

Constraint Programming and Local Search

Filippo Focacci, Andrea Lodi

CP -AI-OR'02, School on Optimization

Outline

- Introduction
- A didactic optimization problem (dTP)
 - Motivations for cooperation
- A zoo of CP / LS hybrids
 - Sequential combination
 - Master / sub-problem decomposition
 - Improved neighborhood exploration
 - Neighborhoods in CP
 - Large neighborhood search
 - Local moves over a heuristic
 - Local moves during construction

page 2

What this tutorial addresses

- Solving large hard combinatorial optimization problems
- Systematic description of ways of combining LS and CP techniques
- Goal: provide a check-list of recipes that can be tried when tackling a new optimization application
- Illustrated on a didactic problem

page 3

When should you enquire about CP / LS hybrids ?

- When you have:
 - A large complex optimization problem
 - No solution neither with CP nor with LS
 - The problem specification may change over time
- Best in case of strong execution requirements
 - Limited planning resource
 - On-line optimization

page 4

When can't it help ?

- When modeling is the issue
- When optimization is the single difficulty
- When thousands of man.year have been spent studying your very problem
=> useless for solving a 1M node TSP

page 5

Comparing CP and LS

- Constraint Programming
 - Solves complex problems
 - Models capturing many side constraints
 - Solves by global search and propagation
- Local search
 - Solves problems with simple models
 - Efficiency: quick first solution, rapid early convergence

page 6

Opportunities for collaboration

- Expected combination of :
 - Generality (solve complex problems)
 - Nice modeling
 - Generic methods from the model
 - Easy to add/modify constraints
 - Efficiency (solve them fast)
 - Initial solution
 - Quick convergence
 - Address both feasibility and optimization issues
 - keep constraints hard
- Difficulty to combine:
 - monotonic reasoning (CP)
 - non-monotonic modifications (LS)

page 7

Outline

- Introduction
- A didactic transportation problem (dTP)
 - Motivations for cooperation
- A zoo of CP / LS hybrids
 - Sequential combination
 - Master / sub-problem decomposition
 - Improved neighborhood exploration
 - Neighborhoods in CP
 - Large neighborhood search
 - Local moves over a heuristic
 - Local moves during construction

page 8

A didactic transportation problem

Collect goods from clients

- Set of trucks located in a depot
- Each truck can carry two bins
- Each bin may contain only goods from the same type
- Clients have time window constraints
- Bins have capacity constraints

page 9

A simple model for dTP

$i, j \in \{1, \dots, n\}$: clients (their locations)

$k \in \{1, \dots, M\}$: trucks

$h \in \{1, \dots, 2M\}$: bins

$l \in \{1, \dots, P\}$: types of goods

page 10

Model

Minimize $totCost = \sum_{k=1}^M cost_k$

On $\forall k, cost_k \geq 0$

$truck_k: UnaryResource(tt, c, cost_k)$

$\forall h, collects_h \in [1 .. P]$

$\forall i, start_i \in [a_i .. b_i]$

$service_i: Activity(start_i, d_i, i)$

$visitedBy_i \in [1 .. M]$

$collectedIn_i \in [1 .. 2M]$

page 11

Model (2)

Subject to

$\forall i, service_i, requires\ truck[visitedBy_i]$

$\forall h, \sum_{i | collectedIn\ i = h} q_i \leq C$

$\forall i, collects[collectedIn_i] = type_i$

$\forall i, visitedBy_i = \lceil collectedIn_i / 2 \rceil$

page 12

A CP approach

- Strengthen the model
 - Add redundant constraints
 - Add global constraints
 - Add constraints evaluating the cost of the solutions
 - Symmetry breaking (dominance) constraints
- Find a search heuristic
 - Variable / value orderings
 - Explore part of the search tree through Branch and Bound

page 13

A CP approach

Redundant models for stronger propagation

Example: redundant routing model

$\forall k, \text{ first}_k \in [1 .. N]$

$\forall i, \text{ next}_i \in [1 .. N+M]$

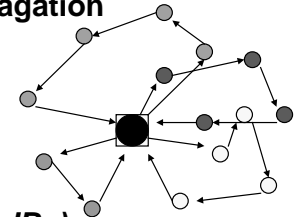
$\text{succ}_i \in [\{\} .. \{1, \dots, N\}]$

$\text{multiPath}(\text{first}, \text{next}, \text{succ}, \text{visitedBy})$

$\text{costPaths}(\text{first}, \text{next}, \text{succ}, c, \text{totCost})$

$\forall i, j, j \in \text{succ}_i \Leftrightarrow$

$(\text{visitedBy}_i = \text{visitedBy}_j \wedge \text{start}_i > \text{start}_j)$



page 14

Solving through CP

- Instantiate visitedBy_i
- Rank all activities on the routes
(instantiate $\text{next}_i / \text{succ}_i$)
- Instantiate start_i to their earliest possible value

page 15

Difficulties with CP

- Poor global reasoning
- Poor cost anticipation
- Goes backtracking « forever »
- As propagation is strengthened, the model is slowed down

page 16

A local search approach

- Two possibilities:
 - Work in the space of feasible solutions
 - Accept infeasible solutions by turning constraints into preferences
- Choose the first method, by adding, if needed, extra resources (trucks and bins)

