pi)} \]  
    standard score
    
    \[ e_2(x, \pi) = \frac{|x(\pi) - x^{opt}(\pi)|}{x^{opt}(\pi)} \]  
    relative error
    
    \[ e_3(x, \pi) = \frac{x(\pi) - x^{opt}(\pi)}{x'(\pi) - x^{opt}(\pi)} \]  
    invariant error
  
  ▶ optimal value computed exactly or known by instance construction
  ▶ surrogate value such bounds or best known values
  ▶ Rank (no need for normalization but loss of information)

Outline

1. Implementation Contest
   Results Task 1

2. Local Search
   Computational Complexity
   Beyond Local Optima

3. Fundamental Search Space Properties

Visualization Examples

Histogram

Boxplot

Comparative Analysis

View of raw data within each instance
Trade off Solution-Quality vs Run-Time

Needed some definitions on dominance relations

In Pareto sense, for points in $\mathbb{R}^2$

- $\vec{y}_1 \preceq \vec{y}_2$ weakly dominates $y_i^1 \leq y_i^2$ for all $i = 1, 2$
- $\vec{y}_1 \parallel \vec{y}_2$ incomparable neither $\vec{y}_1 \preceq \vec{y}_2$ nor $\vec{y}_2 \preceq \vec{y}_1$
The trade off computation time vs sol quality. Raw data.

View of raw data within each instance

View of raw data ranked within instances and aggregated for the 4 instances
The trade off computation time vs sol quality. Raw data.

Scaling Analysis

Linear regression in log-log plots ⇒ polynomial growth

The trade off computation time vs sol quality. Solution quality ranked within the instances and computation time in raw terms
Comparative visualization

Numerical data

<table>
<thead>
<tr>
<th>Size</th>
<th>DSATUR</th>
<th>RLF</th>
<th>ROS</th>
</tr>
</thead>
<tbody>
<tr>
<td>200</td>
<td>00.08</td>
<td>00.0267</td>
<td>0.0267</td>
</tr>
<tr>
<td>400</td>
<td>0.05067</td>
<td>0.01333</td>
<td>0.01067</td>
</tr>
<tr>
<td>800</td>
<td>0.36002</td>
<td>0.05067</td>
<td>0.004000</td>
</tr>
<tr>
<td>1600</td>
<td>2.7175</td>
<td>0.20268</td>
<td>0.16801</td>
</tr>
<tr>
<td>3200</td>
<td>19.711</td>
<td>0.84805</td>
<td>0.66937</td>
</tr>
</tbody>
</table>

Notes on Experimental Environment

Some organizational hints:
- run a script (bash, perl, python, php) that calls different programs, one for each algorithm to test, on different instances.
- when launched each program writes the search profile in a file (log file or output file).

run a script (bash, perl, python, php) that parses the output files above and put it in a file with the format similar to:

```
071275 181180 191076 230183 240284 250684 270383
0.008 0.00267 0.00267 0.5787 0.00533 0.42933 0.01333
0.032 0.58404 0.02667
```

run a script (bash, perl, python, php) that calls different programs, one for each algorithm to test, on different instances.

- make plots: histograms, boxplots, empirical cumulative distribution functions, correlation/scatter plots
- look at the numerical data and interpret them in practical terms: computation times, distance from optimum
- look for patterns

All the above both at a single instance level and at an aggregate level.

Exploratory Data Analysis

Explore your data:
- make plots: histograms, boxplots, empirical cumulative distribution functions, correlation/scatter plots
- look at the numerical data and interpret them in practical terms: computation times, distance from optimum
- look for patterns

All the above both at a single instance level and at an aggregate level.
Some R Commands

```r
> G <- read.table("Task1.le15.res")
> names(G) <- c("alg","inst","run","sol","time")
> library(lattice)
> bwplot(alg~sol|inst,data=G)
> #original data
> boxplot(sol~alg,data=G,horizontal=TRUE,main="Original data",las=1)
> #standard error
> G$scale <- 0
> split(G$scale, G$inst) <- lapply(split(G$sol, G$inst), scale,center=TRUE,scale=TRUE)
> boxplot(scale~alg,data=G,horizontal=TRUE,main=expression(paste("Standard error: \( \frac{x-bar(x)}{\sqrt{\sigma}} \))))
> library(Hmisc)
> Ecdf(G$scale,group=G$alg,main=expression(paste("Standard error: \( \frac{x-bar(x)}{\sqrt{\sigma}} \))))
> #relative error
> G$opt <- 15
> G$err2 <- (G$sol-G$opt)/G$opt
> boxplot(err2~alg,data=G,horizontal=TRUE,main=expression(paste("Relative error: \( \frac{x-x^{(opt)}}{x^{(opt)}} \))))
> Ecdf(G$err2,group=G$alg,main=expression(paste("Relative error: \( \frac{x-x^{(opt)}}{x^{(opt)}} \))))
> #rank
> T2 <- lapply(T1,rank)
> T3 <- unsplit(T2,list(G$inst))
> T4 <- split(T3,list(G$alg))
> T5b <- stack(T4)
> boxplot(values~ind,data=T5b,horizontal=TRUE,main="Ranks")
> Ecdf(T5b$values,group=T5b$ind,main="Ranks")
```

