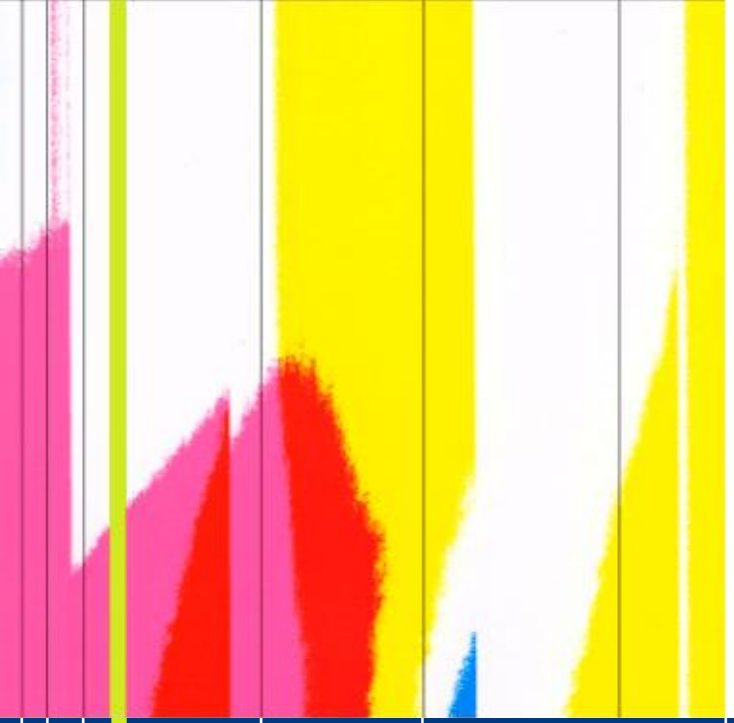


ParadisEO-MO

Design and Unification of Local Search

01/04/2010

INSTITUT NATIONAL
DE RECHERCHE
EN INFORMATIQUE
ET EN AUTOMATIQUE

INRIA

centre de recherche
LILLE - NORD EUROPE

Outline

- Previous design
- New concepts: “Neighbor”, “Neighborhood” and “Evaluation”
- General schema of Local Search algorithms (LS)
- Classical LS: improvements
- Variable Neighborhood Search (VNS)
- Iterated Local Search (ILS)
- Statistics and checkpointing : Fitness Landscape Analysis
- Example of HC (and more) on the FlowShop Problem
- Conclusion and perspectives

Previous design

- Based on “Move”
  - moMove
  - moMoveInit
  - moNextMove: **test** and **compute** the next move
  - moMoveIncrEval: hard to use full evaluation, only fitness is computed (Can’t save others modifications on sub-solutions)

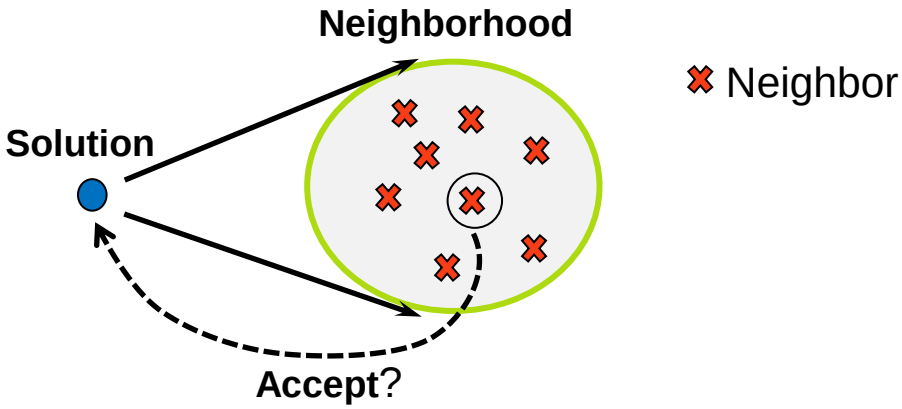
    ⇒ Some bugs appear in limit cases (Initialisation, …)
- Each LS is specific
  - Management of the exploration process using “Move”
  - LS templates of HC ≠ TS ≠ SA …

