6 Heuristics & Metaheuristics

- Despite all optimization techniques, many problems are still too hard to solve optimally
- We can abstain from optimality and use approximations to determine “good” solutions in reasonable time
- In general: heuristic approaches do not guarantee optimality, they trade solution quality vs. computational effort
- A common strategy is called “greedy”
  - In each iteration, the most improving option is applied
  - Note that this strategy often leads to local optima
- In the following, we consider some problem-specific heuristics and problem-independent search strategies called metaheuristics

6.1 The Quadratic Assignment Problem

- The Quadratic Assignment Problem (QAP) is the common model for layout planning problems.
- In this model, every machine or element that has to be placed in the plant has a uniform size
- Every potential location for such an element also has an uniform size
- The main task of the QAP is to find an assignment of elements to potential places where every element is assigned to one definite location while the objective of this model is given by the minimization of the transportation costs

General characterization of the QAP

The QAP bases on the following assumptions:

1. Production and sales programs are given and cannot be influenced by the chosen layout;
2. The sequence of tasks to be executed for every product is known and cannot be changed by the chosen layout;
3. There are *M* elements to be placed in the layout;
4. There are *S* possible positions for the elements in the layout;
5. The transportation paths which connect the *S* possible positions are given;
6. The means of transportation are given;
7. The transportation costs are proportional to the transportation distance and to the transportation quantities.

Observations

- Due to the **assumptions 1,2 and 3**, the transportation quantities $X_{l,m}$ between the elements l and m can be computed. They cannot be changed by the chosen layout.
- Due to the **assumption five**, the distance between two potential locations r and s can be defined by $y_{r,s}$.
- It is irrelevant which measurement is taken to define the distance between two points. It can be an Euclidean or a right angled one.
- Due to the **assumption seven**, there is a constant transportation costs rate k per distance and quantity unit.

Mathematical definition of the QAP I

Parameters:

- M: Number of elements to be placed in the layout;
- $S \geq M$: Number of locations where the elements must be allocated to;
- $y_{r,s}$: Distance between the locations r and s ($r,s \in \{1,2,....,S\}$). Measured in distance units;
- $X_{l,m}$: Quantities to be transported from element l to element m. Measured in transportation units per planning horizon ($l, m \in \{1,2,....,M\}$);
- k: Unified transportation costs rate per distance and transportation unit.

Mathematical definition of the QAP II

Variables:

- $w_{m,s}$: Assignment variable ($m \in \{1,2,...,M\}$; $s \in \{1,2,...,S\}$)
- It is defined as follows

$$w_{m,s} = \begin{cases} 1 & \text{if the } m\text{th element is placed on the } s\text{th position in the layout} \\ 0 & \text{otherwise} \end{cases}$$

The objective function of the QAP

- Only if element *l* is allocated to location *r* and element *m* is allocated simultaneously to location *s*, the distance $y_{r,s}$, respectively $y_{s,r}$, is relevant for the decision
- $X_{l,m}$ units have to be transported from element *l* to *m*
- This fact can be covered by the multiplication of the binary variables $w_{l,r}$ and $w_{m,s}$. In order to get the resulting transportation costs, we have to multiply costs rate *k*
- We get the **following objective function of the QAP**

$$\text{Minimize } Z = \sum_{l=1}^{M}\sum_{m=1}^{M}\sum_{r=1}^{S}\sum_{s=1}^{S} k \cdot X_{l,m} \cdot y_{r,s} \cdot w_{l,r} \cdot w_{m,s}$$

Restrictions of the QAP

1. Every element has to be placed on a definite location, e.g.,

$$\forall m \in \{1,\ldots,M\} : \sum_{s=1}^{S} w_{m,s} = 1$$

2. Every location is occupied by at most one facility, e.g.,

$$\forall s \in \{1,\ldots,S\} : \sum_{m=1}^{M} w_{m,s} \leq 1$$

3. Every decision variable is a binary one, e.g.,

$$\forall m \in \{1,\ldots,M\} : \forall s \in \{1,\ldots,S\} : w_{m,s} \in \{0,1\}$$

An alternative definition…

- In the following, we assume *S*=*M*
- A solution of the model can be seen as a **permutation of the *N* facilities** to be arranged
- Therefore, we have two predefined *N*×*N*-matrices *A* (distance) and *B* (flow)

Sought : A permutation $\pi \in S_N$ which minimizes the following sum: $\text{Minimize}_{\pi \in S_N} \sum_{i=1}^{N}\sum_{j=1}^{N} a_{\pi(i),\pi(j)} b_{i,j}$

where S_N is the **set of possible permutations** of $\{1,2,3,\ldots,N\}$

$\Rightarrow$

As you can see in the definition, the permutation π has to define for every element $i \in \{1,2,\ldots,N\}$ the position $\pi(i)$ in the arising layout.

Observations

- The QAP is also a **very restrictive model** for the mapping of layout planning problems;
- In the QAP, only **uniform machines** can be allocated while the **possible locations are also of uniform size**;
- The model has a **very compact definition**
- However, owing to its high complexity, it is used for many algorithmic techniques as a challenging application.
- The QAP can be **reduced to the simple LAP** by modifying the objective function as follows:

$$\text{Minimize } Z = \sum_{i=1}^{N}\sum_{j=1}^{N}\sum_{k=1}^{N}\sum_{l=1}^{N} d_{i,j,k,l} \cdot x_{i,j} \cdot x_{k,l} \quad \text{with } d_{i,j,k,l} = c_{i,j}$$

$$\Rightarrow$$

$$Z = \sum_{i=1}^{N}\sum_{j=1}^{N}\sum_{k=1}^{N}\sum_{l=1}^{N} c_{i,j} \cdot x_{i,j} \cdot x_{k,l} = \sum_{i=1}^{N}\sum_{j=1}^{N} c_{i,j} \cdot x_{i,j}$$

Complexity

- The **QAP is NP-hard in the strong sense**. Experiments have shown that only moderate instances with up to 20 elements can be solved optimally;
- Therefore, **many heuristic approaches** can be found in literature
- Additionally, a huge set of specific **Branch&Bound procedures** is proposed
- Main problem of an efficient application of Branch&Bound algorithms results from the **absence of tight and computational efficient lower bounds**;
- The efficiency of an applied Branch&Bound algorithm mainly depends on the quality of lower bounds applied during the examination of the solution space;

Heuristics for the QAP

- **Construction methods**
  - Generate a layout from scratch by the iterative assignment of the different facilities
  - Frequently used for generating an initial solution
  - Frequently an iterative application of simple rules
  - One main drawback is that the resulting solution quality of these techniques is often quite poor
- **Improvement methods**
  - Try to improve an already existing solution by the application of a predefined set of operations
  - Local and global search procedures
  - Specific meta-strategies can be distinguished
  - Significantly higher solution quality becomes yieldable

A construction method

- Most simple approaches
- Start with an empty layout and recursively assign locations to facilities according to certain criteria until all facilities have been assigned
- Acceptable results (for a first solution) are yielded by the construction method of Müller-Merbach

- **Reference:**
  - Müller-Merbach, H.: Optimale Reihenfolgen. Springer Verlag, pp.158-171, Berlin, Heidelberg, New York, 1970.

The algorithm of Müller-Merbach

- Method of the increasing degree of freedom:
  - It works with an iteratively updated partial permutation and completes it into a permutation of $\{1,...,N\}$.
  - A partial permutation of $\{1,...,N\}$ in this connection is an injective mapping of a subset $X \subseteq \{1,2,...,N\}$ into $\{1,2,...,N\}$, $\pi: X \rightarrow \{1,2,...,N\}$ with $X \neq \{1,2,...,N\}$.
- The approach of Müller-Merbach starts with an empty permutation, i.e., X is the empty set and with a fixed order of the indices 1,2,...,N, namely, $r_1,r_2,...,r_N$.
- For some $M \subseteq \{1,2,...,N\}$ and $k \in IN$, let $\pi(M)=\{\pi(i) \mid i \in M\}$. Let $\pi_k: M_{\pi,k} \rightarrow \{1,...,N\}$ be the current partial permutation, where it holds $M_{\pi,k}=\{r_1,r_2,...,r_{k-1}\}$.

The algorithm of Müller-Merbach

- Then, in **every step**, a new extended permutation $\pi^1: M_{\pi 1} \rightarrow \{1,2,...,N\}$ is constructed with $M_{\pi 1} = M_\pi \cup \{r_k\}$ where r_k is the first currently unassigned index in $\{1,...,r_N\}$.
  - In order to define the new permutation π^1, r_k is assigned to some j not belonging to $\pi(M_\pi)$.
  - By doing so, the resulting additional costs ΔZ_j are computed.
  - Consider additionally the assignment of r_k to an index $j \in \pi(M_\pi)$.
  - Let $r_i \in \{r_1,r_2,...,r_{k-1}\}$ such that $\pi(r_i)=j$. Define $\Delta Z_{j,l}$ as the change of the objective function which results from assigning r_k to j and r_i to l, for some l not belonging to $\pi(M_\pi)$.
- **In every step of the algorithm, the constellation is chosen that leads to the smallest objective value**.

