Local Search Techniques for Scheduling Problems — A Tutorial —

Andrea Schaerf

Dipartimento di Informatica e Sistemistica
Università di Roma "La Sapienza"
Via Salaria 113, 00198 Roma, Italy

13^{th} European Conference on Artificial Intelligence
Brighton, UK, August 23–28, 1998

Outline of the Tutorial

1. Introduction

2. Local Search Framework

3. Basic Techniques [Case Study No. 1]

BREAK

4. General Improvements [Case Study No. 2]

BREAK

5. Hybrid Techniques

6. Additional Issues, Discussion and Conclusions

Scheduling Problems

$$\boxed{\text{Assign resources to tasks over time}}$$

Examples:

Theoretical problems: job-shop scheduling, traveling salesman, multiprocessor scheduling, ...

Practical problems: course timetabling, crew scheduling, frequency assignment, resource allocation, ...

Common Features of Scheduling Problems

- Combinatorial search or optimization problems

- Involve various types of constraints
  (precedence, overlap, availability, capacity, ...)

- NP-complete

$\rightarrow$ Hard in practical cases

Search Techniques

Constructive Methods:

- Backtracking-based (exhaustive)
  - Forward Checking, Backjumping, ...
  - Branch & Bound, ...
- Backtracking-free
  - Greedy
  - Heuristic repair

Selective Methods:

→ Local Search
  - Hill-Climbing
  - Simulated Annealing
  - Tabu Search
- Evolutionary Methods
  - Genetic Algorithms
  - Memetic Algorithms

Integer Programming:

- Relaxation + Rounding
- ...

Main Features

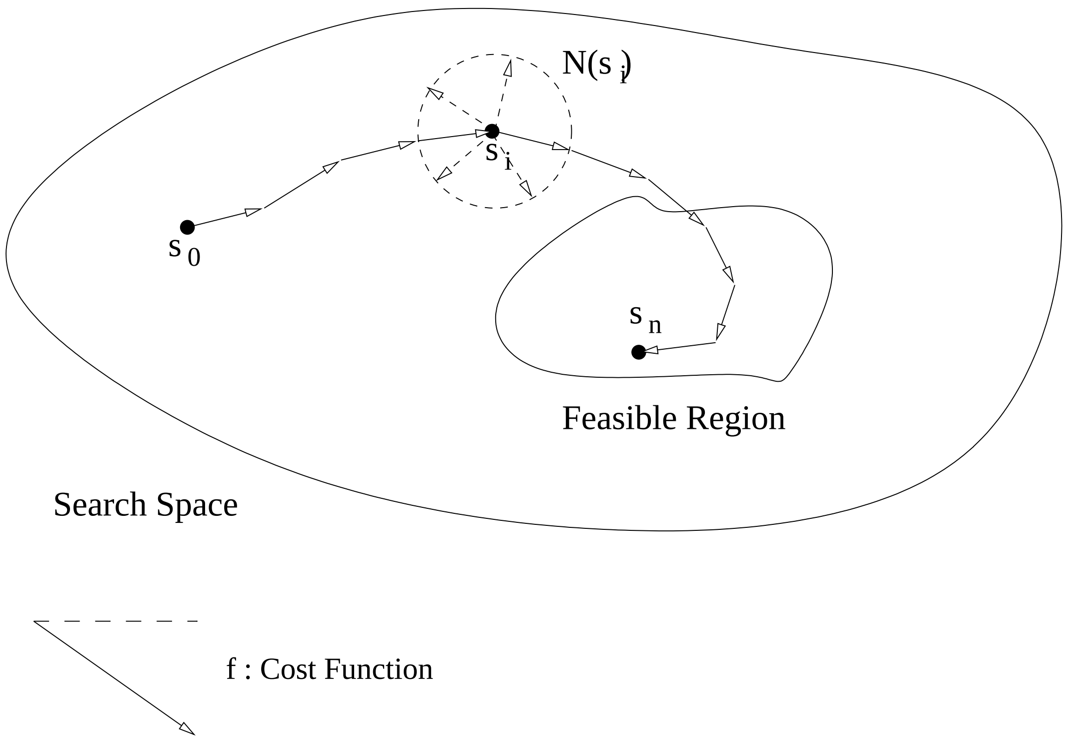
Constructive Techniques

- More natural: better understanding of the automatic search
- Fast for easy cases
- Better control on the critical steps

Selective Techniques

- Proved to be effective in many real applications
- Provide approximate solutions
- Revise previous solutions

Local Search



Main Features

Technique Independent:

- Search Space
- Neighborhood Relation
- Cost Function
- ○ Initial Solution Selection

Technique Specific:

- Move Selection
- Stop (and Restart) Criteria

Search Space

- Contains feasible (optimal) solutions
- Possibly small
- Easy to provide a random element