    ⇒ Users LS from: “eoMonOp” and “Move”
- No standard tools to trace the LS process
  - No checkpointing like in ParadisEO-EO

New concepts: "Neighbor", "Neighborhood" and "Evaluation"

- "Neighbor"
  - Replace "moMove"
  - Move + save neighbor informations (fitness and more)
- "Neighborhood"
  - Replace "moMoveInit" and "moMoveNext"
  - Describe how to compute all the neighbors
  - Methods: init, next, cont and hasNeighbor
- "Evaluation"
  - Replace "moIncrEval"
  - "moEval": set fitness (and more) of the neighbor
  - Can be incremental or full evaluation

General schema of LS



Search explorer:

- Generate neighbors from the neighborhood
- Select a neighbor
- Decide to replace solution by the selected neighbor

Simplified schema of LS

Constructors:

- moNeighborhood(moNeighbor)
- moSearchExplorer(moNeighborhood, moEval)

```
do {
        searchExplorer(solution)
        if ( searchExplorer.accept(solution) )
                searchExplorer.move(solution)
}
while (searchExplorer.continue(solution))
```

Completed schema of LS

```
searchExplorer.initParam(solution)
continuator.init(solution)
do {
        searchExplorer(solution)
        if ( searchExplorer.accept(solution) )
                searchExplorer.move(solution)
        searchExplorer.updateParam(solution)
}
while (continuator(solution) && searchExplorer.continue(solution))
searchExplorer.terminate(solution)
```

Classical LS: improvements

- Hill-climbing like:
  - Simple HC (best improvement)
  - Simple HC neutral (random choice among several best solutions) NEW
  - HC neutral (+ allows move when best solution has equal fitness) NEW
  - First Improvement HC
- Walk like to sample search space: NEW
  - Random walk
  - Random neutral walk
  - Metropolis hasting
- Simulated Annealing

  cooling schudeling: init, decreases, final NEW
- Tabu Search

  "*moMemory" : diversification, intensification, tabu "list"* NEW

Classical LS: improvements

- Hill-climbing like:
  - Simple HC (best improvement)
  - Simple HC neutral (random choice among several best solutions) NEW
  - HC neutral (+ allows move when best solution has equal fitness) NEW
  - First Improvement HC
- Walk like to sample search space: NEW
  - Random walk
  - Random neutral walk
  - Metropolis hasting
- Simulated Annealing
  - *cooling scheduling: init, decreases, final* NEW
- Tabu Search
  - *"moMemory" : diversification, intensification, tabu "list"* NEW

Modularity improved
Easy reuse of basic ls

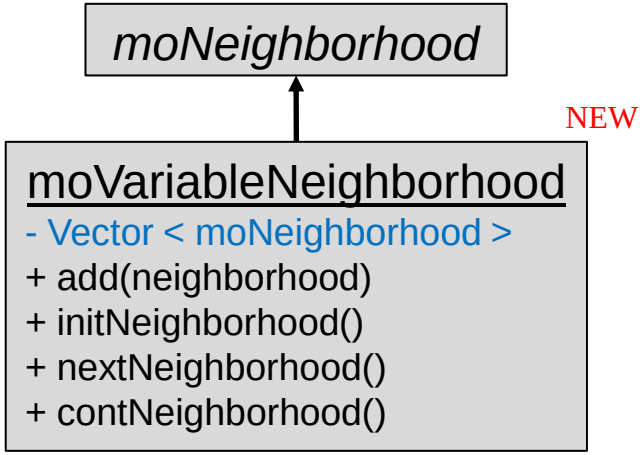
Variable Neighborhood Search (VNS)

Previous design:

Iteration on k “eoMonOp” (stochastic operator of ParadisEO-EO)

Actual design:

A Single explorer based on a **Variable Neighborhood**



Iterated Local Search

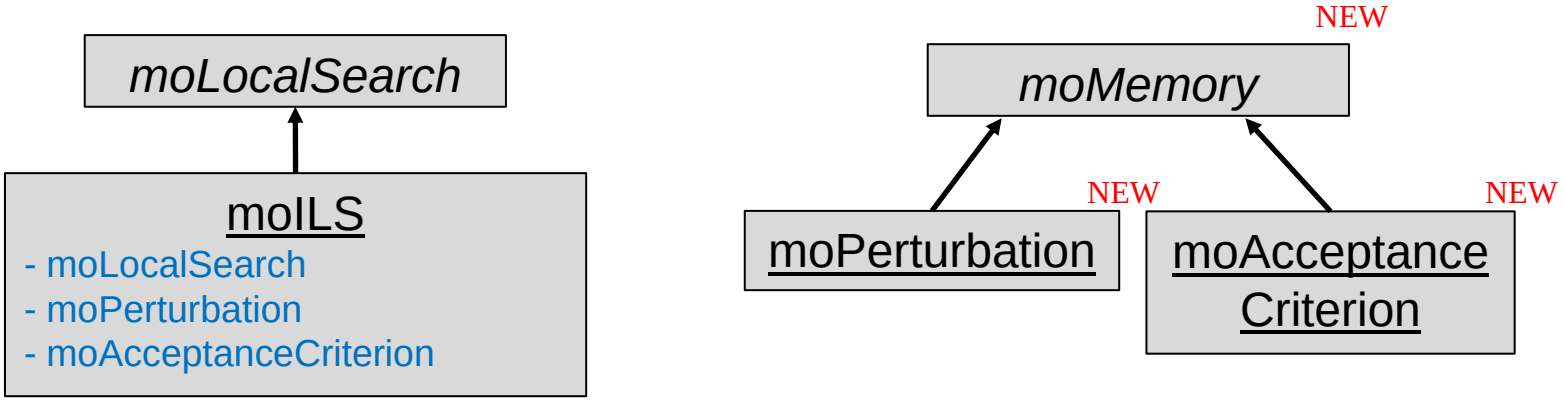
Previous design:

No Memory

Actual design:

Explorer based on a LS

Perturbation and acceptance criterion share Memory (“moMemory”)



Statistics and checkpointing : Fitness Landscape Analysis

Previous design

Nothing

Actual design[NEW]

- Checkpointing like in ParadisEO-EO is available
  - moCheckpoint
  - moContinuator
  - moStatBase
    - Statistics in the Neighborhood : min, max, mean, standard deviation, probability to increase, neutral degree…
    - Autocorrelation of fitness, fitness cloud, FDC…

How to use ParadisEO-MO?
What I need define?

Only problem dependent components:

- Solution representation *EOT*
- “Random” initialisation operator *eoInit*
- Evaluation Functions
  - full Evaluation *eoEvalFunc*
  - incremental (if possible) *moEval*
- Neighbor *moNeighbor*
- Neighborhood
  - *moNeighborhood*
  - or a mapping (int -> Neighbor) using an *moIndexNeighborhood* NEW