In every step of the algorithm

In the k-th $(k \in \mathbb{N})$ step:

$\pi: X \to \{1,2,3,4,...,N\}$ while $X \subseteq \{1,...,N\}$

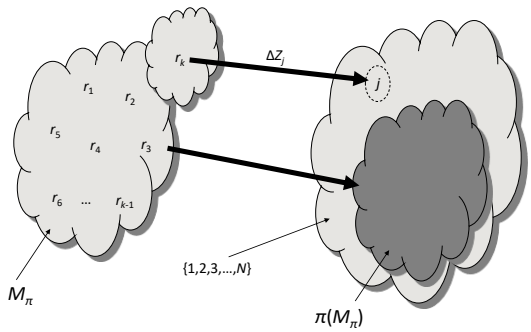
$\pi: M_{\pi,k} \to \{1,2,3,4,...,N\}$ while $M_{\pi,k} = \{r_1,...,r_{k-1}\}$

Generation of a new permutation π^1

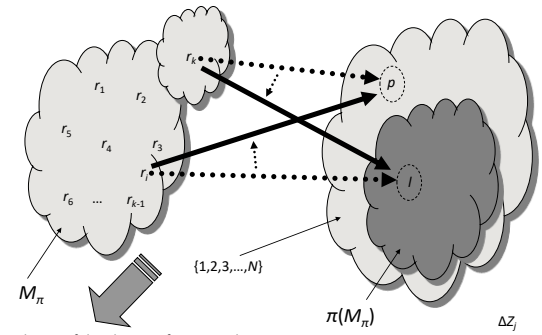
$\pi^1: M_{\pi,k} \cup \{r_k\} \to \{1,2,3,4,...,N\}$ with $\left|\pi^1(M_{\pi,k} \cup \{r_k\}) - \pi(M_{\pi,k})\right| = 1$

After constructing the new permutation π^1, the additional costs ΔZ_j are computed.

First set of possibilities



Second set of possibilities



Change of the objective function value: $\Delta Z_{p,l}$

Complexity

- Choosing *j* in the first set:
  There are altogether *N*-*k*+1 possibilities
- Choosing *l* in the second set:
  There are altogether *k*-1 possibilities
- Choosing *p* in the second set
  There are altogether *N*-*k*+1 possibilities

Total sum of constellations to be considered:

$(N-k+1)\cdot(k-1)+(N-k+1)=k\cdot(N-k+1)$

Total computational effort:

$$\sum_{k=1}^{N} k\cdot(N-k+1)=\sum_{k=1}^{N} kN-k^2+k=N\cdot\sum_{k=1}^{N} k-\sum_{k=1}^{N} k^2+\sum_{k=1}^{N} k=(N+1)\cdot\frac{1}{2}\cdot N\cdot(N+1)-\sum_{k=1}^{N} k^2$$

$$=(N+1)\cdot\frac{1}{2}\cdot N\cdot(N+1)-\frac{N\cdot(N+1)\cdot(2N+1)}{6}=\frac{3\cdot(N+1)\cdot N\cdot(N+1)-N\cdot(N+1)\cdot(2N+1)}{6}$$

$$=\frac{1}{6}N^3+\frac{1}{2}N^2+\frac{1}{3}N\in O(N^3)$$

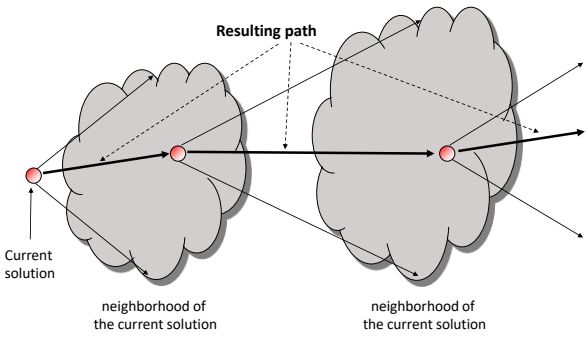
Metaheuristics

- These are general higher-level procedures that define the basic design of search algorithms for efficiently identifying reasonably good solutions for complex problems
- Do not guarantee optimality
- Provide basic rules for guiding the search process in the solution space
- Frequently designed as global search techniques that are theoretically able to overcome local optima
- Many of these approaches are motivated by natural processes, e.g.,
  - Technical processes
  - Biological processes

Path oriented metaheuristics

- These approaches try to **improve an existing solution repetitively**
- These algorithms try to improve an existing solution by the application of certain operations
- Therefore, the computation process of these algorithms is defined as a **sequence of moves** where the current solution is changed by the execution of some **predefined operations**. Owing to this, **in every move, a neighborhood** of potential subsequent solutions is considered
- This set of solutions consists of all constellations which can be generated by a single application of a predefined operation to the current solution
- Depending on the way how the neighborhood is examined and which solution is chosen, we distinguish **different meta-strategies**
- Among them, there is
  - Tabu Search algorithm
  - Simulated Annealing algorithm

Path computation



Solution set oriented metaheuristics

- These procedures maintain a large set of different solutions in memory
- In each iteration this current set of solution is transformed by the application of specific operations
- These operation may combine existing solutions in order to generate improved solutions that are inserted into the next current set
- In genetic procedures these sets are denoted as generations

Intensification vs. Diversification

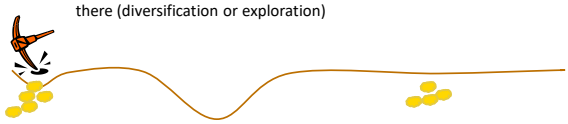
- The balance of the following two concepts significantly influences the success of a metaheuristic
- Intensification
  - Exploitation of the accumulated search experience
  - Goal: quickly identify regions with high quality solutions
- Diversification
  - Exploration of the search space
  - Goal: avoid the wasting of time in regions ...
    - ... which are already explored
    - ... which yield poor solutions

Metaheuristics – Gold seeker illustration

- Imagine a gold seeker during the gold rush in the US
- If the seeker hits gold, a useful strategy would be to search for more in the direct vicinity of the place of discovery
- This can be considered as intensification or exploitation

Having unearthed all the gold of an area or the inability to find any gold at all in an area demands a different strategy:

- The seeker could look around in areas farer away and dig at random there (diversification or exploration)



6.2 Simulated Annealing

- Heuristic method which tries to overcome local optimality;
- Bases on the **analogy between combinatorial optimization problems and problems from statistical mechanics**
- The Metropolis algorithm that simulates the behavior of a physical many-particle system is naturally applied as a heuristic method in optimization. There are **analogies between a combinatorial optimization and a many-particle physical system** that basically rely on the following two facts (c.f. Albrecher et al., 2006):
  - Feasible solutions of the combinatorial optimization problem correspond to states of the physical system
  - The value of the objective function value for a feasible solution of the combinatorial optimization problem corresponds to the energy of the state of the physical system

Annealing

- **Annealing** is a **thermal process** for obtaining lower energy states of a solid in a heat bath. Two phases process:
  - One increases the temperature of the heat bath to a maximum value at which the solid melts;
  - One carefully and slowly decreases the temperature of the heat bath until particles arrange themselves in the ground state of the solid. This ground state is characterized by a minimum of energy.
- The **Metropolis algorithm** works with **Monte Carlo techniques for generating a sequence of states of the solid**.
- Let i be the **current state with energy** E_i. A possible subsequent state j with energy E_j is generated by displacing one of the particles.
  - If the resulting energy difference E_j-E_i is negative, the new state is accepted.
  - Otherwise, j is accepted with a certain probability given by $e^{\frac{(-1)\cdot(E_j - E_i)}{t \cdot k_B}}$

  where t denotes the temperature and k_B is the so-called Boltzmann constant

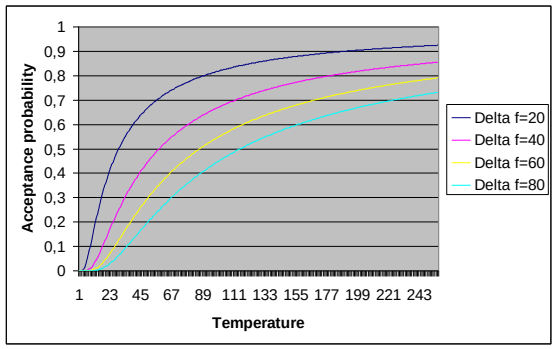
Thermal process vs. combinatorial problem

- Analogies between the thermal process and a combinatorial optimization problem:
  - System states=Feasible solutions to the problem
  - Energy states=Objective values
  - Change of the system state=Move between different solutions
  - Temperature=Control parameter for every move
  - Frozen state=Heuristic solution to the problem

Pseudo Code

1. Generate an initial solution S_0;
2. Generate an initial temperature T_0 and store it in the variable t;
3. Generate a temperature function α;
4. Repeat
   1. Choose S randomly out of the neighborhood S in $N(S_0)$;
   2. $\Delta := f(S) - f(S_0)$;
   3. If $\Delta<0$, then $S_0=S$; otherwise, randomly generate x out of the continuous interval [0,1]
   4. If $x < e^{(-\Delta/t)}$, then $S_0=S$
   5. Generate a new temperature according to the chosen function α
5. Until a specific criterion is fulfilled
6. Return the best found solution after applying hill climbing to it (not mandatory)

Acceptance functions



Observations

- In the frozen state, nearly no deterioration can be executed
- With an increased temperature, the acceptance probability for the deteriorating moves enlarges
- Therefore, the definition of the cooling schedule significantly influences the efficiency of the executed improvement process

Finding efficient cooling schedules

- Thonemann and Bölte propose the application of a genetic algorithm to find a well-working cooling schedule

Reference:

Bölte, A., Thonemann, U., 1996. Optimizing simulated annealing schedules with genetic programming. European Journal of Operational Research 92, 402-416.