page 17

Local search for dTP

- Generate an initial solution
 - Select clients i in random order
 - Assign it to a truck that has a bin of $type_i$, or to a truck that can be added an extra bin of $type_i$
- Move from a solution to one of its neighbors, in order to improve the objective

page 18

Neighborhoods for dTP

- Node transfer:
 - Change values of $visitedBy_i$ and $collectedIn_i$ for some i
- Bin swap:
 - Select bins h_1, h_2 on trucks $k_1 = \lceil h_1/2 \rceil, k_2 = \lceil h_2/2 \rceil$
 - For all clients i ,
 - $collectedIn_i = h_1 \Rightarrow collectedIn_i = h_2, visitedBy_i = k_2$
 - $collectedIn_i = h_2 \Rightarrow collectedIn_i = h_1, visitedBy_i = k_1$
 - Swap $collects_{h_1}$ and $collects_{h_2}$

page 19

Neighborhoods

- k -opt:
 - select i_1, i_2, i_3 such that $visitedBy_{i_1} = visitedBy_{i_2} = visitedBy_{i_3}$
 - Exchange edges:
 - Replace $next_{i_1}=j_1, next_{i_2}=j_2, next_{i_3}=j_3$
 - By $next_{i_1}=j_2, next_{i_2}=j_3, next_{i_3}=j_1$

page 20

Driving the local search process

- **Main iteration:**
 - Until a global stopping criterion is met:
 - generate a new initial solution
 - perform a local walk
- **Each walk:**
 - Until a local criterion is met:
 - Iterate the neighborhood, until a neighbor satisfying all constraints as well as the acceptance criterion is found
 - Perform the move

page 21

Difficulties with LS

- **Generating a good feasible first solution becomes harder**
- **Explore neighborhoods**
 - takes longer: constraints checks
 - is less interesting: fewer valid nodes
 - more local optima appear

page 22

Conclusion

- **Neither of the “pure” approaches works**
- **Need for hybridization with other techniques**
 - Try a cooperation between CP and LS
 - Expect to retain :
 - *Good sides of CP*: handling side constraints, building valid solutions, systematic search
 - *Good sides of LS*: quick easy improvements, quick convergence.

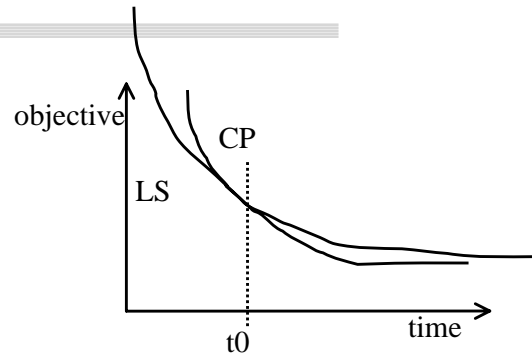
page 23

Outline

- **Introduction**
- **A didactic optimization problem (dTP)**
 - Motivations for cooperation
- **A zoo of CP / LS hybrids**
 - Sequential combination
 - Master / sub-problem decomposition
 - Improved neighborhood exploration
 - Neighborhoods in CP
 - Large neighborhood search
 - Local moves over a heuristic
 - Local moves during construction

page 24

Sequential combination: LS-CP



- Use local search for the beginning of the optimization descent – switch to CP at time t_0

page 25

Discussion

- A good idea
 - When the feasibility problem is easy
 - For time-constrained optimization
- But, the switch from LS to CP is not immediate
 - CP starts with a good upper bound, but without no-goods
- On the didactic Transportation Problem (dTP)
 - Lack of good lower bounds
 - ⇒ Systematic CP search is stuck near the optimal region

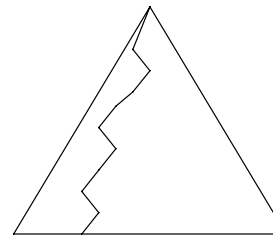
page 26

Sequential combination: CP-LS

- Build a first feasible solution with CP
 - Greedy heuristic
- Try to improve it through LS
 - Constraints can be softened to support dense neighborhoods

page 27

Greedy insertion algorithm

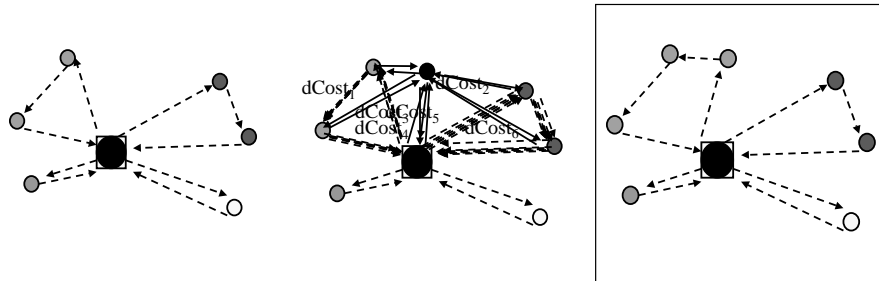


- At each choice point a function h is evaluated for all possible choices
- The choice that minimizes h is considered as preferred decision
- The preferred decision is taken