Example: n-queens problem

- Permutations of $[1..n]$
- Assignments $[1..n] \longrightarrow [1..n]$

Neighborhood Relation

- Search space connected under it
- Easy to compute (and to store)
- Fast to evaluate the cost difference

$\rightarrow$ Effective

Example: n-queens problem

- **Swap** neighborhood (for Permutations)
  $[4, \mathbf{3}, 6, 5, 2, \mathbf{1}, 8, 7] \quad \rightarrow \quad [4, \mathbf{1}, 6, 5, 2, \mathbf{3}, 8, 7]$
- **Replace** neighborhood (for Assignments)
  $[3, 4, 1, 7, 8, \mathbf{4}, 2, 1] \quad \rightarrow \quad [3, 4, 1, 7, 8, \mathbf{6}, 2, 1]$

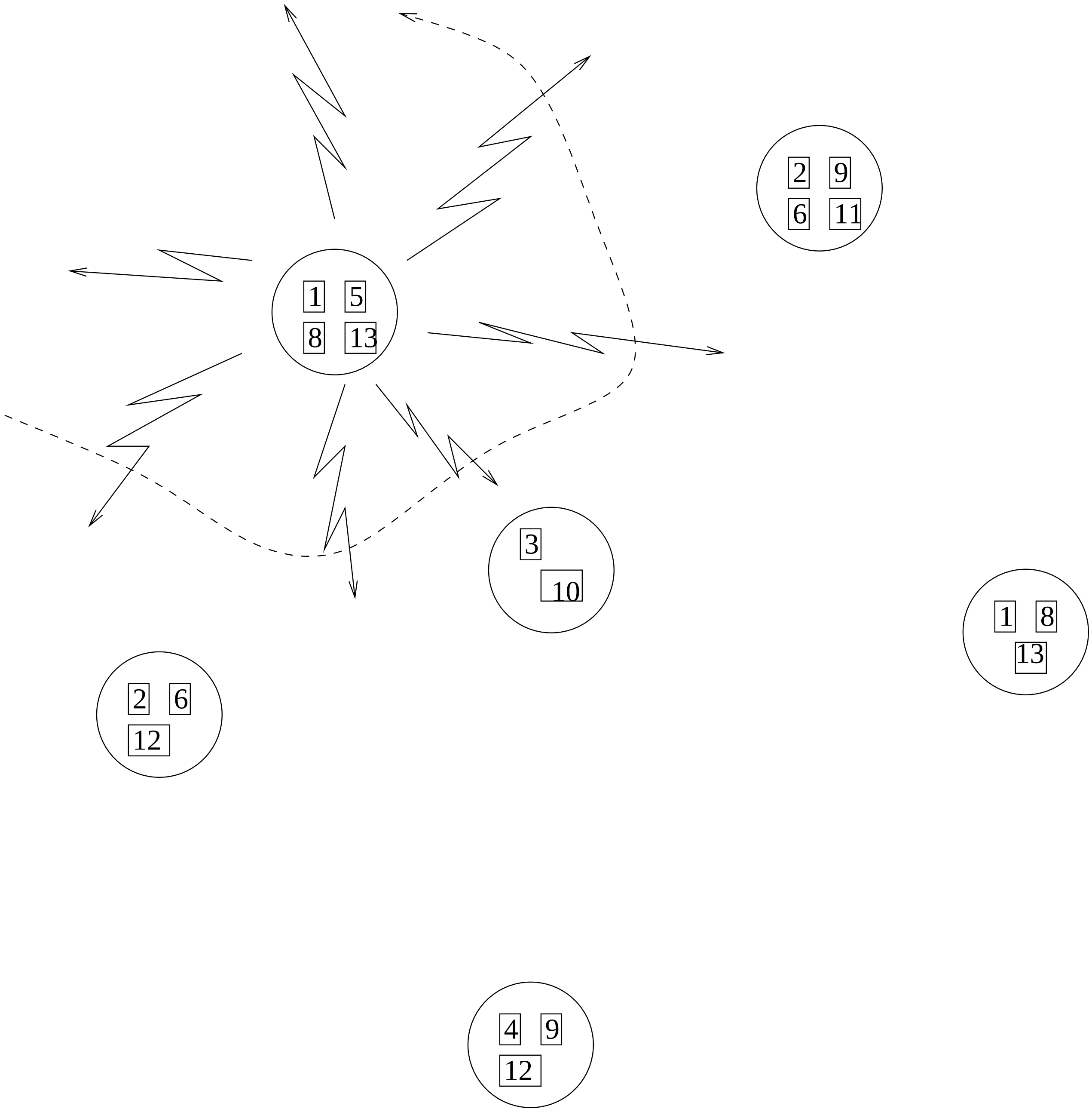
Cost Function

- Embeds distance to feasibility
- Embeds objective function
  - Accounts for invisible improvements
  - Allows diversification

Initial Solution Selection

- Random
- Constructed with a greedy method
- Solution of a previous instance

Case Study No. 1: Frequency Assignment



Frequency Assignment: Definition

Given:

- n *cells*, c_1, c_2, ..., c_n
- m frequencies (or channels) 1, 2, ..., m
- a *request vector* R_n,
- a *separation matrix* $S_{n \times n}$
- a *preassignment matrix* $P_{n \times m}$ $(\{-1, 0, 1\})$

Find for each cell c_i:

- a set of frequencies $F_i = \{f_{i1}, \ldots, f_{ir_i}\}$.

