15 new flat instances created

270383
191076
RLF
DSATUR
230183
141179
240284
ROS

76
60
50

270383
191076
RLF
DSATUR
230183
141179
240284
ROS

−1.5 −0.5 0.0 0.5 1.0 1.5
Standard error:

x −− ... 2 3 4 5 6 7 8
Ranks

270383
191076
RLF
DSATUR
230183
141179
240284
ROS

1 2 3 4 5 6 7 8
Ranks

Algorithm flat-1000-50 flat-1000-60 flat-1000-76
270383 98 98 99
191076 105 104 105
RLF 104 105 105
DSATUR 111 111 111
230183 114 115 114
141179 115 115 115
240284 116 116 116
ROS 120 120 120

Experimental Set up

15 new flat instances created

<table>
<thead>
<tr>
<th>Type</th>
<th># instances</th>
<th>Upper bound</th>
</tr>
</thead>
<tbody>
<tr>
<td>flat-1000-50-0-?.col</td>
<td>5</td>
<td>50</td>
</tr>
<tr>
<td>flat-1000-60-0-?.col</td>
<td>5</td>
<td>60</td>
</tr>
<tr>
<td>flat-1000-76-0-?.col</td>
<td>5</td>
<td>76</td>
</tr>
</tbody>
</table>

each algorithm run once on each of the 15 new instances

Fairness principle: same computational resources to all algorithms

90 seconds on Intel(R) Celeron(R) CPU 2.40GHz, 1GB RAM (120 seconds for 230183)

restart ROS heuristic used as reference algorithm

restart RLF and DSATUR also included

Outline

1. Results Task 2
2. Dynamic Local Search
3. Iterated Local Search
4. Exercise
5. Evolutionary Algorithms
Program Profiling

- Plot the development of
  - best visited solution quality
  - current solution quality
  over time and compare with other features of the algorithm.

- Profile time consumption per program components
  under Linux: `gprof`
  1. add flag `-pg` in compilation
  2. run the program
  3. `gprof program-file > a.txt`

Dynamic Local Search

- Key Idea: Modify the evaluation function whenever a local optimum is encountered.
- Associate penalty weights (penalties) with solution components; these determine impact of components on evaluation function value.
- Perform Iterative Improvement; when in local minimum, increase penalties of some solution components until improving steps become available.

Dynamic Local Search (DLS):

determine initial candidate solution \( s \)
initialize penalties
While termination criterion is not satisfied:
  - compute modified evaluation function \( g' \) from \( g \) based on penalties
  - perform subsidiary perturbative search on \( s \) using evaluation function \( g' \)
  - update penalties based on \( s \)

Potential problem:
Solution components required for (optimal) solution may also be present in many local minima.

Possible solutions:

A: Occasional decreases/smoothing of penalties.
B: Only increase penalties of solution components that are least likely to occur in (optimal) solutions.

Implementation of B:
[Voudouris and Tsang, 1995] Only increase penalties of solution components \( i \) with maximal utility:

\[
\text{util}(s', i) := \frac{g_i(\pi, s')}{1 + \text{penalty}(i)}
\]

where \( g_i(\pi, s') \) is the solution quality contribution of \( i \) in \( s' \).

Dynamic Local Search (continued)

- Modified evaluation function:
  
  \[
g'(\pi, s) := g(\pi, s) + \sum_{i \in SC(\pi', s)} \text{penalty}(i),
  \]

  where \( SC(\pi', s) \) is the set of solution components of problem instance \( n' \) used in candidate solution \( s \).

- Penalty initialization: For all \( i \): \( \text{penalty}(i) := 0 \).
- Penalty update in local minimum \( s \): Typically involves penalty increase of some or all solution components of \( s \); often also occasional penalty decrease or penalty smoothing.
- Subsidiary perturbative search: Often Iterative Improvement.

Example: Guided Local Search (GLS) for the TSP
[Voudouris and Tsang 1995; 1999]

- Given: TSP instance \( G \)
- Search space: Hamiltonian cycles in \( G \) with \( n \) vertices;
- Neighborhood: 2-edge-exchange;
- Solution components edges of \( G \);
  \( g_e(G, p) := w(e) \);
- Penalty initialization: Set all edge penalties to zero.
- Subsidiary perturbative search: Iterative First Improvement.
- Penalty update: Increment penalties for all edges with maximal utility by
  \[
  \lambda := 0.3 \cdot \frac{w(s_{2-opt})}{n}
  \]
  where \( s_{2-opt} \) = 2-optimal tour.