page 28

Greedy insertion for dTP

•••••



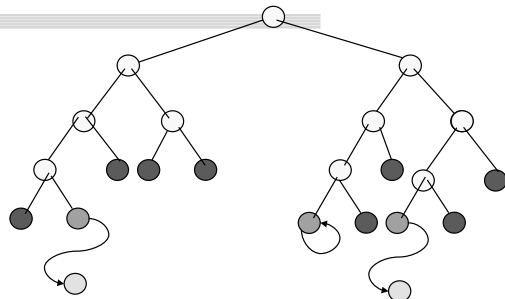
page 29

Discussion

- CP then LS: can be interesting for dTP
 - In particular in case of tight side constraints
- « One-shot » use of CP:
 - as long as no valid solution has been found, we look for one
 - Guarantees that during LS, a valid solution is always available

page 30

A systematic combination



- Solve the problem through CP (global search tree)
- Try to improve each solution found through local search
- Improve the optimization cuts

page 31

Discussion

- Local moves should change the assignment of « early » variables
 - Avoid visiting the same region as with backtracking
- Especially interesting in case of incomplete search
 - CP provides a set of diversified seeds for local search

page 32

Outline

- Introduction
- A didactic optimization problem (dTP)
 - Motivations for cooperation
- A zoo of CP / LS hybrids
 - Sequential combination
 - Master / sub-problem decomposition
 - Improved neighborhood exploration
 - Neighborhoods in CP
 - Large neighborhood search
 - Local moves over a heuristic
 - Local moves during construction

page 33

Master / sub-problem decomposition

- Idea: identify two sub-problems and solve them by different techniques
 - Master problem
 - Induced sub-problem
- Decomposition: the sub-problem can only be stated once the master problem is solved.

page 34

Purpose of decomposition

- Decompose into easier problems
 - Smaller size
 - Simpler models
 - Well known structure
- Traditional approach with exact methods (Dantzig, Lagrangean, Benders)

page 35

A decomposition on dTP

- Master Problem:
 - Assignment of clients to trucks (*visitedBy*)
- Induced sub-problem:
 - Traveling salesman with time windows
- Algorithms:
 - Assess a cost for each client (e.g. distance to neighbor), solve assignment with some method
 - Solve small TSPs with CP
 - Analyze TSPs, re-assess client cost and try improving local moves on the master problem.

page 36

Discussion

- Decomposition makes the problem easier to solve
- Estimating the cost in the master problem may be difficult
- Try local changes on the evaluated cost of the master problem
 - improve subsequent optimization (feedback from the sub-problem)

page 37

Outline

- Introduction
- A didactic optimization problem (dTP)
 - Motivations for cooperation
- A zoo of CP / LS hybrids
 - Sequential combination
 - Master / sub-problem decomposition
 - Improved neighborhood exploration
 - Neighborhoods in CP
 - Large neighborhood search
 - Local moves over a heuristic
 - Local moves during construction

page 38

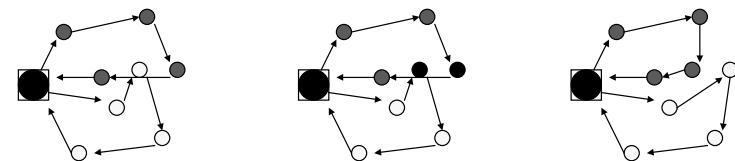
Constrained local search

- Small neighborhoods
 - A neighbor solution S_1 can be reached from a given solution S^* by performing “simple” modifications of S^* .
 - Examples:
 - Choose two visits i_1 and i_2 , remove i_1 from its current position and reinsert it after i_2
 - Choose two visits i_1 and i_2 and exchange their positions

page 39

Constrained local search

- Node Exchange
 - Choose two visits i_1 and i_2 assigned to different trucks and exchange their positions
 - Accept the first exchange improving the cost



page 40

Node Exchange: version 1

```

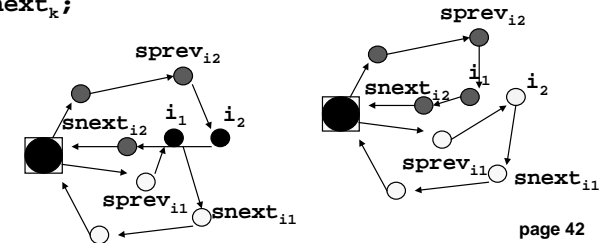
Procedure exchange(P,S)
  forall nodes i1
    forall nodes i2 | (svisitedByi1 ≠ svisitedByi2)
      exchangeInstantiate(P,S,i1,i2)
      // check feasibility and check cost function
      if (propagate(P) && improving(P,S))
        storeSolution(P,S)
        resetProblem(P)
        exit iterations
      // reinitialize the domain variables
      resetProblem(P)
  
```

page 41

Node Exchange: version 1

```

Procedure exchangeInstantiate(P,S,i1,i2)
  // exchange i1 and i2
  next[sprevi1] = i2; next[i2] = snexti1;
  next[sprevi2] = i1; next[i1] = snexti2;
  // restore the rest
  forall k ∉ {i1,i2,sprevi1,sprevi2}
    next[k] = snextk;
  