Such that:

- $\forall\ i, j, h, k : 1 \leq i \leq j \leq n$,
  $1 \leq h \leq r_i,\ 1 \leq k \leq r_j$

$$| f_{ih} - f_{jk} | \leq S_{ij}$$

- $\forall\ i, j :\ 1 \leq i \leq n, 1 \leq j \leq m$

$$\begin{array}{ll} \text{if} \quad P_{ij} = -1 & \text{then} \ \ j \notin F_i \\ \text{if} \quad P_{ij} = 1 & \text{then} \ \ j \in F_i \end{array}$$

Search Space for Frequency Assignment

Assignment matrix: $X_{n \times m}$

$$x_{i \times j} : \quad \begin{cases} 0 & \text{if } \; j \in F_i \\ 1 & \text{if } \; j \notin F_i \end{cases}$$

Such that:

- Requests are satisfied: $\Sigma_{j=1}^{m} x_{ij} = r_i$
- Preassignments and satisfied:

$$\begin{array}{lll} \text{if} \quad P_{ij} = -1 & \text{then} & x_{ij} = 0 \\ \text{if} \quad P_{ij} = 1 & \text{then} & x_{ij} = 1 \end{array}$$

Note: It is easy to generate a random assignment!

Neighborhood Relation for Frequency Assignment

- Defined by a triple: $\langle i, j, k \rangle$
- $j \in F_i$ and $k \notin F_i$
- $P_{ij} \neq 1$ and $P_{ik} \neq 0$

Example:

$$\begin{array}{c ccccccc}
 & & & j & & & k & \\ \cline{2-8}
 & 1 & 0 & 0 & 0 & 0 & 1 & 0 \\
 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\
i & 0 & 1 & 1 & 0 & 0 & 0 & 1 \\
 & 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ \cline{2-8}
\end{array}$$

Cost Function for Frequency Assignment

$\boxed{\text{Search problem} \longrightarrow \text{No objective function}}$

Sum of separation violations

- Intra-cell
- Inter-cell

Example:

$$X = \begin{array}{ccccccc} 1 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 0 & 1 & 0 & 0 & 0 \end{array} \qquad S = \begin{array}{cccc} 2 & 1 & 0 & 1 \\ 1 & 2 & 0 & 1 \\ 0 & 0 & 3 & 1 \\ 1 & 1 & 1 & 2 \end{array}$$

Basic Local Search Techniques

- Hill Climbing
- Simulated Annealing
- Tabu Search
- ~~Genetic Algorithms~~

Hill Climbing

"make only improving (and sideways) moves"

- random descent, first descent, best descent

$\rightarrow$ MCHC [Minton et al. 90]

$\rightarrow$ GSAT [Selman et al. 92]

Main Parameters:

- `IDLE_ITERATIONS`

Min-Conflict Hill Climbing

Solve CSPs with binary constraints

variables: $\{x_1, x_2, \ldots, x_n\}$

values: $v(x_i) \in D_i$ (typically $D_i = \{1, \ldots, m\}$)

constraints: $x_i \leq x_j$

Algorithm

- Select a variable that is in *conflict*
- Assign to it a value that minimize the number of conflicts (break ties randomly)

Move: $\{v_1, \ldots, v_i, \ldots, v_n\} \longrightarrow \{v_1, \ldots, v_i', \ldots, v_n\}$

Cost function: "Minimize conflicts"

GSAT

Solve satisfiability of CNF propositional formulas

letters: $\{p_1, p_2, \ldots, p_n\}$

assignment: $A(p_i) \in \{true, false\}$

clauses: $p_i \vee p_j \vee \neg p_k$

Algorithm

- Select the letter that falsifies the max number of clauses (break ties randomly)
- Change (flip) its truth value

Move: $\{a_1, \ldots, a_i, \ldots, a_n\} \longrightarrow \{a_1, \ldots, \bar{a}_i, \ldots, a_n\}$

Cost function: "Maximize Satisfied Clauses"

Simulated Annealing

"make worsening moves with
probability decreasing with time"

Algorithm: Pick a random move m:

- if $\Delta \leq 0$ always accept m
  if $\Delta > 0$ accept m with probability $e^{-\Delta/T}$
- if (max_iterations_per_temperature) decrease T