6.3 Genetic algorithms

- Genetic programming is based on Darwin's evolutionary theory about "survival of the fittest"
- In order to apply the principles of this theory to combinatorial problems, several generations of feasible solutions are created by using specific operations representing evolutionary processes in nature
- By applying these genetic operators on the individuals of the current generation, the next population is produced

Genetic algorithms

- Basic genetic operators are:
  - Reproduction: Copying an individual from the current population, without alteration, into the next generation
  - Crossover: New offspring is generated consisting of attributes of two individuals of the current population (parents)
  - Mutation: Copying an individual from the current population, with alteration, into the next generation
- Fitness is measured by the respective objective function value and is used as the main attribute to be selected for the generation of the subsequent population
- Simulation of a natural evolutionary process to adapt the population according to the attributes of the considered problem

6.3.1 Genetic operators for cooling schedules

- Temperature functions are defined in the programming language LISP
- Variable *t* (Time)
- Resulting function value is interpreted as the temperature in *t*
- For instance $0.7^{t} \cdot 10$ = (*(^0.7 *t*) 10)
- By combining terminals and functions, different expressions can be generated

First generation

- *F*={+, -, *, %, ^, sign, cos, sin}
- *T*={*t*, *R*}, while *R*=[-10.0, +10.0] subset of *IR*
- The initial population is randomly generated
- For every individual, at first an element of *F* is chosen as the root node
- Then, an arc is attached for each of the required arguments. Next, one element of *F* or *T* is chosen for every element of the arcs. If it is a terminal, the branch is terminated; while a function leads to additional arcs
- While the minimum depth is two, every randomly generated expression tree can have at most six levels
- In order to get a wide variety of different constellations in the first population, 20% of the generated trees have depth 2, 20% have depth 3, 20% have depth 4, 20% have depth 5, and finally 20% have depth six
- In every group, 50% of the trees are generated as full trees where every path has the predefined length

Fitness and selecting

- An individual I is selected according to the following fitness level $AF(I)$:

$$AF(I) = \frac{1}{1 + SA(I) - LB}$$

In this formula, $SA(I)$ is the objective function value of the Simulated Annealing procedure working with I as the annealing schedule while LB denotes a lower bound of the objective function value. In order to decide about the selection during the population generation process, a normalized fitness function is used:

$$NF(I) = \frac{AF(I)}{\sum_{I'} AF(I')}$$

Selecting

- A selected individual is not removed from the current population, i.e., the selection is performed without replacement
- Therefore, an individual can be selected more than once

- Applied operators:
  - Reproduction operation
  - Crossover operation

Reproduction and mutation operation

Reproduction:

- Good individuals remain in the population with probability NF
- 10% of the population is created by this operation

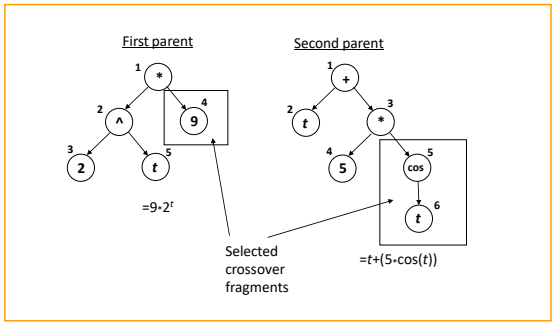
Mutation:

- Thonemann and Bölte argue that due to the nonlinear structure of the QAP, the use of mutation operations is not necessary
- Additionally, mutation is part of the crossover operations

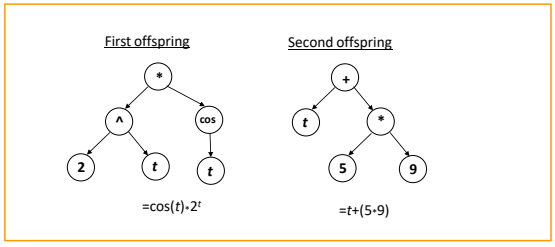
Crossover operation

- First, two individuals are selected with the probability *NF*
- Both trees are labeled to get definite possible crossover points
- A crossover point is randomly selected for the first parent
- The corresponding node becomes the root of a subtree which we will refer to as the crossover fragment
- A second crossover fragment from the second tree is generated subsequently by a random selection of a node
- In order to obtain two offsprings (individuals of the next generation), we attach the crossover fragment of the first tree to the remainder of the second tree and vice versa
- 80 % of inner nodes and up to 20 % of leafs are used
- Maximal depth is 17

Crossover operation example



Crossover operation example



Note that if a resulting offspring has more than 17 levels, it is deleted while one of the parents is copied to the subsequent generation

6.3.2 Applied neighborhood search

- In each step, the total neighborhood consists of all applicable exchanges of two machines that are arranged in the layout
- Therefore, we get the total number of altogether $\frac{1}{2} \cdot N \cdot (N-1)$ applicable operations
- In every move of the SA procedure, the tested operations can be chosen randomly as well as iteratively in a predefined order. The latter one can be implemented by applying a lexicographical order:
  (1,2),(1,3),(1,4),…,(1,*N*),(2,3),…,(2,*N*),…,(*N*-1,*N*)
  - Note that the following move always starts examining the subsequent exchange operation to the one considered before
  - Experimental analysis underline that the systematic examination of the neighborhood yields better results in comparison to the random choice

Algorithm TB1

1. Generate Generation 0
2. *Gen=0*;
3. *I*=1
4. Optimize the problem with SA using cooling schedule *I*
5. Compute fitness of *I*
6. *I*=*I*+1
7. If *I*≤500 go to step 4
8. If *Gen*=50 go to step 12
9. Produce next generation
10. Gen=Gen+1
11. Go to step 3
12. Report best solution found

Parameters in TB1

- The problem data is normalized in order to get values in the matrices between 0 and 10
- The number of attempted pairwise exchanges is restricted to $SL(N)=0.5 \cdot C \cdot N \cdot (N-1)$ with $C=50$
- The subchain length that is used for the duration of the application of each current temperature is set to 500
- 51 generations are considered with 500 individuals, respectively
- Therefore, we obtain, all in all, 25.500 executions of the SA algorithm
- Consequently, only small problems can be solved
- The procedure was applied to five specific problems of a well-known benchmark set

Results of the computational tests

- Annealing schedules with a constant temperature oftentimes performed well, but TB1 always generates a non-constant one that performed at least as well
- Oscillating schedules enable the search process to escape from local minima also in later parts of the performed computation
- Thonemann and Bölte report that TB1 yielded quite well results for moderate-sized experiments

Properties of good annealing schedules

- The yielded schedules are tailored to the respective instance. Therefore, they perform poorly if applied to other (different) problems
- Therefore: An additional experiment was executed:
  - Three instances were solved again by the TB1 procedure with two different initial assignments
  - By making use of different random number streams, each run was executed twice
  - Consequently, every schedule was used for 12 different optimizations

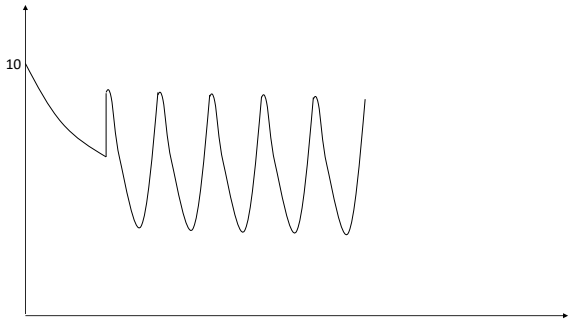
Attributes of the best schedules

1. Most of the time, the temperature was between 4 and 8
2. The temperature was usually not constant or monotonously decreasing, but oscillated between 4 and 8
3. The period length of the oscillation was between 20 and 100 (based on a schedule length of 500)
4. The shape of the oscillation was not relevant
5. Starting and ending temperature was far above zero