```



page 42

Node Exchange: version 1

- **Pros:**
 - Independent from side-constraints
- **Cons:**
 - CP imposes monotonic changes
 - while moving from one neighbor to the next one all problem variables are un-instantiated and re-instantiated
 - Constraints are checked in “generate and test”
 - inefficient

page 43

Node Exchange: version 2

Add inlined constraint checks

```

Procedure exchange(P,S)
  forall nodes i1
    forall trucks k | (k ≠ svisitedByi1)
      if (not binCompatible(P,S,svisitedByi1,k)) continue
      forall nodes i2 | (svisitedByi2 = k)
        if (not timeWindowCompatible(P,S,i1,i2)) continue
        if (not improving(P,S,i1,i2)) continue
        exchangeInstantiate(P,S,i1,i2)
        // check feasibility && check cost function
        if (propagate(P) && improving(P,S))
          storeSolution(P,S)
          resetProblem(P)
          exit iterations
        resetProblem(P); // reinitialize the domains
  
```

page 44

Node Exchange: version 2

- Pros:
 - “Almost” independent from side-constraints
 - Some constraints are tested before performing the move
 - much more efficient
- Cons:
 - CP imposes monotonic changes
 - Some constraints are still checked in “generate and test”

page 45

Outline

- Introduction
- A didactic optimization problem (dTP)
 - Motivations for cooperation
- A zoo of CP / LS hybrids
 - Sequential combination
 - Master / sub-problem decomposition
 - Improved neighborhood exploration
 - Neighborhoods in CP
 - Large neighborhood search
 - Local moves over a heuristic
 - Local moves during construction

page 46

Node Exchange: even more CP

CP based neighborhoods

- The neighborhood of a solution S for a problem P is defined by a constraint problem

$$NP(P,S) :: [\{I_1, \dots, I_n\}, \{C_1, \dots, C_m\}]$$

- Each solution of NP represents a neighbor of S for P

page 47

Node Exchange: nhood model

- Variables: $I :: [0..n-1]$, $J :: [0..n-1]$, $DCost :: [-\infty..0]$
 I, J are domain-variables representing the nodes i, j that we want to exchange.

- Constraints:

// neighborhood cst

$I > J$

$svisitedBy[I] \neq svisitedBy[J]$

$next[I] = snext[J]$

$next[J] = snext[I]$

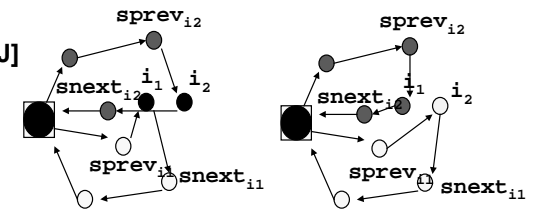
$next[sprev[I]] = J$

$next[sprev[J]] = I$

// interface cst

forall k , ($k \neq I \wedge k \neq J$)

$\Rightarrow next[k] = snext[k]$, $visitedBy[k] = svisitedBy[k]$



page 48

Node Exchange: nhoud model

- DCost represents the gain w.r.t S:

$$\begin{aligned} \text{DCost} = & \text{cost}[\text{sprev}[\text{J}], \text{I}] + \text{cost}[\text{I}, \text{snext}[\text{J}]] + \\ & \text{cost}[\text{sprev}[\text{I}], \text{J}] + \text{cost}[\text{J}, \text{snext}[\text{I}]] - \\ & \text{cost}[\text{sprev}[\text{I}], \text{I}] - \text{cost}[\text{I}, \text{snext}[\text{I}]] - \\ & \text{cost}[\text{sprev}[\text{J}], \text{J}] - \text{cost}[\text{J}, \text{snext}[\text{J}]] \end{aligned}$$

// improving cst

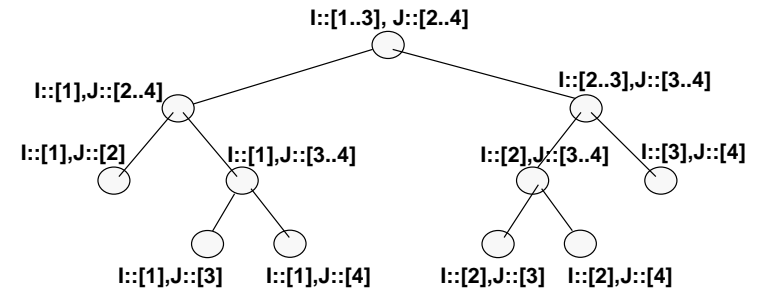
DCost < 0

- Search (explore the neighborhood via tree search):
instantiate(I) && instantiate(J)

page 49

Node Exchange: nhoud model

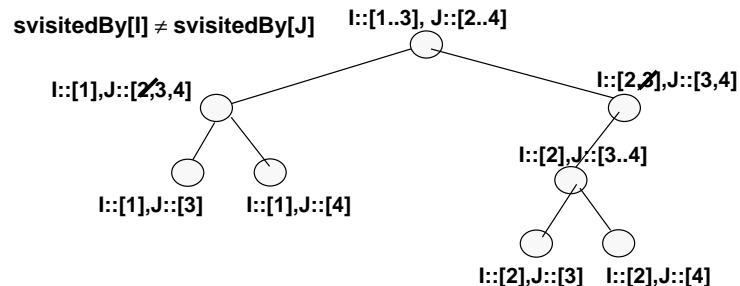
- Search: instantiate(I) && instantiate(J)
- Each leaf defines a feasible exchange