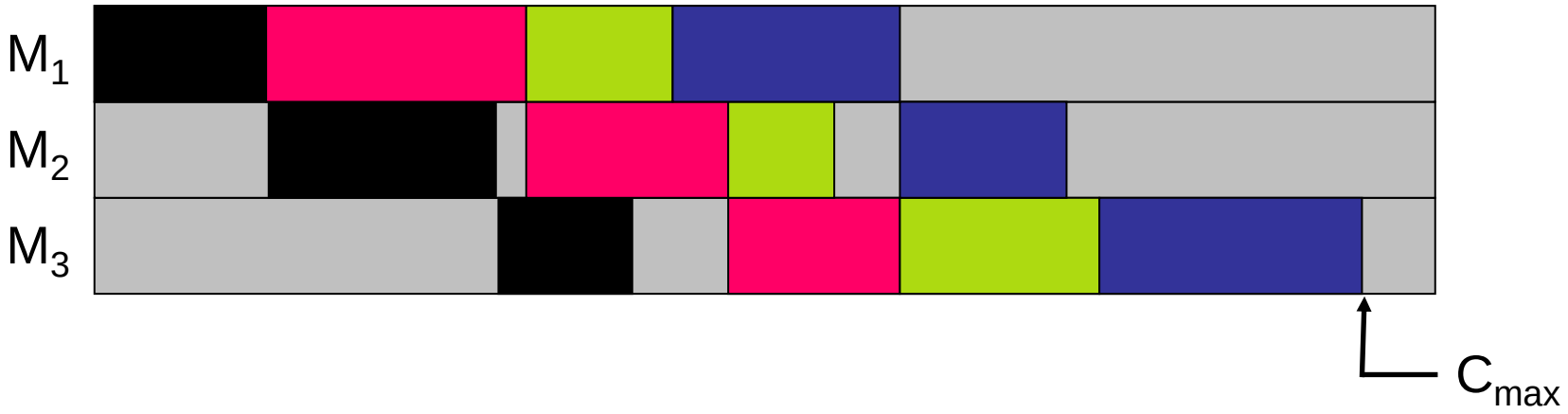
Predefined components for the classical search spaces: NEW

- Bit string
- Permutation

Example of HC (and more) on the FlowShop Problem

FlowShop

- N jobs to schedule on M machines
- Machines are critical resources
- Minimize the makespan (C_{max})



Example of HC (and more) on the FlowShop Problem

Solution representation

- Permutation *eoInt<eoMinimizingFitness>*

Random initialisation

- Random permutation *eoInitPermutation*

Full Evaluation

- To be defined by user

No incremental Evaluation

- *moFullEvalByCopy* NEW or *moFullEvalByModif* NEW can be used with the full evaluation function

Neighbor

- *moShiftPermutation* NEW or *moSwapPermutation* NEW

Example of HC (and more) on the FlowShop Problem

Example for swap:

- n : current neighbor
- first, second: indexes to swap
- size : Permutation size

```
class moSwapNeighborhood : public moNeighborhood

void init(sol, n){ n.first =0; n.second =1;}

void next(sol, n){
        if (n.second < size -1)
                n.second++;
        else
                n.first++; n.second=n.first+1;
}

bool cont(sol,n){ return (n.first < size-2); }
```

Example of HC (and more) on the FlowShop Problem

New possibility: used an indexed neighbor and neighborhood

NEW

```
class moSwapNeighbor : public moIndexNeighbor

void move(sol){
        first= f1(key);
        second= f2(key);
        swap(sol, first, second);
}
```

key	f1	f2
0	0	1
1	0	2
2	0	3
3	1	2
4	1	3
5	2	3

NEW

3 indexed neighborhoods available:

- *moOrderNeighborhood*
- *moRndWithReplNeighborhood*
- *moRndWithoutReplNeighborhood*

Example of HC (and more) on the FlowShop Problem

Now we can declare the HC !!!

- *moNeighborComparator nComp;*
- *moSolNeighborComparator solnComp;*
- ***moHC** hc(neighborhood, eval, nComp, solnComp);*

• And more a TS

- *moIntensification intens;*
- *moDiversification div;*
- *moBetterAcceptCrit aspiration;*
- *moSolVectorTabuList tabuList;*
- ***moTS** ts(neighborhood, eval, nComp, solnComp, tabuList, intens, div, aspiration);*

hc(sol); ts(sol); …… moILS ilsts(ts, perturb, accept); …. Ilsts(sol);

Conclusion

- More reusable design (“Neighbor”, “Neighborhood”, …)
- 11 stantard local search vs 6 before
- More than 70 classes (≈ 7000 code lines) vs 40 before
- All classes are tested and documentation available
- Checkpointing is now available: fitness landscape analysis
- Standard components added (Bit string, permutation and so…)
- …A lot of NEW

Perspectives

- Connection with GPGPU
- Create tool boxes for automatics fitness landscape analysis
- Extend to multiobjective optimization: set-based local search
- Create problems repository
- Create tools for automatics parameters tunings and parameters control
- Create /interface with exact methods