6.3.3 The procedure TB2

- Uses a predefined cooling schedule with some of the itemized attributes of good schedules
- Starting temperature is $v(0)=10$;
- Subsequent cooling phase:
  - $v(t)=v(t\text{-}1)/(1+b\cdot v(t\text{-}1))$; $b=8/(20\cdot SL(N))$
  - Annealing schedule length of $SL(N)=C\cdot N\cdot(N\text{-}1)/2$ with C=50, C=250, C=1000
  - If $SC(N)=N\cdot(N\text{-}1)/4$ consecutive pairwise exchanges are rejected, cooling is stopped with v_c and t_c
- After the cooling process
  - $v(t)=v_c\cdot[1+½\cdot\cos(w(t\text{-}t_c))]$ is used with $w=16\pi/(25\cdot N\cdot(N\text{-}1))$
  - Oscillates around v_c with an amplitude of $0.5v_c$

Structure of the resulting annealing schedule



The procedure

1. $E^*=Z(a^0)$; $E=Z(a^0)$; t=0; s=0; $a^*=a^0$; $a=a^0$; v=10; b=8/(20 SL(N)) /* with annealing schedule length SL(N)=C·N(N-1)/2 */
2. Choose x,y
3. If $\Delta Z_{x,y}<0$, then go to 6
4. g=unif[0,1]
5. If $g\geq\exp(-\Delta Z_{x,y}/v)$, then $s=s+1$; go to step 8
6. $E=E+\Delta Z_{x,y}$; Exchange x and y; s=0;
7. If $E<E^*$, then $a^*=a$; $E^*=E$
8. $t=t+1$
9. If $t=SL(N)$, go to step 14
10. If b=0, then go to step 13
11. If $s\geq SC(N)$, then $v_c=v$; $t_c=t$; b=0; Go to step 13
12. $v=v/(1+b\cdot v)$; Go to step 2
13. $v=v_c\cdot[1+½\cdot\cos(w(t\text{-}t_c))]$; Go to step 2
14. Apply hill climbing to a^*

Computational results

- Thonemann and Bölte show that the TB2 procedure yields substantial improvements of the solution quality in comparison to Connolly's SA approach
- TB2 generates improved results, particularly for more complex problem instances
- By increasing the value of parameter C, the solution quality of TB2 is further improved, but at the expense of a significantly increased computational effort

6.4 Tabu Search (TS)

- The main idea of Tabu Search is to select **the best not forbidden neighbor in every move** of the computation
- This is an HAMD (=highest ascent mildest descent) strategy
- This exhaustive search increases the intensification
- Since in its basic form Tabu Search is deterministic, a tabu list is additionally applied that temporarily forbid specific moves
- These tabu moves may bring back the searching process to a solution that was already visited before, i.e., prevention of cyclical computations
- The **main ingredients of Tabu Search** are the **neighborhood structure**, the **moves**, the **tabu list**, and the **aspiration criterion**
- An **aspiration criterion** is a condition which, when fulfilled by a tabu move, overrules its tabu status

6.4.1 Tabu Search for the QAP

- In case of **QAP**, the moves are usually transpositions and the **neighborhood is the pair-exchange neighborhood**
- Frequently, the inverted exchange is forbidden by a tabu status for a predefined number of moves
- Alternatively, fingerprints of current solutions can be computed and compared
  - After visiting a new solution its fingerprint is computed
  - If it has occurred before there is an entry and the considered solution may be blocked if it does not improve the best solution found so far
  - This criterion identifies very reliable identical solutions if more than one fingerprint is applied

Tabu Search Pseudocode

A generic Tabu Search algorithm:

- Select an initial solution $x \in X$
- $x^* := x$, k:=0, T:=empty
- While ($k<K$ and $S(x)$-T not empty)
  - k:=k+1
  - Select $s_k \in S(x) - T$, such that $s_k(x)$ = OPTIMUM($s(x)$:s from $S(x)$ -T)
  - x:=$s_k(x)$
  - If ($c(x)<c(x^*)$)
    - x^*:=x
  - Update T
- End While

X: set of all feasible solutions
x: current solution
$c(x)$: objective value of solution x
$S(x)$: Neighborhood function
T: set of forbidden tabu moves
k: iteration counter
K: max number of iterations
x^*: best found solution
$s_k(x)$: best non-tabu solution out of the neighborhood of x in iteration k

6.4.2 Sophisticated TS for the VRP

- In what follows, we introduce a series of powerful Tabu Search approaches for the well-known VRP
- These approaches try to overcome the limited diversification within the enumeration process of Tabu Search by integrating this procedure into an extended procedure
- This procedure applies the Tabu Search optimization process iteratively in different parts of the solution space in order to temporarily intensify the search process
- A so-called adaptive memory stores best parts (tours) of completed solutions in order to combine them in later steps

6.4.2.1 The Vehicle Routing Problem (VRP)

- Standardized problem for mapping distribution processes conducted by a fleet of vehicles
- Basic attributes
  - Fleet of vehicles whose tours have to be planned
  - Single depot or multiple depots where all tours to be planned start and end
  - Pure tours, i.e., pure distribution or pickup tours
  - Capacitated or incapacitated case
  - Time window integration
  - Sought: Tours of the vehicles, Time tables
  - Restrictions:
    - Each Customer has to be visited once
    - Each tour starts and ends at a predefined depot
    - Compliance with existing time windows
    - Compliance with capacity restrictions

Time window constraints

- A very important model extension comprises the integration of specific time restrictions
- Therefore, specific time windows are defined by upper and lower bounds while the respective pick up or delivery activity can be executed only within this predefined interval
- By integrating time windows, the VRP becomes to the VRPTW or VRPSTW
- These extensions, however, require the integration of a detailed time table planning
- Thus, this problem is frequently denoted as the Vehicle Routing and Scheduling Problem

Soft vs. Hard time windows

- **Hard time windows**
  - Do not allow any violations of the defined intervals, i.e., each pickup or delivery activity has to be generated inside each time window
  - Can prevent feasible solutions
- **Soft time windows**
  - Can be violated, but this causes additional costs, i.e., the respective pickup or delivery activity does not have to be executed within the time window
  - Have no impact on the set of feasible solutions, i.e., can be used for real-time control activities
  - More general case, i.e., hard time windows can be mapped as soft time windows with infinitive costs for violation

The CVRPSTW

N : Number of customers to be satisfied. Again, the model is mapped as a directed weighted graph where each customer is a node, numbered from 1 up to N while the single depot gets the number zero

V : Number of available vehicles

$c_{i,j}\ (0 \le i, j \le N)$: Costs occurring for traveling from node i to j

$d_i\ (1 \le i \le N)$: Demand of customer i

C : Capacity of all vehicles

CVRPSTW – Parameters

$[e_i, l_i]$ $(0 \le i \le N)$: Time window at customer *i*. These times are defined according to the end of the delivery processes, i.e., the service times are included. Note that e_0 is the earliest start time and l_0 is the latest end time of each vehicle that can be used

s_i $(0 \le i \le N)$: Service time for picking up or unloading the goods at customer *i*. It holds $s_0 = 0$

$\bar{t}_{i,j}$ $(0 \le i, j \le N)$: Time for traveling from node *i* to *j*

α_i $(0 \le i \le N)$:Cost rate per time unit lateness at customer *i*

CVRPSTW – Variables

$x_{i,j,v}$ $(0 \le i, j \le N; 1 \le v \le V)$: Binary decision variable deciding about the tours of the available vehicles, e.g., if vehicle *v* travels directly from *i* to *j* $\Leftrightarrow x_{i,j,v} = 1$

$y_{i,v}$ $(0 \le i \le N; 1 \le v \le V)$: Binary decision variable which is one if and only if vehicle *v* satisfied the demand of customer *i*

$t_{i,v}$ $(0 \le i \le N+1; 1 \le v \le V)$: Point of time when vehicle *v* finished serving customer *i*. N+1 represents the arrival at the depot after the execution of the respective tour

A solution *L* is defined completely by

$$L = \{x_{i,j,v} \mid 0 \le i, j \le N; 1 \le v \le V\} \cup \{y_{i,v} \mid 0 \le i \le N; 1 \le v \le V\} \cup \{t_{i,v} \mid 0 \le i \le N+1; 1 \le v \le V\}$$

CVRPSTW – Restrictions

1. Capacity requirements: $\forall v \in \{1,...,V\} : \sum_{i=1}^{N} d_i \cdot y_{i,v} \le C$

2. Demand satisfaction: $\forall i \in \{1,...,N\} : \sum_{v=1}^{V} y_{i,v} = 1$

3. Each customer i that is satisfied by vehicle v is left by this vehicle exactly once:

$\forall i \in \{0,...,N\} : \forall v \in \{1,...,V\} : \sum_{j=1}^{N+1} x_{i,j,v} = y_{i,v}$

CVRPSTW – Restrictions

4. Each customer i that is satisfied by vehicle v is visited by this vehicle exactly once:

$$\forall i \in \{1,...,N\} : \forall v \in \{1,...,V\} : \sum_{j=0}^{N} x_{j,i,v} = y_{i,v}$$

