2

A Modelling and Optimization Framework for Real-World Vehicle Routing Problems

Tonči Carić[1], Ante Galić[1], Juraj Fosin[1], Hrvoje Gold[1] and Andreas Reinholz[2]
[1]*Faculty of Transport and Traffic Sciences, University of Zagreb*
[2]*TU Dortmund*
[1]*Croatia,*
[2]*Germany*

1. Introduction

The globalisation of the economy leads to a rapidly growing exchange of goods on our planet. Limited commodities and transportation resources, high planning complexity and the increasing cost pressure through the strong competition between logistics service providers make it essential to use computer-aided systems for the planning of the transports. An important subtask in this context is the operational planning of trucks or other specialized transportation vehicles. These optimization tasks are called Vehicle Routing Problems (VRP). Over 1000 papers about a huge variety of Vehicle Routing Problems indicate the practical and theoretical importance of this NP-hard optimization problem. Therefore, many specific solvers for different Vehicle Routing Problems can be found in the literature. The drawback is that most of these solvers are high specialized and inflexible and it needs a lot of effort to adapt them to modified problems. Additionally, most real world problems are often much more complex than the idealized problems out of literature and they also change over time. To face this issue, we present an integrated modelling and optimization framework for solving complex and practical relevant Vehicle Routing Problems. The modular structure of the framework, a script based modelling language, a library of VRP related algorithms and a graphical user interface give the user both reusable components and high flexibility for rapid prototyping of complex Vehicle Routing Problems.

1.1 Vehicle routing problem

The problem of finding optimal routes for groups of vehicles, the Vehicle Routing Problem (VRP), belongs to the class of NP-hard combinatorial problems. The fundamental objectives are to find the minimal number of vehicles, the minimal travel time or the minimal costs of the travelled routes. In practice the basic formulation of the VRP problem is augmented by constraints such as e.g. vehicle capacity or time interval in which each customer has to be served, revealing the Capacitated Vehicle Routing Problem (CVRP) and the Vehicle Routing Problem with Time Windows (VRPTW) respectively. The real-world problems mostly encompass the capacity and time constraints. For solving VRPTW problems, a large variety of algorithms has been proposed. Older methods developed for the VRPTW are described in

Source: Vehicle Routing Problem, Book edited by: Tonci Caric and Hrvoje Gold, ISBN 978-953-7619-09-1, pp. 142, September 2008, I-Tech, Vienna, Austria

the survey (Cordeau et al., 2002) and (Laporte, 1992). Most of the new methods tested on Solomon's benchmarks are comprised in (Bräysy & Gendreau, 2005a; Bräysy & Gendreau, 2005b). The methods that applied the two-phase approach for solving VRPTW are found to be the most successful (Bräysy & Dullaert, 2003). During the first phase the constructive heuristic algorithm is used to generate a feasible initial solution. In the second phase an iterative improvement heuristics is applied to the initial solution. The mechanism for escaping the local optima is often implemented in the second phase, too.
For the real-world application that solves VRP it is essential to perform a fast selection of methods (constructive heuristics, neighbourhood operators and escaping mechanisms) which produce the desired improvement of the objective function. Commercial VRP applications mostly converge to self-adaptive procedures with major aim of robustness to solve the problem with minimal human intervention in algorithms tuning. On the other side tailor-made solution needs easy-to-use prototyping tool with the well defined performance measure for estimating the optimal number of restarts and iterations of the implemented algorithms. To speed up the prototyping a new VRP framework has been developed.

2. Framework

The framework consists of the Framework Scripting Language (FSL), library of VRP-related algorithms and graphical user interface which enables loading of standard benchmarks and real-world problems, Fig. 1.

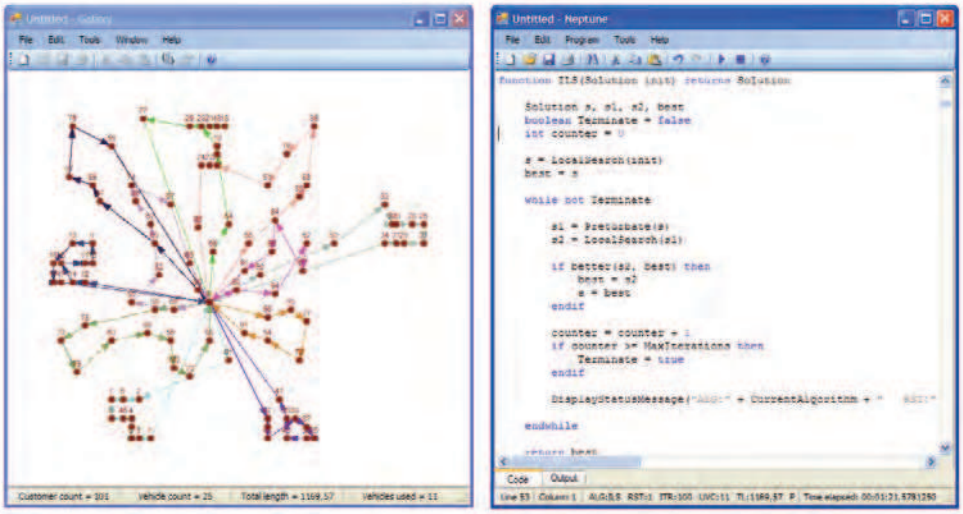


Fig. 1. VRP working environment and Framework Scripting Language

The VRP framework provides the working environment and the reusable code modules such as constructive heuristic methods and common improvement operators. This programming environment leaves more time for the developer to focus on the implementation and testing of the new ideas. The whole library is written in the framework language so that the included programming modules can be easily adapted to the needed functionality.