page 50

Node Exchange: nhoud model

- Suppose that in S:
 - clients 1,2 are visited by truck 1,
 - clients 3,4 are visited by truck 2



page 51

Node Exchange: nhoud model

- Pros:
 - Independent from side-constraints
 - Constraint Propagation removes infeasible neighbors a priori.
 - efficient when many side constraints
 - efficient when large neighborhood
 - May freely mix tree search and local search
- Cons:
 - Overhead due to tree search

page 52

Node Exchange: nhoud model

- Overhead due to tree search
 - Often most problem variables are instantiated by the interface constraints only when ALL neighborhood variables are instantiated (at every leaf of the nhoud tree search)
 - In this case the nhoud tree search keeps “doing” and “undoing” the instantiations of ALL the problem variables

page 53

Local Search via solution deltas

- Goal: avoid instantiating and un-instantiating ALL problem variables while moving from one neighbor to the other
 - A neighbor is identified by the modification over the original solution S. This modification is defined *solution delta*.
 - A neighborhood is an array of *deltas*.
 - The exploration of the neighborhood takes place on a tree search.

page 54

LS via solution deltas : Node Exchange

```
Procedure exchange(P,S)
  SolutionArray neighborArray
  forall nodes i1
    forall nodes i2 | (svisitedByi1 ≠ svisitedByi2)
      Solution delta = {(next[sprev[i2]] = i1),
                       (next[i1] = snext[i2]),
                       (next[sprev[i1]] = i2),
                       (next[i2] = snext[i1])}
      neighborArray.add(delta)
  exploreNeighborhood(P,S,neighborArray)
```

page 55

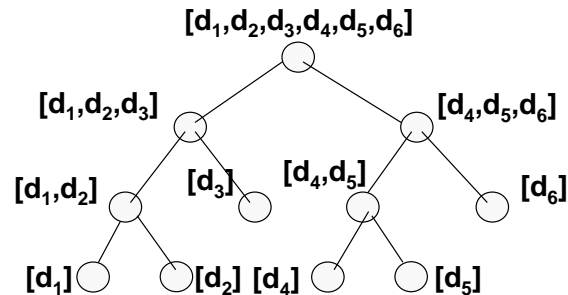
LS via solution deltas: explore the neighborhood

- Map the array of deltas in a tree search
 - recursively split the array of deltas in two parts
 - a split correspond to a branching node in the tree search
 - each feasible neighbor is a leaf of the tree
 - at each node restore the fraction of S that is shared by all neighbors in that node

page 56

Exploring the neighborhood through solution deltas

- Example: $\text{deltas} = [d_1, d_2, d_3, d_4, d_5, d_6]$



page 57

Local Search via solution deltas

- Pros:
 - Independent from side-constraints
 - Constraint Propagation removes infeasible neighbors a priori.
 - efficient when many side constraints
 - efficient when large neighborhood
 - May freely mix tree search and local search
 - Reduced overhead of the tree search
- Cons:
 - Requires an explicit generation of the neighborhood
 - Requires to fully specify each move

page 58

Outline

- Introduction
- A didactic optimization problem (dTP)
 - Motivations for cooperation
- A zoo of CP / LS hybrids
 - Sequential combination
 - Master / sub-problem decomposition
 - Improved neighborhood exploration
 - Neighborhoods in CP
 - Large neighborhood search
 - Local moves over a heuristic
 - Local moves during construction

page 59

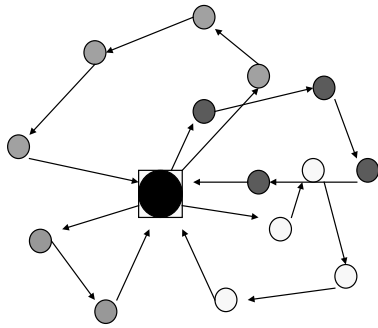
Large neighborhood search

- Idea: partition the variables of the current solution into two subsets
 - A fragment: assignments are kept as they are
 - A shuffle set: assignments may be changed

page 60

Large neighborhood search on dTP

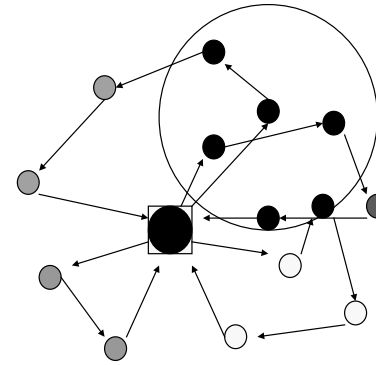
- From a solution



page 61

Large neighborhood search on dTP

- Select a shuffle set

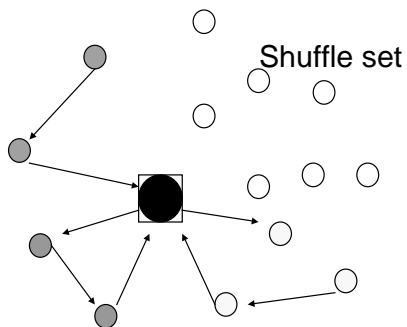


Select a subset of clients
 $i_1, i_2, \dots, i_k \subset C$

page 62

Large neighborhood search on dTP

- Example on dTP



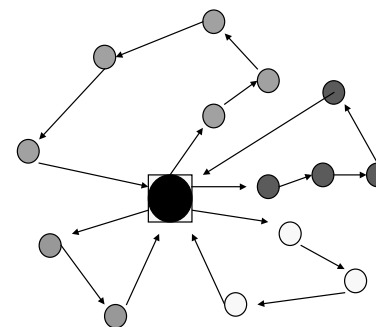
For all clients i from the shuffle set :