5. Preventing inner cycles:

$$\forall v \in \{1,...,V\} : \forall Q \subseteq \{1,...,N\} \text{ with } 2 \le |Q| \le N :$$

$$\sum_{i \in Q} \sum_{j \in Q} x_{i,j,v} < |Q|$$

6. Node 0 belongs to each cycle: $\forall v \in \{1,...,V\} : y_{0,v} = 1$

CVRPSTW – Restrictions

7. Start time of each vehicle tour:

$$\forall v \in \{1,...,V\} : t_{0,v} \ge e_0$$

8. End time of each vehicle tour:

$$\forall v \in \{1,...,V\} : t_{N+1,v} \le l_0$$

9. Earliest completion of delivery:

$$\forall i \in \{1,...,N\} : \forall v \in \{1,...,V\} : e_i \cdot y_{i,v} \le t_{i,v}$$

10. Time dependencies inside the tours:

$$\forall v \in \{1,...,V\} : \forall i,j\,(i \ne j) \in \{1,...,N\} :$$

$$t_{i,v} \cdot y_{i,v} \ge \left(t_{j,v} \cdot y_{j,v} + \bar{t}_{j,i} + s_i\right) \cdot x_{j,i,v} \wedge$$

$$\forall v \in \{1,...,V\} : \forall j \in \{1,...,N\} : t_{N+1,v} \ge \left(t_{j,v} \cdot y_{j,v} + \bar{t}_{j,0}\right) \cdot x_{j,0,v}$$

CVRPSTW – Objective function

$$\text{Minimize } F(L) = \underbrace{\sum_{i=0}^{N} \sum_{j=0}^{N} c_{i,j} \cdot \left(\sum_{v=1}^{V} x_{i,j,v} \right)}_{\text{Transportation costs}}$$

$$+ \underbrace{\sum_{i=1}^{N} \sum_{v=1}^{V} y_{i,v} \cdot \alpha_i \cdot \max\{t_{i,v} - l_i, 0\}}_{\text{Lateness costs}}$$

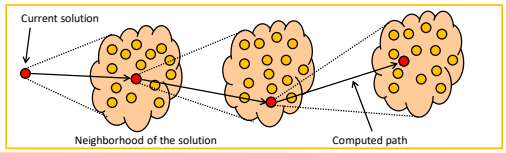
6.4.2.2 The procedure of Taillard, Badeau et al.

- Makes use of the meta-strategy Tabu Search
- This is a well-known metaheuristic attaining a competitive solution quality for various complex optimization problems
- While only the basic structure of this strategy is predetermined, the applied instruments/ procedures have to be generated application-dependent

Basic attributes of Tabu Search

- A single current solution and a best solution are stored
- By applying specific operations, this current solution is modified iteratively
- In each iteration (move), solutions that result from a single application of the operations define the neighborhood of a current solution
- Tabu Search selects the best performing alternative that is not set tabu out of the current neighborhood
- Tabu states are assigned to solutions in order to prevent cyclical computations

Illustration of the computation:



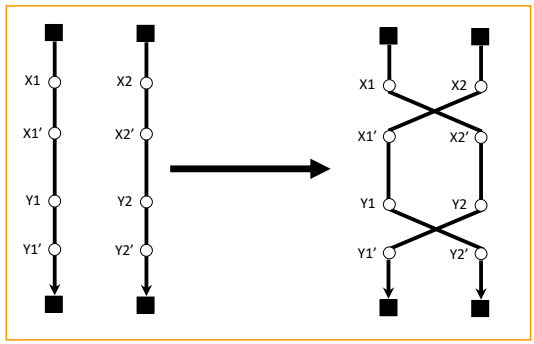
The used exchange heuristic

- Generally speaking, Tabu Search usually attains a high intensification, i.e., applied to a specific locality in the solution space, it may be able to find the best solutions located there (if the neighborhood is reasonably defined)
- However, since the computational effort is considerable, Tabu Search may suffer from limited diversification
- Hence, Taillard, Badeau et al. deal with this tradeoff by integrating Tabu Search into a diversifying environment
- In the procedure of Taillard, Badeau et al., the neighborhood is defined as a set of exchange operations resulting from the application of a specific CROSS operation which exchanges elements of two different tours
- In order to yield a high solution quality, the CROSS operation contains a large variety of possible constellations resulting in a very flexible neighborhood

The used exchange heuristic

- This well-known operation exchanges parts of two selected tours
- In order to do so, we have to chose two edges from each tour
  - First route: Edges (X1, X1') and (Y1, Y1')
  - Second route: Edges (X2,X2') and (Y2, Y2')
- In order to generate the subsequent constellation, we delete these edges in the current solution and integrate in tour 1 the subtour X2' – Y2 and in tour 2 the subtour X1' – Y1. Therefore, the edges X1 – X2' and Y2 – Y1' are added in tour 1 and the edges X2 – X1' and Y1 – Y2' in tour 2
- Always the best constellation is implemented

CROSS-exchange

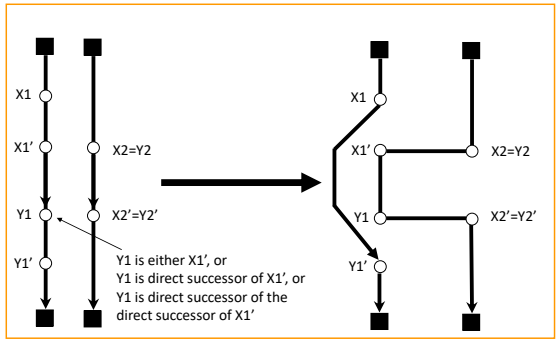


Or-opt

- Originally proposed by Or (1976)
- Moves all possible sequences of at most three visited customers to another tour
- This can be simulated by the CROSS operation as follows
  - By setting X2=Y2 and X2'=Y2', we have an empty subtour to be exchanged in tour 2
  - In order to exchange at most three visited customers, we set Y1 either
    - equal to X1',
    - or equal to the direct successor of X1'
    - or equal to the direct successor of the direct successor of X1'

    in tour 1

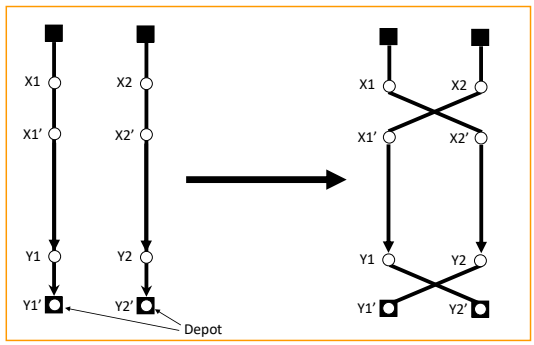
CROSS exchange – Or-opt simulation



2-opt*

- Originally proposed by Potvin and Rosseau (1995)
- Only exchanges two edges taken from different tours
- I.e., the remaining steps up to the depot in both tours are exchanged
- Consequently, we can simulate 2-opt* by the flexible CROSS operation as follows
  - Y1' and Y2' are the depots, i.e., there is no return to remaining steps of the respective original tour

CROSS-exchange – 2-opt* simulation



Neighborhood

- The resulting neighborhood contains all possible applications of the CROSS operation to two arbitrarily chosen tours in the current solution
- We restrict the possible segment length of the subtours to be exchanged to L while N is the number of existing customers
- The application of the CROSS operation preserves the orientation of the respective tours
- Therefore, each neighborhood contains altogether N possibilities to choose the first edge while at most L possibilities remain for the second one
- Hence, for each exchange, we have altogether $O(N^2 L^2)$ possible constellations

Neighborhood – Move evaluation

- Each possible examined move has to be evaluated to decide about its realization
- Due to the large size of each considered neighborhood, it is necessary to generate an efficient technique in order to compute the objective value of each considered constellation
- In the following, we simplify the objective function by introducing equal weights, i.e., we assume the modified objective function value for each solution S using M chosen edges with transfer costs $d_1, d_2, \ldots, d_M$:

$$f(S) = \sum_{k=1}^{M} d_k + \alpha \cdot \sum_{i=1}^{N} \max\{0, t_i - l_i\}$$

Neighborhood – Move evaluation

- After executing an arbitrary CROSS operation, the following objective difference is calculated:

$$\Delta f = \underbrace{\Delta d}_{\text{Distance difference}} + \alpha \cdot \underbrace{\Delta l}_{\text{Lateness difference}}$$

- It is easy to evaluate Δd in constant time since we only subtract the length of the (at most four) edges that have been removed and add the (at most four) edges that have been inserted. Note that, due to homogeneous vehicles, exchanged subtours do not affect the objective function value
- Unfortunately, an accurate computation of Δl cannot be executed in constant time. Therefore, the following separation is used:
- For the second part, an approximation function is used:

$$\tilde{\Delta} l = \Delta l_{X2'\text{-}Y2} + \tilde{\Delta} l_{Y1'\text{–depot}}$$

$$\tilde{\Delta} l_{Y1'\text{–depot}} = g_{Y1'}(\Delta b_{Y1'})$$

Exact evaluation of $\Delta l_{X2'-Y2}$

- This is possible in constant time
- Let us consider the modified tour 1. The service beginning b_i at customer i is modified by Δb_i
- Therefore, we can compute $\Delta b_{X2'}=b_{new,X2'}-b_{X2'}$
- With $b_{new,X2'}=\max\{e_{X2'},b_{X1}+s_{X1}+t_{X1,X2'}\}$
- If we propagate these values along the tour, we cannot yield a constant complexity
- But in order to reduce the computational complexity to constant size, we use the following trick
  - Fortunately, the exploration of the total neighborhood is done in such a way that we can use results that are computed in the previous enumeration step
  - For this purpose, we execute the following mainframe characterized by four nested loops