Main Parameters of Simulated Annealing

- `TEMP_FACTOR` : cooling rate
- `SIZE_FACTOR` : no. of iterations at each temperature w.r.t. size of neighborhood
- `INIT_PROB` : no. of accepted moves at the starting temperature
- `MIN_PERCENT` : no. of accepted moves at the stop temperature

Main Variants of Simulated Annealing

- Cooling schemes [Aarts et al. 97]
  - Geometric
  - Polynomial: new temperature based on standard deviation
  - Efficient: based on no. of accepted moves per temperature

- Deterministic Acceptance [Moscato & Fontanari 90]
  - Threshold Accepting:

    if $\Delta \geq 0$, accept only if $\Delta \leq T$

    if $\Delta < 0$, accept only if $\Delta < -T$

Tabu Search

"make the best move (even if worsening)"

Algorithm: Find the "best" move m which is not in the Tabu List:

- accept m and include it in the Tabu List;
- remove from the Tabu List the oldest move;

Main Parameters of Tabu Search

- `TABU_LIST_LENGTH`: number of tabu iterations for a move
- `IDLE_ITERATION`: number of iterations since the last improvement (stop criterion)
- `NEIGHBORHOOD_SHARE`: number of examined neighbors at each iteration

Tabu Status

- Tabu Solutions
- Tabu Moves
- Tabu-Active Attributes
- Mixed

Tabu Tenure

- Fixed
- Systematic: Tabu Patterns
- Variable-size Tabu List

→ Random Tabu Tenure

Aspiration Criteria (by move or by attribute)

- Default: if all moves are tabu drop the tabu status of some of them
- Improve Best:
  - global
  - regional
- Influence Criteria: drop the tabu status of a low influence move after a high influence move

Frequency Assignment by Local Search

- Simulated Annealing [Duque-Antòn 93] (Mixed Cooling Scheme, Extended Neighborhood)
- Tabu Search [Castelino et al. 96] (Frequency-based Tabu)
- Comparison [Costa 93]: Tabu Search Wins!

General Considerations

- The three methods correspond to 3 different philosophies (simple, randomized, memory-based)
- Different techniques might work differently for different neighborhood relations
- It is important to compute Δf efficiently

$\boxed{\text{BREAK!}}$

Improvements: Guiding Principles

- Improving efficiency
- Diversification
- Intensification

→ Learning while searching

Improvements: General Strategies

- Modeling the neighborhood relation
- Modeling the cost function
- Improving the tabu mechanism
- *Iterated* local search
- Combining local search techniques

Modeling the Neighborhood Relation

Neighborhood Sampling

- Random sampling
- Hierarchical cost function
- *Plus* strategy

→ Candidate list strategies

Interleaving different neighborhood relations

- Alternate neighborhoods

→ Composed neighborhood

Candidate List Strategies

- Elite Candidates
- Activation bits ("fast local search" [Tsang, 95])
- Filtering based on independent attributes

Composed Neighborhood

- Two neighborhood relation:
  e.g. *insert* and *swap*

- An auxiliary penalty weight related to the type of move:

$$\Delta f'(m) = \begin{cases} \alpha_i \cdot \Delta f(m) & m \text{ is an insert} \\ \alpha_s \cdot \Delta f(m) & m \text{ is an swap} \end{cases}$$

Improving the Tabu Mechanism

- Long term memory and frequency-based tabu status
  - transition frequency
  - residence frequency

  ...

→ Reactive Local Search

Reactive Local Search [Battiti & Tecchioli 96]

Tabu list length is adjusted reactively:

- *long* when close to a local minimum
- *short* when far from all of them

A solution is revisited often $\Rightarrow$ length increased
No revisited for a while $\Rightarrow$ length decreased.

Note: expensive in terms of space, needs hashing mechanisms

Modeling the Cost Function (Shifting Penalty)

- Auxiliary components of the cost function related to local minima (*Guided Local Search*)

- "Learning" the weight of elements of the cost function

→ Adaptive weight of different components

Adaptive Relaxation of Hard Constraints

for each iteration

  for each type of infeasibility i:

    if for k consecutive iterations:

      $\#Violations(i) = 0$

    then

      $\alpha_i := \alpha_i/2$

    else if for k consecutive iterations:

      $\#Violations(i) > 0$

    then

      $\alpha_i := \alpha_i \cdot 2$

    else

      α_i is unchanged

Iterated Local Search

When stopped make a “large step” and restart

- Random walk (WSAT)
- Problem-dependent large step

Combining local search techniques

- Combining features of different techniques:
  e.g. simulated annealing with tabu list

$\rightarrow$ Interleaving different techniques

Case Study No. 2: High School Timetabling

Find the weekly schedule of all lectures of a high school, given:

- Lectures' requirement matrix

  {teacher `smith` teaches 4 periods to class `3A`}

- Teacher and class unavailabilities

  {teacher `jones` is off on Monday}

  {class `3A` is off last period of Friday}

- Other hard and soft constraints

  {classes `1A` and `1B` do gymnastic together}

NP complete [Even et al. 76]

Genuinely hard to solve in practical cases [Abramson 91, Costa 94, Yoshikawa et al. 94]

Example

Requirements	A1	A2
smith	3	2
jones	0	5
brown	6	1

Availability	Mo	Tu	We	Th	Fr
smith	— —	— —	— —	X X	— —
jones	— —	X X	— —	— —	— —
brown	— —	X —	X —	— —	— —
A1	— —	— —	— —	— —	— X
A2	X —	— —	— —	— —	— —

Timetable	Mo	Tu	We	Th	Fr
smith	A1 A2	A1 A2	A1 —	— —	— —
jones	— —	— —	A2 A2	A2 A2	A2 —
brown	— A1	— A1	— A1	A1 A1	A1 —

Solution Space

$$
\begin{array}{lrl}
 & find \qquad x_{ijk} & (i = 1..m; j = 1..n; k = 1..p) \\
s.t. & \sum_{k=1}^{p} x_{ijk} = r_{ij} & (i = 1..m; j = 1..n) \\
 & \sum_{i=1}^{m} x_{ijk} \leq t_{jk} & (j = 1..n; k = 1..p) \\
 & \sum_{j=1}^{n} x_{ijk} \leq c_{ik} & (i = 1..m; k = 1..p) \\
 & x_{ijk} = 0 \text{ or } 1 & (i = 1..m; j = 1..n; k = 1..p)
\end{array}
$$

$R_{c \times t}$: requirement matrix
$A_{t \times p}, C_{c \times p}$: unavailability matrices
$X_{c \times t \times p}$: timetable $\rightarrow x_{ijk} = 1$ if teacher j teaches to class i at period k

Other Hard Constraints

- Class covering (all periods but last)
- Simultaneous lectures (e.g. gymnastic)
- Bilingual classes
- Other teacher's assignments (e.g. parent meeting, at disposal,...)

Objective Function (Soft Constraints): Some Components

Comp(weight)	Teacher's Daily Assignment				
Holes(1)	1B	—	2B	2B	2A
Too few(3)	—	—	2B	—	—
Splits(6)	2B	2A	1B	2B	—
Clusters(6)	—	2B	2B	2B	2B

Representation of the Problem

```
---------------------------------------------------------
teacher     |       Monday        |       Tuesday       | ..
---------------------------------------------------------
parenti     |       3A 3A <> 2A |                <> 2A | ..
scocchera   | 1A 1A 4A 5A 5A XX | 1A 4A 4A <>    XX | ..
mariani     | 2B 2B 2B <> 1B    | 2B 3B             | ..
marcattili  | 1B 1B 4B 4B       | <> 1B ** 4B 5B    | ..
giacchetti  | 1C <> 3C 2C       |       <> 3C 3C 2C | ..
mori        | 5C 4C 4C <> 1C    |          5C 1C 1C | ..
saconis     | 1D <> <> 2D 2D    |             2D 2D | ..
gabriele    | XX XX XX XX XX XX | 2E 2E 3D 3D 5D    | ..
giagnoro    | 3A 2A             |    5A 3A 2A 1A 4A | ..
greco       | XX XX XX XX XX XX |    2C 3C 2B 1B 3B | ..
ranieri     |    2d 2D 1D 3D 2e |          1D 3D    | ..
spadoni     | XX XX XX XX XX XX |             3A 4a | ..
di_pofi     |          2B 4B    |          1B 3B 5B | ..
meloni      | <> <> 5C 1C 4C    | XX XX XX XX XX XX | ..
valle       | XX 2D 2d 3D 3D 2E | XX XX XX XX XX XX | ..
---------------------------------------------------------
```

Algorithm

Interleave

- Tabu Search {atomic moves}
- Hill Climbing {double moves}

Search Features

$\boxed{\text{Hill Climbing}}$ (fast)

- Random double move
  {Long moves might find improvements that even smart algorithms with short moves can overlook}
- Accept sideways moves
  {Navigate plateaux}

$\boxed{\text{Tabu Search}}$ (slow)

- Adaptive relaxation
  {Weight of hard constraints varies depending on number of feasible solutions visited}
- Variable-size tabu list and double tabu status
  {Randomize permanence in tabu list}
  {Tabu moves and tabu positions}

Advantages of Interleaving

- Hill Climbing finds the first solution
- Hill Climbing “shakes up” the solution when Tabu Search fails
- Tabu Search works only when necessary

Experimental Setting

Three case studies (Italian high schools)

1. 13 teachers, 96 lectures, 3803–46000 neighbors
2. 44 teachers, 610 lectures, 18288–304600 neighbors
3. 61 teachers, 1102 lectures, 25375–388500 neighbors