- Unassign variable:
 - $visitedBy_i$, $collectedIn_i$
 - $start_i$
- Undo all ordering decisions between i and other clients j
- Unassign all cost variables
- Post a cost improvement cut
 $cost \leq getValue(cost, S)$

page 63

Large neighborhood search on dTP

- Look for a new solution by solving the remaining sub-problem



page 64

Large neighborhood search

● Exploring the neighborhood

```
Procedure moveLNS(P,S,algorithm)
  // define current sub-problem
  Problem P' = P  $\wedge$  (cost  $\leq$  getValue(cost,S) -  $\epsilon$ )
  propagate(P')
  while (not stop())
    VariableSet shuffleSet = defineShuffleSet(P,S)
    foreach variable X | X  $\in$  shuffleSet
      | P' = P'  $\wedge$  (X = getValue(X,S))
      | if solve(P',algorithm,S) succeeds
      | stop iteration
```

page 65

Selecting shuffle sets

- Select a set:
 - large enough to introduce enough flexibility
 - small enough to reduce the overall problem
 - of inter-dependent variables
 - of ill-assigned variables (an improvement can be expected)
- Vary the types of sets that are shuffled
- Vary the size of sets that are shuffled
 - variable neighborhood search

page 66

Shuffle sets for dTP

- A set of clients that are
 - Within short distance of some specific client
 - Visited by the same truck
 - Sharing a common type of goods
 - Visited within a common time frame
 - ...

page 67

A few hints for LNS with CP

- Use incomplete tree search to speedup the sub-problem solution (e.g. LDS)
- Use strong constraint propagation to reduce the neighborhood exploration
- Compute relaxations to prune non-improving neighbors
- Rather switch neighborhood than fully explore one by backtracking

page 68

Outline

- Introduction
- A didactic optimization problem (dTP)
 - Motivations for cooperation
- A zoo of CP / LS hybrids
 - Sequential combination
 - Master / sub-problem decomposition
 - Improved neighborhood exploration
 - Neighborhoods in CP
 - Large neighborhood search
 - Local moves over a heuristic
 - Local moves during construction

page 69

Local moves over a heuristic

- LS is defined as variations over a solution.
- LS can also be applied over an encoding of a solution
 - For a greedy CP method, the search heuristic itself is an encoding
 - Idea: instead of exploring the whole tree, explore variations of the constructive heuristic.

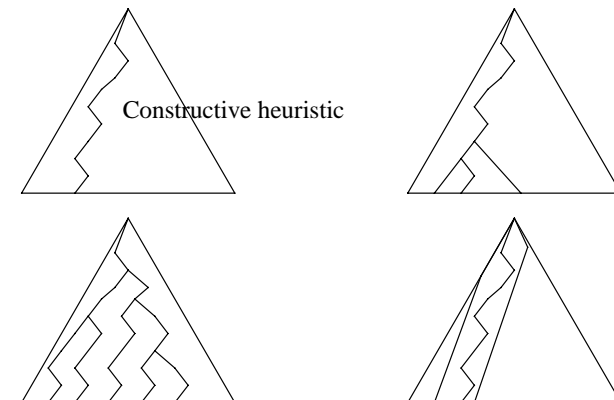
page 70

Local search over a heuristic

- Two family of methods
 - Local moves over a value ordering heuristic
 - Restricted candidate lists
 - GRASP
 - Discrepancy based search
 - Local moves over a variable ordering heuristic
 - List scheduling heuristics
 - Preference-based programming

page 71

Local search over the value selection heuristic



page 72

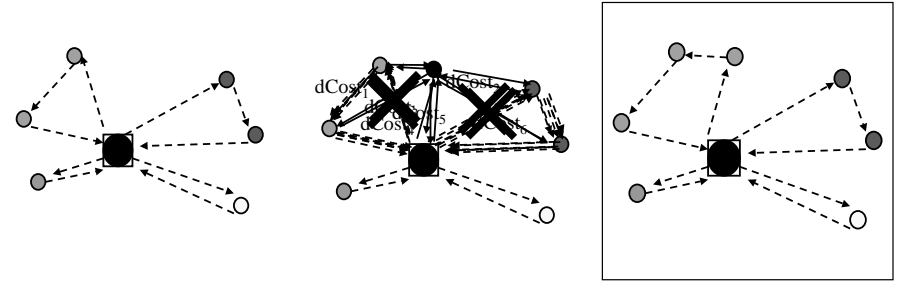
Restricted candidate list

- At each choice point a function h is evaluated for all possible choices:
 - the k “worst” choices (with high value for h) are discarded
 - the choice that minimizes h is considered as preferred decision
 - the preferred decision is taken, the remaining choices are taken upon backtracking