Four nested loops

- For X1 from depot to the last customer
  Set X1' to the immediate successor of X1
  - For X2 from depot to last customer
    Set X2' to the immediate successor of X2
    - For Y1 from X1 to depot
      Set Y1' to the immediate successor Y1
      - For Y2 from X2 to depot
        Set Y2' to the immediate successor Y2

Observation

- Up to the end of the exchanged segments, we can use the results of the preceding iteration for the subsequent one which contains only one additional step since we have increased its length at most by one
- Therefore, in each iteration, we have only a constant complexity to adjust the objective function value $\Delta l_{X2'-Y2'}$

Approximation of $\Delta l_{Y1'-depot}$

- With Δb_{Y2}, we can compute $\Delta b_{Y1'}$ in constant time
- This offset has an additional impact on the subsequent tour steps up to the depot
- Unfortunately, we cannot directly generate this impact (i.e., in constant time) from the results of the neighborhood constellation considered before
- In order to guarantee an efficient neighborhood examination, we use a function g_i that defines an approximation value of the resulting additional lateness for a given offset value Δb_i
- This functions is updated each time the best CROSS exchange value is applied to the current solution, i.e., **the functions are kept unchanged during the neighborhood examination**
- Furthermore, the update is only performed at customer locations found on the two routes involved in the CROSS exchange

Approximation of $\Delta l_{Y1'-depot}$

- After the generation of the best found CROSS exchange, the update is executed for each effected customer location along the tours
- Specifically, Z=6 values are used to generate the corrected approximation function for each customer
- This is a compromise between an adequate approximation and the reduction of the computational effort
- Specifically, a correlation coefficient varying from 0.5 to 0.8 has been observed between the approximation function value and the true modification

Chosen values

- Positive shifts (positive values):
  - $z_{i,1}$: Set to the maximum modification at some customer observed during the preceding examination
  - $z_{i,2}$: = $z_{i,1}/50$
  - $z_{i,3}$: = $z_{i,1}/2500$
- Negative shifts (negative values):
  - $z_{i,4}$: =$z_{i,6}/2500$
  - $z_{i,5}$: =$z_{i,6}/50$
  - $z_{i,6}$: Set to the lateness at customer i plus total lateness at all subsequent customers. Note that this value determines the possible total reduction of all lateness in the subtour starting at customer i, i.e., the local upper bound for lateness costs reduction of this subtour

Neighborhood examination

- By using the approximation function, the P(=15) best solutions are saved for further considerations
- Subsequently, the best one of them **rated by the exact objective value** is implemented
- Therefore, there is a **two-stepped examination** starting with an approximation followed by an exact one only analyzing the best 15 ones

Feasibility

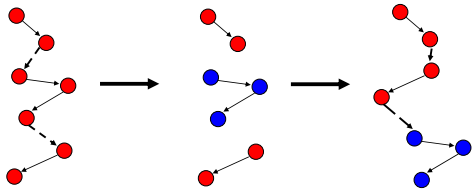
- Considered restrictions:
  - Capacity constraints for each used vehicle
    - ➢ Easy to check by integrating the current capacity requirements in each step of every tour
  - Hard time window at the depot
    - ➢ Can be checked in constant time by integrating the current latest beginning of the customer services in each step of every tour. If this date is violated at customer Y1' (respectively Y2'), the modified solution is not feasible any more. Note that this latest date can be updated in constant time after executing the CROSS operation

Neighborhood reductions

- The size of the neighborhood can be reduced by discarding moves that are unlikely to yield any improvement
- Reduction rules:
  - The execution of the nested loops is reduced by the **stoppage of the inner loop at level Y2** if a monotonous degradation of the (approximate) objective value is observed over three consecutive iterations
  - The same approach is applied at the level of X2. In this case, the objective value associated with X2 is the best solution found after iterating completely over Y1 and Y2
  - Note that **this interruption leads to a significant reduction of the computational effort since the variable X2 is iterated in the second inner loop**

Intra-route exchange

- Beside the CROSS operation which is applied in different constellations of each step of the search process, it make sense to integrate operations modifying the sequence of a single tour only instead of reducing the neighborhood to the exchange of steps between two different tours
- Therefore, an additional operation is applied. Namely, two edges of a given tour are removed, and the segment between the two edges is moved to another location within the same route



The algorithm

1. Construct I different solutions using a stochastic insertion heuristic. Then, apply the Tabu Search procedure to each solution and store subsequently the resulting routes in the adaptive memory
2. While the stopping criterion does not apply, do:
   1. Construct an initial solution from the routes found in the adaptive memory, and define this solution to be the current solution
   2. For W iterations do:
      1. Decompose the current solution into C disjoint subsets of routes
      2. Apply the Tabu Search on each subset of routes
      3. Reconstruct a complete solution by merging the new routes found by the Tabu Search, and define this to be the new current solution
   3. Store the routes of the current solution in the adaptive memory
3. Apply a post-optimization procedure to each individual route of the best solution

Initialization

- Here, we have to fill the adaptive memory used in the following to generate new combined constellations
- For this purpose, the **routes are initialized by randomly selecting m seed customers**. The remaining non-serviced customers are inserted one by one (in a random order) at the location that minimizes the function c_1 used in Solomon's heuristic I1
- Consequently, we receive I solutions which are subsequently improved one by one using the Tabu Search algorithm described later
- Finally, all routes of the I solutions are stored in the adaptive memory. By doing so, the adaptive memory comprises a large diversity of solution routes

The adaptive memory

- At first, the **memory is partially filled** with routes produced during the initialization process
- The routes associated with the **best solutions are found in the first positions** of the memory
- In order to generate a new combined solution, one route after another is **randomly chosen** to be integrated in a new feasible solution
- Therefore, **routes associated with a better solution get a higher selection probability**
- Once, the first route is selected, the routes in memory with at least one common customer are discarded from the selecting procedure currently in execution
- One by one, further routes are selected…
- This procedure is repeated until the set of selected routes **covers all customers** or until there is **no admissible route in memory**. In the latter case, Solomon's insertion procedure I1 is invoked to insert the remaining customers. If this fails additionally (due to capacity and/or depot's time window restrictions), the customers are left aside temporarily but reinvoked at each D&R step to try to insert these customers in the current solution

Decomposition / Reconstruction

- Here, we have to define how the decomposition process is executed to **decompose a current solution in altogether C sets of routes**
- An **upper bound U is predefined for the maximum number of routes belonging to the same subproblem**
- Therefore, we determine **C as the smallest integer value larger or equal to one, such that m/C≤U** while m (number of vehicles) defines the number of routes belonging to the solution currently considered
- A procedure **related to the sweeping technique** is used to generate altogether C regions of tours which are modified by the application of the Tabu Search procedure. The decomposition changes from one D&R to the next one by choosing a different starting angle for creating sectors, thus allowing the CROSS exchange heuristic to exploit new pairs of routes
- After the execution of W D&R-operations, the final routes are stored in the adaptive memory, if it is not filled yet. Otherwise, the routes of the worst solution found in the adaptive memory are discarded and replaced by the new ones (if the new solution is better than the solution the old tours belong to)

The Tabu Search

- Exploits the neighborhood described before in each step of the executed improvement process
- It is applied to the initial solution as well as to the subproblems generated by each decomposition step, i.e., in each of the W D&R steps
- Stopping criterion:
  - Only A[1+(DR-1)/B] steps are executed
  - A,B: parameters and DR is the number of the current D&R step, i.e., **there are more steps of the Tabu Search procedure at the end of each improvement process**

The steps of the Tabu Search procedure

1. Set the current solution to the initial subset of routes
2. While the stopping criterion is not met, do:
   1. Generate the neighborhood of the current solution by applying the CROSS exchanges
   2. Select the best non-tabu solution in this neighborhood and define this solution to be the new current solution
   3. If the current solution is better than the best overall solution, then **reorder the customers within each route using Solomon's I1 insertion heuristic**; define this new solution (if it outperforms the former one) to be the new current solution and the best overall solution
   4. Update the Tabu list
3. Return the best overall solution

Stopping criterion

- The Tabu Search stops after a certain number of iterations
- This number is calculated by the formula

$$A \cdot [1+(DR-1)/B]$$

where A and B are predefined parameters and DR gives the iteration number DR=1,…,W, i.e., the algorithm is applied in longer periods towards the end of the procedure. This is due to the fact that **towards the end of the algorithm an improvement frequently requires more elaborated search processes. This makes longer computation paths reasonable.**