Hardware and Software

- SUN machine `sparc10`
- Language C++
- CPU time (not including parameter setting):

  School 1: 5 min

  School 2: 1 hour 40 min

  School 3: 4 hours 30 min

Outcomes of the Experimental Results

- Automatic solutions outperform manual ones

- Manual adjustments are sometimes needed

- Interaction is absolutely necessary
  - Tabu Search is especially suitable for interactive run (it makes as few changes as necessary)

$\boxed{\text{BREAK!}}$

Combining Local Search With Other Techniques

- Local search + Genetic Algorithms
- Local search + Constructive Heuristics
- Repeated Local Search
  (“Matrioshka instances”)

Main Ingredients of Genetic Algorithms

- Selection
- Recombination
- Mutation and/or Filtering

Memetic Algorithms

$\boxed{\text{Apply local search before selection}}$

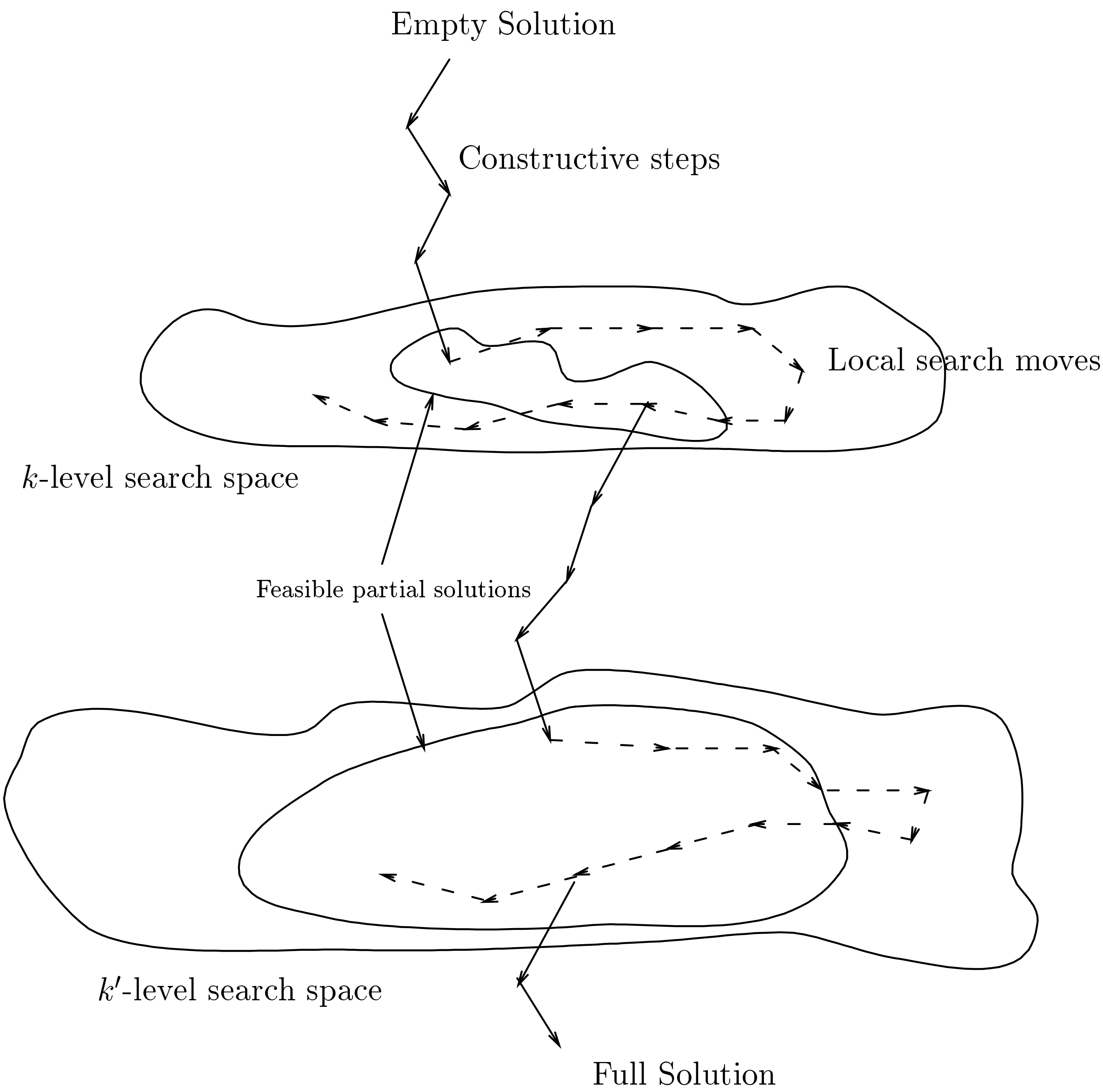
- Local search must be fast
- Different Strategies can be applied to different elements
- Local Search can work as Filtering