Outline

1. Results Task 2
2. Dynamic Local Search
3. Iterated Local Search
4. Exercise
5. Evolutionary Algorithms

Hybrid Methods

Combination of ‘simple’ methods often yields substantial performance improvements.

Simple examples:
- Commonly used restart mechanisms can be seen as hybridisations with Uninformed Random Picking
- Iterative Improvement + Uninformed Random Walk = Randomized Iterative Improvement
Iterated Local Search

Key Idea: Use two types of LS steps:
- subsidiary perturbative (local) search steps for reaching local optima as efficiently as possible (intensification)
- perturbation steps for effectively escaping from local optima (diversification).

Also: Use acceptance criterion to control diversification vs intensification behavior.

Iterated Local Search (ILS):
- determine initial candidate solution s
- perform subsidiary perturbative search on s
- While termination criterion is not satisfied:
  - perform perturbation on s
  - perform subsidiary perturbative search on s
  - based on acceptance criterion, keep s or revert to r := t

Note:
- Subsidiary perturbative search results in a local minimum.
- ILS trajectories can be seen as walks in the space of local minima of the given evaluation function.
- Perturbation phase and acceptance criterion may use aspects of search history (i.e., limited memory).
- In a high-performance ILS algorithm, subsidiary perturbative search, perturbation mechanism and acceptance criterion need to complement each other well.

Perturbation mechanism:
- Needs to be chosen such that its effect cannot be easily undone by subsequent perturbative search phase.
  (Often achieved by search steps larger neighborhood.)
- Example: perturbative search = 3-opt, perturbation = 4-exchange steps in ILS for TSP.
- A perturbation phase may consist of one or more perturbation steps.
- Weak perturbation ⇒ short subsequent perturbative search phase; but:
  - risk of revisiting current local minimum.
- Strong perturbation ⇒ more effective escape from local minima; but:
  - may have similar drawbacks as random restart.
- Advanced ILS algorithms may change nature and/or strength of perturbation adaptively during search.

Example: Iterated Local Search for the TSP (1)
- **Given**: TSP instance G.
- **Search space**: Hamiltonian cycles in G.
- **Subsidiary perturbative search**: Lin-Kernighan variable depth search algorithm
- **Perturbation mechanism**: 'double-bridge move' = particular 4-exchange step:
- **Acceptance criterion**: Always return the best of the two given candidate round trips.

Example: Iterated Local Search for the TSP (2)
- Note:
  - Double-bridge move perturbation cannot be directly reversed by a sequence of 2-exchange steps as performed by “usual” LK implementations.
  - This perturbation is empirically shown to be effective independent of instance size.

Note:
- This ILS algorithm for the TSP is known as Iterated Lin-Kernighan (ILK) Algorithm.
- Although ILK is structurally rather simple, an efficient implementation was shown to achieve excellent performance [Johnson and McGeoch, 1997].

Iterated local search algorithms . . .
- are typically rather easy to implement (given existing implementation of subsidiary simple LS algorithms);
- achieve state-of-the-art performance on many combinatorial problems, including the TSP.

There are many LS approaches that are closely related to ILS, including:
- Large Step Markov Chains [Martin et al., 1991]
- Chained Local Search [Martin and Otto, 1996]
- Variants of Variable Neighbourhood Search (VNS) [Hansen and Mladenović, 2002]
1. Results Task 2
2. Dynamic Local Search
3. Iterated Local Search
4. Exercise
5. Evolutionary Algorithms

Evolutionary Algorithms

**Key idea** (Inspired by Darwinian model of biological evolution): Maintain a population of individuals that compete for survival, and generate new individuals, which in turn again compete for survival.

Iteratively apply genetic operators mutation, recombination, selection to a population of candidate solutions.

- **Mutation** introduces random variation in the genetic material of individuals.
- **Recombination** of genetic material during reproduction produces offspring that combines features inherited from both parents.
- **Differences in evolutionary fitness** lead selection of genetic traits ('survival of the fittest').