Tabu list

- Length T
- Its positions are indexed from 0 to T-1
- At the position, the **iteration number is stored at which the respective solution will loose its Tabu status**
- After executing a CROSS operation, the generated **objective value modulo T provides its Tabu list position**. If the value found at this position is larger than the iteration number, the **move is Tabu**; otherwise it is accepted
- Note that this approach can filter out legitimate solutions
- However, if T is large enough and the possible objective values differ frequently, this occurs rarely
- The Tabu tenure is defined as the number of iterations divided by two
- Note that only integer objective function values are possible

Diversification methods

- In order to lead the search process in new currently not considered regions of the solution process, the algorithm **penalizes CROSS exchange constellations** that are frequently performed during the search process
- Therefore, we introduce the following parameters:
  - fr_e is the frequency of a given exchange e
  - fr_{max} denotes the maximal frequency observed during the searching process
  - *iter* gives the current number of iterations
  - n defines the total number of customers to be serviced
  - m defines the number of routes, respectively

Diversification methods

- If x is a random value uniformly chosen in the interval [0.0,0.5] and $\Delta_{max,iter}$ is the maximal absolute difference observed between the objective values of two consecutive solutions up to iteration iter, then the respective exchange is penalized by

$$x \cdot \Delta_{max,iter} \cdot fr_e / fr_{max}$$

Reordering of each route

- If a new overall best solution is found, **a reordering of its different routes is evoked**. For this purpose, the I1 insertion procedure proposed by Solomon is applied
- Therefore, **all customers of the considered route are erased and reinserted** one by one starting with the one farthest from the depot as the seed customer generating an initial one customer cycle
- The **procedure is executed R times** while the best one is chosen if it improves the solution previously considered; otherwise the process tracks back to the old constellation
- This instrument can be seen as a **specific form of intensification** after finding an improved constellation

Post-processing

- At the end of the procedure, a specific heuristic, originally constructed for the TSP (respecting time windows) is applied on each route of the found solution (Gendreau et al. (1998))
- This method is an adaption of the GENIUS heuristic originally devised for the TSP (Gendreau et al. (1992))
- This heuristic only slightly improved the total distance of the considered instances (10 solutions in Solomon's test were improved by at most 1% (with one exception), but it consumes only a few seconds computational time

Computational results

- Test problems
  - Standard benchmark of Solomon (100-customer problems)
  - Note that this is a benchmark set for the VRPHTW (Solomon (1987))
  - 56 instances, clustered
  - Locations are distributed within a $[0,100]^2$ square
  - Experiments performed on a SUN Sparc 10 workstation (50 Mhz)
- New objective: Hierarchical objective function
  - First objective: Minimization of the number of routes
  - Second objective: Minimization of the total travel distance (costs)

Parameter setting for the Tabu Search

Parameter	Setting
Objective function	α=100
Initial solutions	I=20
Adaptive Memory	M=30
D&R/Number of routes	W=6/U=8
Iterations	A=30, B=3, i.e. 30,40,50,60,70, and 80 iterations for DR=1,2,3,4,5, and 6 respectively Total: 330
Neighborhood	L=5 or L=7
Tabu list length	T=100,000
Tabu tenure	Number of iterations/2
Reordering number	R=20

Results – Overall

Experiment Class	Chiang& Russel 1993	Potvin& Bengio 1996	Thangiah et al. 1994	Rochat& Taillard 1995	New Approach 1997
R1	12,42 1289,95	12,58 1296	12,33 1238	12,25 **1208,50**	**12,17** 1209,35 (2nd)
C1	*10* 885,86	*10* 838,01	*10* 832	*10* **828,38**	*10* **828,38**
RC1	12,38 1455,82	12,13 1446,20	12 **1284**	11,88 1377,39	**11,5** 1389,22 (3rd)
R2	2,91 1135,14	3 1117,7	3 1005	2,91 **961,72**	**2,82** 980,27 (2nd)
C2	*3* 658,88	*3* 589,93	*3* 650	*3* **589,86**	*3* **589,86**
RC2	*3,38* 1361,14	*3,38* 1360,57	*3,38* 1229	*3,38* 1119,59	*3,38* **1117,44**

Results – Overall

- Fleet size *m* was set to the number of routes of the best solution reported in the literature for each problem
- Best known solutions of altogether 17 instances were improved by the new approach
- Tied 20 best known solutions
- Average results are always at the top of the list of the best known results (between 1st and 3rd position in the "quality list")

➢ Competitive solution procedure

Results – CPU-time

Experiment Class	CPU time (seconds)	Average number of tours	Average overall distance
R1	2.296 6.887 13.774	12,64 12,39 12,33	1.233,88 1.230,48 (0,28 %) 1.220,35 (1,10 %)
C1	2.926 7.315 14.630	10 10 10	830,41 828,59 (0,22 %) 828,45 (0,23 %)
RC1	1.877 5.632 11.264	12,08 12 11,9	1.404,59 1.387,01 (1,25 %) 1.381,31 (1,66 %)
R2	3.372 10.116 20.232	3 3 3	1.046,56 1.029,65 (1,62 %) 1013,35 (3,17 %)
C2	3.275 8.187 16.375	3 3 3	592,75 591,14 (0,27 %) 590,91 (0,31 %)
RC2	1.933 5.798 11.596	3,38 3,38 3,38	1248,34 1220,28 (2,25 %) 1198,63 (3,98 %)

Results – CPU-time

- Significant improvements of the average solution quality (up to 4 percent!) even after long lasting computations
- That can be mainly explained by the flexible neighborhood defined by the complex CROSS operation

Results – Neighborhood

Neighborhood	CPU time	Average number of routes	Average total distance
CROSS	2.296	12,64	1.233,88
	6.887	12,39	1.230,48
	13.774	12,33	1.220,35
2-opt*	1.259	12,88	1.304,03
	3.147	12,73	1.293,97
	6.294	12,58	1.288,92

- A flexible neighborhood is necessary to yield a high solution quality which improves best solutions in benchmarks
- Unfortunately, the more elaborated the neighborhood structures are, the more they consume significantly higher computational times

6.4.2.3 The parallel Tabu Search version

- In order to improve its solution quality, the Tabu Search procedure has been parallelized on a workstation cluster using PVM library for communication
- Doing so, simultaneous examination in different parts of the solution space is initiated

Two main kinds of parallel systems

- Parallel Computers:
  - Specific system architectures designed for high-end computations with a fine granularity
  - A large number of CPUs are connected by a specialized network
  - We again distinguish systems with additional shared memory from systems where each CPU can access its own memory only while each interaction between the processors results form message passing in the network
  - Ratio of communication and internal calculation is much better than the one of distributed systems. Especially initiating new messages does not cause a comparable amount of time in such kind of parallel systems, i.e., the latency is much better

Two main kinds of parallel systems

- But compared with internal calculations, communication operations still cause significantly higher effort, i.e., an efficient parallel program has to prevent unnecessary communication routines
- Parallel systems are extremely expensive and their use requires expert knowledge
- The systems obsolesced very quickly

➢ In industry, significant use of this kind of specific parallel systems seems to be not very likely

Distributed systems

- System architectures designed for normal office applications and communications, known as PC-networks
- A large number of PCs connected by a Local Area Network leading to a more loosely connected system
- Compared to parallel systems considered before, the ratio of communication and internal calculation is much worse. Especially, initiating new messages is extremely costly, i.e., the latency is very high
- Therefore, compared to internal calculations, communication operations cause extreme effort, i.e., an efficient parallel program can achieve substantial speedups only if an appropriate design of the procedure is elaborated
- PC-network already exists in many companies today, while in most cases its total performance is left unused
- The system stays up to date by exchanging the PCs at the working places

➢ In industry, modern software systems should integrate distributed routines for hard-to-solve problems by starting distributed background processes

Consequences

- **Dynamic work balancing**
  - Solution approach is a background process working simultaneously to normal office application
  - The performance of each processor changes dynamically depending on the complexity of the office application
  - Therefore, in order to guarantee an efficient execution, an applied algorithm must distribute the current work according to the existing performance of the computers in the considered network
- **Specific program design**
  - Since the ratio between communication and internal calculation is much more unfavorable than known from parallel systems, a specific program design is necessary, which prevents frequently occurring communication periods and reduces the proportion of small messages significantly
  - In order to do so, appropriate theoretical models mapping the behavior of distributed systems accurately are used for analyzing designed procedures
  - Coarse grained program structures are necessary to use these kind of systems efficiently

Parallel scheme

- **Master – Slave approach**
  - **Master**
    - Manages the adaptive memory and generates solutions by using its current routes
    - Transfers these solutions to the independently working slaves
  - **Slaves**
    - Apply the Tabu Search procedure on the received solutions
    - Send back newly generated constellations
- By doing so, the approach executes several distinct search paths in parallel