Local Search + Constructive Search

- Build + Repair
- Iterated (Build + Repair)
- Local Search + Exhaustive Completion
  [Zhang & Zhang 96]
- Local Search + (Destruction & Construction)
  [Glover et al. 96]

→ Local Search on Partial Solutions

Heuristic Backtracking-Free Construction + Local Search



```
Procedure Solve(ProblemVars)
begin
    current := ∅; rest := ProblemVars;
    while rest ≠ ∅
        x = SelectVariable(rest);
        v = AssignValue(x);
        if v ≠ nil then
            current := current ∪ {x ← v};
            rest := rest \ {x};
        else
            if LastTrial then return ∅
            best := current;
            repeat
                move := SelectMove(current,rest);
                current := MakeMove(current,move);
                if Improves(current,best)
                then best := current
            until LastIteration or LowerBoundReached
            current := best
        endelse
    endwhile;
    return current
end;
```

Basic Components

Construction

- Heuristic Variable Ordering:
  $\longrightarrow$ SelectVariable
- Heuristic Value Ordering:
  $\longrightarrow$ AssignValue

Revision of the Partial Solution

- Local Search Technique:
  $\longrightarrow$ SelectMove
- Cost Function:
  $\longrightarrow$ SelectMove and Improves

Cost Function in Local Search

Adaptively-weighted sum of:

1. Distance to Feasibility
2. Lower Bound of the Objective Function
3. Look-Ahead Factor

$$\boxed{CF = \alpha \cdot \#Violations + \beta \cdot LB(o.f.) + \gamma \cdot F_{lookahaed}}$$

Look-Ahead Factor

Min-Conflict (MC) : Number of available values for all uninstantiated variables [Frost & Dechter, IJCAI-95]

Short-term Min-Conflict (SMC) : Number of available values for next uninstantiated variables

Look-Ahead Scheme:

1. First dead-end at level k $\longrightarrow$ use MC
2. All subsequent ones at level k $\longrightarrow$ use SMC

Results for Course Timetabling (1/2): General results

	No. of instances solved	
Search Method	Problem 1 130 Lectures	Problem 2 580 Lectures
Pure local search methods		
Random HC	46.3%	20%
Steepest HC	15.6%	3%
Min-Conflict HC	59.0%	45%
Combined methods		
Construction + RHC	88.5%	78%
Construction + SHC	87.0%	21%
Construction + MCHC	97.4%	89%
Pure constructive methods		
Backtracking	0%	0%
Forward checking	0%	0%

Repeated Local Search (Used for Euclidean TSP)

1. Create an initial partial problem
2. Run local search
3. Insert a new element
4. Return to step 2

Additional Issues

- Theoretical issues
  - PLS-completeness
  - Convergence results
- Implementation issues
  - $\rightarrow$ Parallel local search
  - $\rightarrow$ Local search software tools

PLS Problems [Yannakakis 95]

There exist three polynomial algorithms for:

- Generating one solution
- Computing the cost function
- Generate a neighbor with $\Delta \leq 0$ if it exists

Classification of Parallel Local Search Approaches [Verhoeven and Aarts, 95]

- Multiple walks
  - Independent vs. Interacting
    Exchange information on good solutions, good parts, ...
  - Synchronous vs. Asynchronous

- Single walk
  - Single-step
  - Multiple-step

Discussion of Parallel Local Search

- Little has been done
- Multiple walks approaches are more promising
- Single walk approaches work only for best-fit strategies