page 73

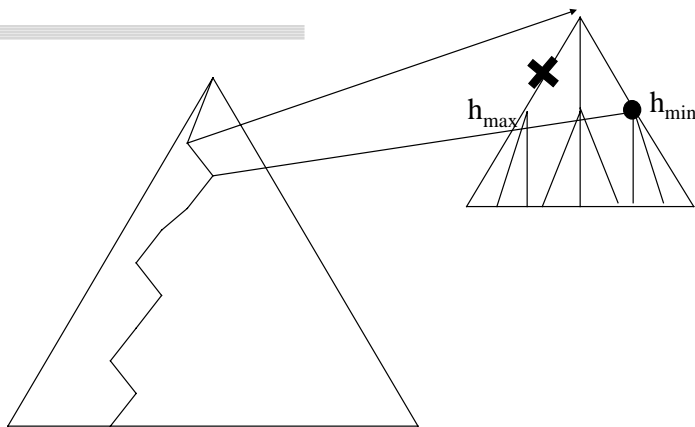
Restricted candidate list

• • • • •



page 74

Restricted candidate list



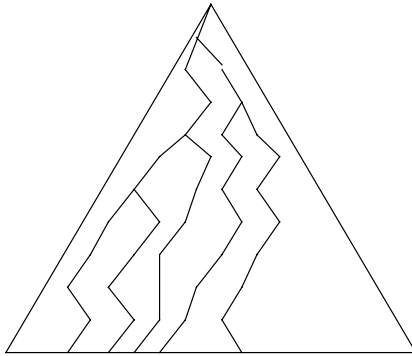
page 75

GRASP

- “Greedy Randomized Adaptive Search Procedure”
- At each choice point a function h is evaluated for all possible choices:
 - the preferred decision is chosen by a random function biased towards choices having small value for h
 - the preferred decision is taken
 - the process is iterated until a stopping condition is met

page 76

GRASP



page 77

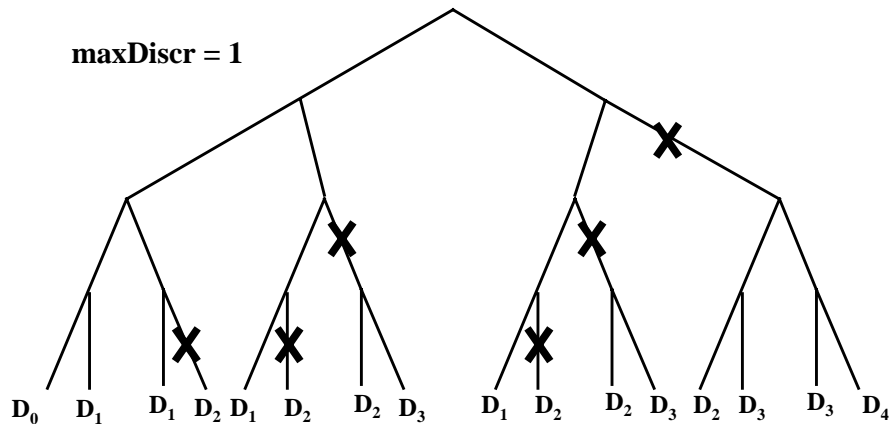
Discrepancy based search

- Idea: good solutions are more likely to be constructed by following always but a few times the heuristic
 - during search, count the number of times the heuristic is not followed (number of *discrepancies*)
 - a maximal number of discrepancies is allowed when generating solutions in the tree.

page 78

Discrepancy based search

maxDiscr = 1



page 79

CP + some LS: putting things together

- Example:

```
Procedure solve(P)
  while (not stopping condition)
    Solution S = ∅
    int failLimit = 50
    bool result = solveGRASP(P,S,failLimit)
    if (result)
      P = (P ∧ (Cost(P) < Cost(S)))
```

page 80

CP + some LS: putting things together

- **Example:**

```
Procedure solveGRASP(P,S, failLimit)
  while (unscheduled clients exist
        and failLimit not reached)
    Client v = selectVariable(P,S)
    discardBadValues(P,S) //Restricted Candidate List
    InsertionPosition position = evaluateRandom(P,S,v)
    try (insert(P,S,v, position) OR
        notInsert(P,S,v, position))
```

page 81

Local search over the variable selection heuristic

- In some problems, a solution can be described by a variable ordering
 - Natural value ordering heuristics
- **Examples:**
 - List-scheduling heuristics
 - Configuration problems
- Local moves can be applied on the variable sequence itself

page 82

Local moves on a heuristic

- **Standard process in Genetic Algorithms:**
 - Encode the solution
 - Apply local changes to the encoding
 - Construct the new solution (can be done by a CP-based solver)
- **In CP: Preference-based programming**

page 83

Preference-based programming

- **Example on Job-Shop scheduling:**
 - Consider a ordered list of tasks (priority list)
 - Choice point: (Schedule asap OR Postpone)
 - Take one task at a time from the list and schedule it at its earliest start time
 - otherwise “postpone” the decision on the task for later
 - Local moves on the preferred list of tasks generate different schedules
 - Use tree search to explore a neighborhood of the preferred list