Process scheme

- In the parallel algorithm, the following independent processes work together
  - Manager process (1):
    - Master job defined above
    - Managing adaptive memory, creates starting solutions for the decomposition procedure
  - Decomposition processes (S):
    Decompose the problem into D subproblems
  - Dispatcher (1):
    Dispatches the work equally among the used processors
  - Tabu processes (P):
    Apply Tabu Search to the generated subproblems
  - Initialization processes (I):
    Generate the initial solutions

Process scheme

- Allocation scheme processes – processors:
  - **1 dedicated processor executes the manager**, the **decomposition,** and the **dispatcher process**
  - Each **Tabu Search process is executed by a distinct processor**, which is additionally used for executing a predefined number of **initialization processes** before, i.e., $I=k \cdot P$ with a natural number k
- Therefore, we use **altogether P+1 processors in the network**
  - P processors work on independent parallel searching processes
  - One additional process is responsible for some managing tasks

Computational results

- **Test problems**
  - Standard benchmark of Solomon (100-customer problems)
  - 56 instances, clustered in the classes R1, R2, C1, C2, RC1 and RC2
  - Locations are distributed within a $[0,100]^2$ square while the respective travel times are equivalent to the Euclidean distances in the problems
  - Experiments performed on a network of altogether 17 SUN Sparc 5 workstation
  - The network was entirely allocated to the planning process wherefore the measured results were not distorted by background applications

Parameter setting for the Tabu Search

Parameter	Setting
Objective function	α=100
Initial solutions	I=20
Adaptive Memory	M=30
D&R/Number of routes	W=6/U=8
Iterations	A=30, B=3, i.e. 30,40,50,60,70 and 80 iterations for DR=1,2,3,4,5 and 6 respectively Total: 330
Neighborhood	L=5 or L=7
Tabu list length	T=100
Tabu tenure	Number of iterations/2
Reordering number	R=20

Measuring efficiency in parallel algorithms

- Parallel processing can help in order to…
  - yield improved results in the same period of time or to
  - compute identical solutions much faster
- In our application, we can compute the efficiency of the parallel algorithm by the following formula

$$\frac{\text{sequential time}}{\text{parallel time} \times \text{number of used processors}}$$

The master process

- The master or controller executing the manager, the dispatcher and the decomposition processes **is not used sufficiently**
- Therefore, the efficiency measurement is only calculated according to the slave processes

Computational results

Number of processors	Number of search threads per node	Number of calls to the adaptive memory		
		20	40	80
1	1	2,645 seconds	4,031	6,796
5 Max 80 %	2	877 60,3 %	1,324 60,9 %	2,223 61,1 %
9 Max 88,8 %	4	459 64 %	691 64,8 %	1,146 65,9 %
17 Max 94,1 %	8	239 65 %	363 65,3 %	587 68,1 %

Observations

- The more work to do, the better the efficiency becomes
- In case of 80 calls to the adaptive memory, an acceptable speedup is yielded with up to 17 processors

Computational results - efficiency

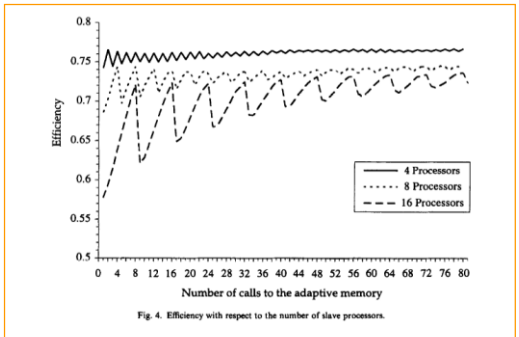


Fig. 4. Efficiency with respect to the number of slave processors.

Observations

- The **oscillating behavior** of the efficiency curves can be easily explained by the fact that the threads are somewhat synchronized
  - The S Decomposition processes reconstruct their solutions
  - and afterwards send them to the manager almost simultaneously
- As time passes, the **threads become less and less synchronized**; as a consequence, this reduces the unused waiting times for each process receiving a new subproblem from the controller
- This explains why a decrease in the amplitude of the patterns and a slight increase with time is observed

Observations

- If we compute the efficiency according to all used processors in the network, we get reduced values for all network sizes (now around 65 percent instead of 75 as measured before). But now, the largest system sizes yield the highest efficiency rates for long computation times
- This is a clear indication of the under-utilization of the controller, i.e., the master processor during the computation process
- Possible work around
  - Moving the control process, which is currently resident on the master, to one of the slaves
  - Note that in order to get a considerable improvement, these moved processes need a high priority to avoid additional bottlenecks during the computation

Literature for Section 6

- Albrecher, H.; Burkhard, R. E.; Çela, E.: An asymptotical study of combinatorial optimization problems by means of statistical mechanics. Journal of Computational and Applied Mathematics 186 (1), 148-162, 2006.
- Badeau, P.; Guertin, F.; Gendreau, M.; Potvin, J.-Y.; Taillard, E.: A Parallel Tabu Search for the Vehicle Routing Problem with Time Windows. Transportation Research C, Vol.5, No.2, pp.109-122, 1997.
- Battiti, R.; Tecchiolli, G.: The Reactive Tabu Search. ORSA Journal on Computing 6, 126-140, 1994.
- Bierwirth, C.; Mattfeld, D.C.: Production scheduling and rescheduling with genetic scheduling. Evolutionary Computation 7(1), 1-18, 1999.
- Bock, S: Solving Complex QAP-Instances by a PC-LAN. In: Günther, H.-O.; Mattfeld, D.C.; Suhl, L.: Supply Chain Management und Logistik: Optimierung, Simulation, Decision Support (German), 531-552, Physica Verlag, Heidelberg, 2005.
- Bölte, A.; Thonemann, U.: Optimizing simulated annealing schedules with genetic programming. European Journal of Operational Research 92, 402-416, 1996.
- Brucker, P.: Scheduling algorithms. Fifth edition. Springer, Berlin, 2007.

Literature for Section 6

- Brucker, P.; Knust, S.: Complex Scheduling. Springer, Berlin, 2006.
- Connolly, D.T.: An improved annealing scheme for the QAP. European Journal of Operational Research 46, 93-100, 1990.
- Gendreau, M.; Hertz, A.; Laporte, G.: New insertion and postoptimization procedures for the Traveling Salesman Problem. Operations Research, Vol 40, No. 6, 1086-1094, 1992.
- Gendreau, M.; Hertz, A.; Laporte, G.; Stan, M.: A Generalized Insertion Heuristic for the Traveling Salesman Problem with Time Windows. Operations Research Vol. 46, No. 3, 330-335, 1998.
- Gendreau, M.; Guertin, F.; Potvin, J.-Y.; Taillard, Ê.: Parallel Tabu Search for Real-Time Vehicle Routing and Dispatching. Transportation Science, Vol.33, No.4, 1999.
- Glover, F. 1990. Tabu Search Part 1 and 2. ORSA Journal on Computing 1 (1989) pp. 190-206 and Vol. 2 (1990) pp.4-32.
- Glover, F., Laguna, M. 1997. Tabu Search. Kluwer Acad. Publishers.
- Müller-Merbach, H.: Optimale Reihenfolgen. Springer Verlag, Berlin, Heidelberg, New York, 1970.
- Nowicki, E.; Smutnicki, C.: An advanced Tabu Search Algorithm for the Job Shop Problem. Journal of Scheduling 8, 145-159, 2005.

Literature for Section 6

- Nowicki, E.; Smutnicki, C.: A fast Taboo Search Algorithm for the Job Shop Problem. Management Science 42, 797-813, 1996.
- Or. I. Traveling Salesman-Type Combinatorial Problems and Their Relation to the Logistics of Regional Blood Banking. Ph. D. Thesis, Northwestern University, Evanstonm IL, 1976.
- Pinedo, M.: Scheduling: Theory, Algorithms, and Systems. Third edition. Springer, Berlin, 2008.
- Potvin J-Y, Rousseau J-M. An exchange heuristic for routing problems with time windows. Journal of the Operational Research Society Vol. 46, pp.1433-46, 1995.
- Skorin-Kapow, J.: Tabu Search applied to the Quadratic Assignment Problem. ORSA Journal on Computing 2, 33-45, 1990.
- Skorin-Kapow, J.: Extensions of a Tabu Search adaptation to the Quadratic Assignment Problem. Computers & Operations Research 21(8), 855-865, 1994.
- Solomon, M.M.: Algorithms for the Vehicle Routing and Scheduling Problems with Time Window Constraints. Operations Research 35 254-265, 1987

Literature for Section 6

- Taillard, É.; Badeau, P.; Gendreau, M.; Guertin, F.; Potvin, J.-Y.: A Tabu Search Heuristic for the Vehicle Routing Problem with Soft Time Windows. Transportation Science Vol.31, No.2, pp.170-186, 1997.
- Taillard, É.D.: Robust Taboo Search for the Quadratic Assignment Problem. Parallel Computing 17, 443-455, 1991.
- Taillard, É.D.: Comparison of iterative searches for the Quadratic Assignment Problem. Location Science 3 (2), 87-105, 1995.