**Evolutionary Algorithm (EA):**

| Initial population \( sp \)
<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>While termination criterion is not satisfied:</td>
</tr>
<tr>
<td><strong>generate set</strong> ( spr ) of new candidate solutions by recombination</td>
</tr>
<tr>
<td><strong>generate set</strong> ( spm ) of new candidate solutions from ( spr ) and ( sp ) by mutation</td>
</tr>
<tr>
<td><strong>select</strong> new population ( sp ) from candidate solutions in ( sp ), ( spr ), and ( spm )</td>
</tr>
</tbody>
</table>

**Solution Approaches for Graph Coloring**

- Design an application of SA to GCP
- Design an application of TS to GCP

Different choices for the candidate solutions, neighborhood structures and evaluation function define different approaches to the problem:

| \( k \)-fixed complete proper | ++ |
| \( k \)-fixed partial proper | + |
| \( k \)-fixed complete improper | + + + |
| \( k \)-variable complete proper | ++ |
| \( k \)-variable partial proper | + |
| \( k \)-variable complete improper | + + |
| \( k \)-variable partial improper | - |

Population-based LS Methods

LS methods discussed so far manipulate one candidate solution of given problem instance in each search step.

**Straightforward extension:** Use population (i.e., set) of candidate solutions instead.

**Note:**
- The use of populations provides a generic way to achieve search diversification.

Evolutionary Algorithm (EA):

| Individual | Solution to a problem |
| Population | Set of Solutions |
| Fitness | Quality of a solution |
| Chromosome | Representation for a solution (e.g., set of parameters) |
| Gene | Part of the representation of a solution (e.g., parameter or degree of freedom) |
| Crossover Mutation | Search Operators |
| Natural Selection | Promoting the reuse of good solutions |

**Selection**

<table>
<thead>
<tr>
<th>Time ( t )</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>String 1</strong></td>
</tr>
<tr>
<td><strong>String 2</strong></td>
</tr>
<tr>
<td><strong>String 3</strong></td>
</tr>
<tr>
<td><strong>String 4</strong></td>
</tr>
<tr>
<td><strong>String 5</strong></td>
</tr>
<tr>
<td><strong>...</strong></td>
</tr>
<tr>
<td><strong>String n</strong></td>
</tr>
</tbody>
</table>

**Crossover & Recombination**

<table>
<thead>
<tr>
<th>Time ( t )</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Child 1(1&amp;2)</strong></td>
</tr>
<tr>
<td><strong>Child 2(1&amp;2)</strong></td>
</tr>
<tr>
<td><strong>Child 1(2&amp;3)</strong></td>
</tr>
<tr>
<td><strong>Child 2(2&amp;3)</strong></td>
</tr>
<tr>
<td>** intermediate**</td>
</tr>
<tr>
<td><strong>Time ( t+1 )</strong></td>
</tr>
</tbody>
</table>
Problem: Pure evolutionary algorithms often lack capability of sufficient search intensification.

Solution: Apply subsidiary perturbative search after initialization, mutation and recombination.

⇒ Memetic Algorithms (aka Evolutionary Local Search)


- transmission of memes, mimicking cultural evaluation which is supposed to be direct and Lamarckian

Solution representation

- neat separation between solution encode or representation (genotype) from actual variables (phenotype)
- a solution \( s \in S \) is represented by a string that is: the genotype set is made of strings of length \( l \) whose elements are symbols from an alphabet \( A \) such that there exists a map:
  \[
  c : A^l \rightarrow S
  \]
  - the elements of strings are the genes
  - the values of elements can take are the alleles
- the search space is then \( \mathcal{X} \subseteq A^l \)
- if the strings are member of a population they are called chromosomes and their recombination crossover
- strings are evaluated by \( f(c(x)) = g(x) \) which gives them a fitness

⇒ binary representation is appealing but not always good (e.g., in constrained problems binary crossovers might not be good)

Conjectures on the goodness of EA

schema: subset of \( A^l \) where strings have a set of variables fixed. Ex.: \( 1 \ast \ast 1 \)

- exploit intrinsic parallelism of schemata
- Schema Theorem:
  \[
  E[N(S, t + 1)] \geq \frac{f(S, t)}{f(S)} N(S, t)[1 - c(S, t)]
  \]
- a method for solving all problems ⇒ disproved by the No Free Lunch Theorems
- building block hypothesis