Computational Complexity of Local Search (1)

For a local search algorithm to be effective, search initialization and individual search steps should be efficiently computable.

Computational Complexity of Local Search (1)

For a local search algorithm to be effective, search initialization and individual search steps should be efficiently computable.

Complexity class \( PLS \): class of problems for which a local search algorithm exists with polynomial time complexity for:

- search initialization
- any single search step, including computation of any evaluation function value
For a local search algorithm to be effective, search initialization and individual search steps should be efficiently computable.

**Complexity class PLS**: class of problems for which a local search algorithm exists with polynomial time complexity for:
- search initialization
- any single search step, including computation of any evaluation function value

For any problem in PLS...
- local optimality can be verified in polynomial time
- improving search steps can be computed in polynomial time

**PLS-complete**: Among the most difficult problems in PLS; if for any of these problems local optima can be found in polynomial time, the same would hold for all problems in PLS.

Some complexity results:
- TSP with $k$-exchange neighborhood with $k > 3$ is PLS-complete.
Computational Complexity of Local Search (2)

\( \mathcal{PLS} \)-complete: Among the most difficult problems in \( \mathcal{PLS} \); if for any of these problems local optima can be found in polynomial time, the same would hold for all problems in \( \mathcal{PLS} \).

Some complexity results:

- TSP with \( k \)-exchange neighborhood with \( k > 3 \) is \( \mathcal{PLS} \)-complete.
- TSP with 2- or 3-exchange neighborhood is in \( \mathcal{PLS} \), but \( \mathcal{PLS} \)-completeness is unknown.

Simple Mechanisms for Escaping from Local Optima

- **Restart**: re-initialize search whenever a local optimum is encountered.
  (Often rather ineffective due to cost of initialization.)

- **Non-improving steps**: in local optima, allow selection of candidate solutions with equal or worse evaluation function value, e.g., using minimally worsening steps.
  (Can lead to long walks in plateaus, i.e., regions of search positions with identical evaluation function.

Note: Neither of these mechanisms is guaranteed to always escape effectively from local optima.
Diversification vs Intensification

- Goal-directed and randomized components of LS strategy need to be balanced carefully.

Intensification: aims to greedily increase solution quality or probability, e.g., by exploiting the evaluation function.

Diversification: aim to prevent search stagnation by preventing search process from getting trapped in confined regions.

Examples:

- Iterative Improvement (II): intensification strategy.
- Uninformed Random Walk/Picking (URW/P): diversification strategy.

Balanced combination of intensification and diversification mechanisms forms the basis for advanced LS methods.
Diversification vs Intensification

- Goal-directed and randomized components of LS strategy need to be balanced carefully.
- **Intensification**: aims to greedily increase solution quality or probability, e.g., by exploiting the evaluation function.
- **Diversification**: aim to prevent search stagnation by preventing search process from getting trapped in confined regions.

Examples:

- Iterative Improvement (II): *intensification* strategy.
- Uninformed Random Walk/Picking (URW/P): *diversification* strategy.

Balanced combination of intensification and diversification mechanisms forms the basis for advanced LS methods.

Learning Goals

- Review basic concepts
- Try to **understand** the relationships between search space features and LS performance.
- Learn about techniques and goals of search space analysis.

Concepts that Define the Search Landscape

- Search space $S$
- Neighborhood structure $\mathcal{N} \subseteq S \times S$
- Evaluation Function $g(\pi) : S \mapsto \mathbb{R}$

**Definition:**

The **Search Landscape** $\mathcal{L}(\pi)$ of $\pi$ is the triple

$$\mathcal{L}(\pi) := \langle S(\pi), N(\pi), g(\pi) \rangle$$
Ideal visualization of metaheuristic principles

- Simplified landscape representation
- Tabu Search
- Guided Local Search
- Iterated Local Search
- Evolutionary Alg.