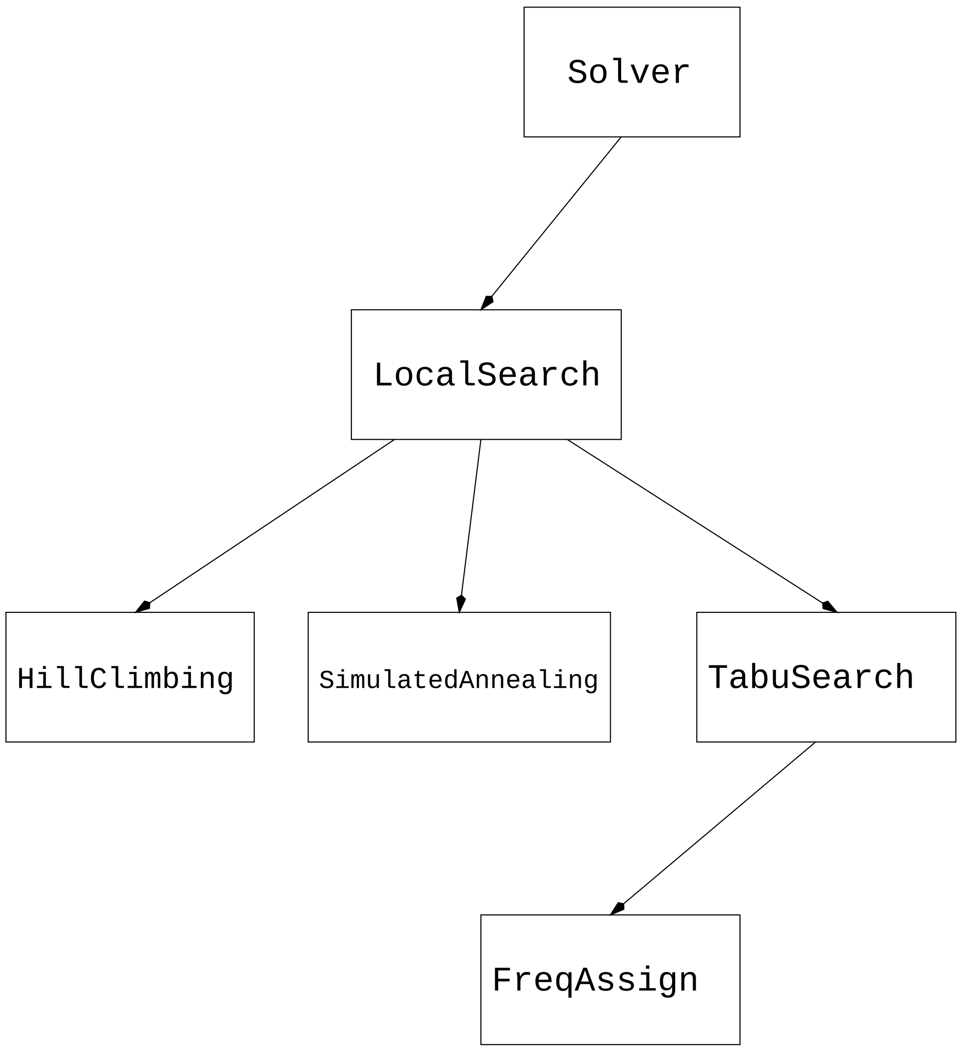
Local Search Software Tools

- Conceptual Frameworks
  [Vaessens et al. 95]

- Black-box systems: LOCALIZER
  [Michel and Van Hentenrick, 97]

→ Object-Oriented Frameworks

Local Search Framework



Abstract Tabu Search

```
template <class Move, class Solution>
void TabuSearch<Move,Solution>::Solve()
{ BestSolution() = CurrentSolution();
  int current_value = CurrentSolution().NumberOfViolations();
  int best_value = current_value;
  while (number_of_iterations - iteration_of_best < max_idle_iteration)
    { number_of_iterations++;
      Move move = FirstMove();
      Move best_move = move;
      int best_delta = CurrentSolution().DeltaNumberOfViolations(best_move);
      do // look for the best move
        { int current_delta = CurrentSolution().DeltaNumberOfViolations(move);
          if (current_delta < best_delta && (!TabuMove(move) || Aspiration(move)))
            { best_move = move; best_delta = current_delta; }
          move = NextMove(move);
        }
      while (!LastMove(move));
      CurrentSolution().MakeMove(best_move);
      PushTabuList(best_move);
      UpdateAspirationFunction(best_move);
      current_value += best_delta;
      if (current_value < best_value)
        { iteration_of_best = number_of_iterations;
          BestSolution() = CurrentSolution();
          best_value = current_value;
          if (best_value == 0)
            { OutputPositiveAnswer();
              return;
            }
        }
    }
  OutputNegativeAnswer();
}
```

Abstract Tabu List Management

```
template <class Move, class Solution>
void TabuSearch<Move,Solution>::PushTabuList(Move m)
{ tabu_list[list_top++] = m.Reverse();
  if (list_top == list_length) list_top = 0;
}
```

```
template <class Move, class Solution>
bool TabuSearch<Move,Solution>::TabuMove(Move m)
{ for (int i = 0; i < list_length; i++)
    if (tabu_list[i] == m)
      return true;
  return false;
}
```

Applications

- Scheduling
  - Timetabling
  - Workforce Scheduling
  - Production Scheduling
  - Sport Scheduling
  - ...
- Others
  - Vehicle Routing
  - Circuit Design
  - Financial Analysis

Conclusions

- State of the art
- Future trends

$\rightarrow$ Critical issues

$\rightarrow$ Essential bibliography

Critical Issues

- Learning while searching
  - Automatic parameter tuning
  - Selection of the heuristic
- Parallelization
- Flexible software tools
- Generalize Local Search and Constructive Search
- ...

Essential Bibliography

- Glover and Laguna. Tabu Search. Kluwer Academic Publishers, 1997.

- Aarts and Lenstra (Eds). Local Search in Combinatorial Optimization. Springer-Verlag, 1995.