Selection

Main idea: selection should be related to fitness

- Fitness proportionate selection (Roulette-wheel method)
  \[
  p_i = \frac{f_i}{\sum_i f_i}
  \]
- Tournament selection: a set of chromosomes is chosen and compared and the best chromosome chosen.
- Rank based and selection pressure

Initial Population

- Which size? Trade-off
- Minimum size: connectivity by recombination is achieved if at least one instance of every allele is guaranteed to be present at each locus. Ex: if binary:
  \[
  P_{s}^{2} = (1 - (0.5)^{M-1})^{l}
  \]
  for \( l = 50 \), it is sufficient \( M = 17 \) to guarantee \( P_{s}^{2} > 99.9\% \).
- Often: independent, uninform random picking from given search space.
- Attempt to cover at best the search space, eg. Latin hypercube.
- But: can also use multiple runs of construction heuristic.

Recombination (Crossover)

- Binary or assignment representations
  - one-point, two-point, m-point (preference to positional bias w.r.t. distributional bias
  - uniform cross over (through a mask controlled by a Bernoulli parameter \( p \))
- Non-linear representations
  - (Permutations) Partially mapped crossover
  - (Permutations) mask based
- More commonly ad hoc crossovers are used as this appear to be a crucial feature of success
- Two off-springs are generally generated
- Crossover rate controls the application of the crossover. May be adaptive: high at the start and low when convergence
Example: crossovers for binary representations

Mutation

- Goal: Introduce relatively small perturbations in candidate solutions in current population + offspring obtained from recombination.
- Typically, perturbations are applied stochastically and independently to each candidate solution; amount of perturbation is controlled by mutation rate.
- Mutation rate controls the application of bit-wise mutations. May be adaptive: low at the start and high when convergence.
- Possible implementation through Poisson variable which determines the minimum of all functions as is hill climbing.
- The role of mutation (as compared to recombination) in high-performance evolutionary algorithms has been often underestimated.

New Population

- Determines population for next cycle (generation) of the algorithm by selecting individual candidate solutions from current population + new candidate solutions obtained from recombination, mutation (+ subsidiary perturbative search).
- Goal: Obtain population of high-quality solutions while maintaining population diversity.
- Selection is based on evaluation function (fitness) of candidate solutions such that better candidate solutions have a higher chance of ‘surviving’ the selection process.
- It is often beneficial to use elitist selection strategies, which ensure that the best candidate solutions are always selected.
- Most commonly used: steady state in which only one new chromosome is generated at each iteration.
- Diversity is checked and duplicates avoided.

Types of evolutionary algorithms

- Genetic Algorithms (GAs) [Holland, 1975; Goldberg, 1989]:
  - have been applied to a very broad range of (mostly discrete) combinatorial problems;
  - often encode candidate solutions as bit strings of fixed length, which is now known to be disadvantageous for combinatorial problems such as the TSP;
- Evolution Strategies [Rechenberg, 1973; Schwefel, 1981]:
  - originally developed for (continuous) numerical optimization problems;
  - operate on more natural representations of candidate solutions;
  - use self-adaptation of perturbation strength achieved by mutation;
  - typically use elitist deterministic selection.
- Evolutionary Programming [Fogel et al., 1966]:
  - similar to Evolution Strategies (developed independently), but typically does not make use of recombination and uses stochastic selection based on tournament mechanisms;
  - often seek to adapt the program to the problem rather than the solutions.

Theoretical studies

- Through Markov chains modelling some versions of evolutionary algorithms can be made to converge with probability 1 to the best possible solutions in the limit [Fogel, 1992; Rudolph, 1994].
- Convergence rates on mathematically tractable functions or with local approximations [Bäck and Hoffmeister, 2004; Beyer, 2001].
- “No Free Lunch Theorem” [Wolpert and Macready, 1997]. On average, within some assumptions, blind random search is as good at finding the minimum of all functions as hill climbing. However:
  - These theoretical findings are not very practical.
  - EAs are made to produce useful solutions rather than perfect solutions.
Research Goals

- Analyzing classes of optimization problems and determining the best components for evolutionary algorithms.
- Applying evolutionary algorithms to problems that are dynamically changing.
- Gaining theoretical insights for the choice of components.