page 84

Outline

- Introduction
- A didactic optimization problem (dTP)
 - Motivations for cooperation
- A zoo of CP / LS hybrids
 - Sequential combination
 - Master / sub-problem decomposition
 - Improved neighborhood exploration
 - Neighborhoods in CP
 - Large neighborhood search
 - Local moves over a heuristic
 - Local moves during construction

page 85

Local Search and Greedy Construction

- Local search is most often applied to complete solutions
- First build a solution, then improve it
- Idea: better repair while building than afterwards.
=> Incremental Local Optimization

page 86

Incremental Local Optimization

- The greedy algorithm makes a mistake at step n
- The mistake is discovered at step $n+k$
- Try to repair the steps $n .. n+k$
- Resume the greedy construction at step $n+k+1$

page 87

ILO for general CSPs

A simple incomplete method:

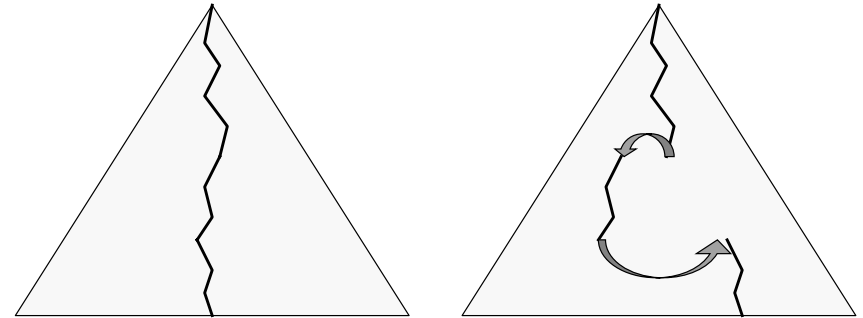
- For a variable ordering $v_1 \dots v_n$
- Compute a lower bound lb
- Start assigning variables
- Choose the value a_{ik} such that $v_i = a_{ik}$ yields the least increase in lb
- Whenever lb strictly increases,
 - keep $v_i = a_{ik}$,
 - un-assign all variables linked to v_i and
 - try to re-assign them to find the least increase for lb

page 88

ILO illustrated on dTP

- Enriched greedy construction scheme:
 - Place clients on a stack
 - Insert them one by one minimizing the insertion cost.
 - For client i , instantiate
 - $visitedBy_i$
 - $succ_i$
 - Apply local optimization on the truck assigned to i
 - Change the order of visits j (forall $j \mid visitedBy_j = visitedBy_i$)
 - If an improving sub-route is found, change it

Illustration on a search space

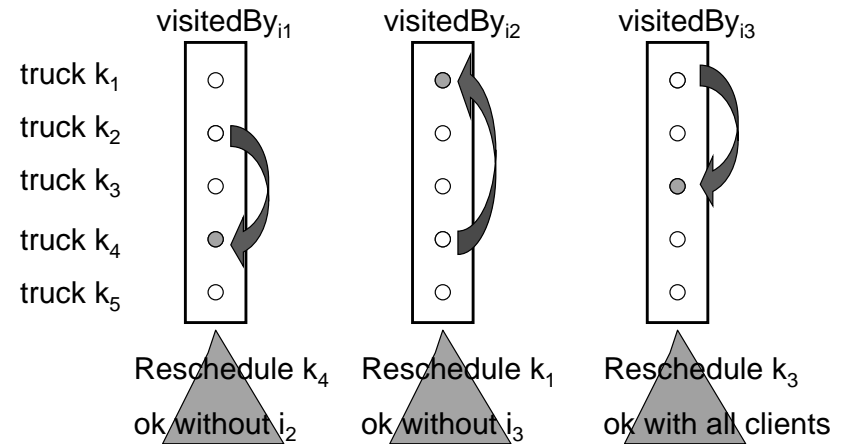


Related to conflict-directed search

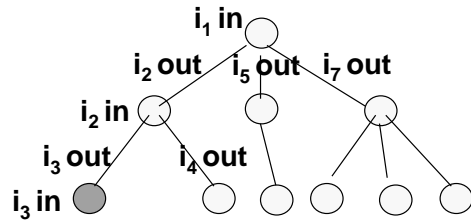
Ejection chains during a greedy process

- Recursive version of the ILO approach
 - For a variable ordering $v_1 \dots v_n$
 - Choose the value a_{ik} such that $v_i = a_{ik}$ yields the least increase Δlb in lb
 - When $\Delta lb > 0$, un-assign some variable v_i so that lb decreases
 - Reassign v_i to some other value
 - Go-on un-assigning / re-assigning past variables until the least increase in lb is found

Ejection chains on dTP



Finding a good ejection chain



- Search for the smallest ejection chain in breadth first search
- Similar to the search for augmenting paths (flows)

page 93

Conclusion

Real life combinatorial optimization problems often require crafting hybrid optimization methods:

- local search is a technique that can complement CP
- many hybrids are possible

« Is it cookery or alchemy ? » M. Wallace

Recipes and tools are emerging ...

page 94

page 95

page 96