2.1 Framework scripting language

Like many other programming languages, the Framework Scripting Language (FSL) has a core set of basic data types (boolean, int, double, string) and program control statements (if, for, while, repeat). The FSL structure is the improved version of the previously developed VRP solving oriented language (Galić et al., 2006a). The VRP problem is described with *Problem* data structure which stores all customers in the list *Customers* and all vehicles in the list *Vehicles*. Each *Customer* and *Vehicle* is instances of the corresponding data type whose attributes describe in detail the concrete problem being solved. The VRP-related data types and the corresponding attributes and methods which are available for use in FSL are depicted in Fig. 2. Every solution of the VRP problem can be stored in an instance of *Solution* data type which holds the routes of vehicles (visiting order of customers) and other information which describe specific solution such as the number of used vehicles, total travelled distance and total estimated time which is the sum of the driving, serving and waiting time for each used vehicle. Using the *SetCurrentSolution* method it is possible to switch between different solutions of the same VRP problem.

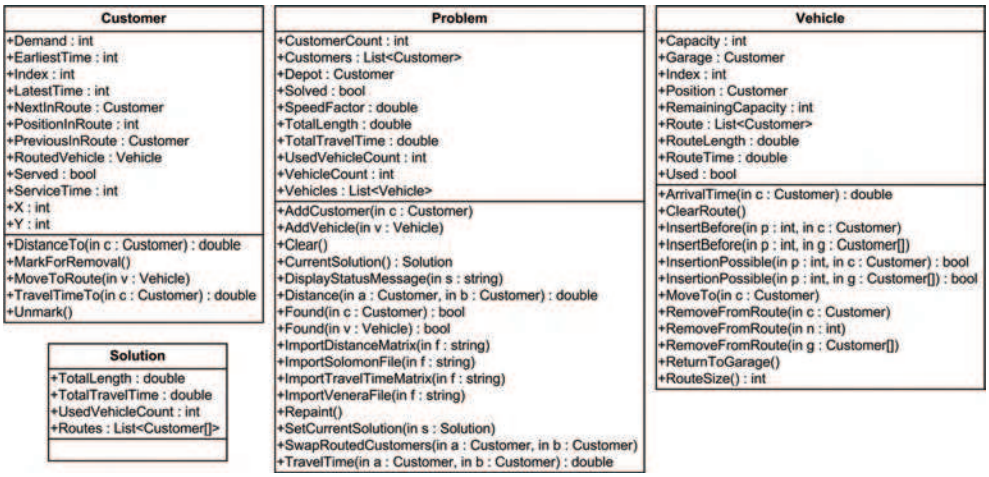


Fig. 2. VRP Framework Scripting Language data types, attributes and methods

2.2 Library

The library includes a variety of constructive heuristics, e.g. Clark and Wright, Solomon Insertion I1, Coefficient Weighted Distance Time Heuristic and neighbourhood operators, e.g. relocate, exchange, cross exchange. The included examples of procedures for escaping the local optima like Simulated Annealing, Iterated Local Search and Variable Neighbourhood Search present the proposed programming style of modules 'gluing'. VRP solver produced by this prototyping tool is an algorithm which can be composed by choosing and tuning modules from a library and that guides the search through local optima to achieve a better solution.

2.2.1 Constructive heuristics

The first step of the heuristic VRP solving is the construction of a feasible initial solution. In the lucky case when handmade solution already exists, it can be considered as a substitution

for the constructive heuristics. Generally, constructive heuristics follows the idea that customers are selected on some cost minimization criterion and routes are constructed matching capacity and time constrains. Methods with sequential approach construct one route at a time, while parallel methods build several routes simultaneously. Some constructive methods are two-phase methods and can be divided into two classes: cluster-first, route-second methods and route-first, cluster-second methods. In the first case, customers are first organized into feasible clusters, and a vehicle route is constructed for each of them. In the second case, a tour is first built on all customers and then segmented into feasible vehicle routes.

2.2.1.1 Nearest Neighbour Heuristic for CVRP

The Nearest Neighbour Heuristic (NNH) is a constructive method for generating initial feasible solution for CVRP with the simple idea of inserting the nearest neighbour of the last inserted customer in the route. The first inserted customer on the route can be selected randomly or with some arbitrary criteria like the farthest distance customer from the depot. From this seed route, every other customer is inserted by the criteria of the nearest neighbour from the last inserted customer until the capacity of the vehicle is exhausted according to the definition of the CVRP problem where every customer has its own demand for the delivery or pick up. This method is derived from Travelling Salesman Problem (TSP) heuristic approach (Flood, 1956).

2.2.1.2 Nearest Addition Heuristic for CVRP

The Nearest Addition Heuristic (NAH) is an extended version of NNH where one of the unserved customers is selected for insertion and added to the existing route between two already visited neighbours. The total price of insertion has to be the minimal value that is calculated by adding two new distances produced by linking of the unvisited customer with neighbours and by subtracting the distance between the visited neighbour's customers in the selected route.

2.2.1.3 Sweep Heuristic for CVRP

One of the most known two-phase constructive methods for CVRP is the sweep algorithm (Gillet & Miller, 1974). This is a two-phase algorithm that belongs to the cluster-first, route-second methods. In the first phase the algorithm decomposes the CVRP problem by clustering customers in m-TSP problems. The customer clustering is conducted by two criteria. The positions of all customers are transformed in polar coordinates with the depot in the origin of the coordinate system. The first criterion for grouping customers is the minimal angle. The second criterion matches the capacity of the vehicle which is assigned to the cluster, so that the total demands of all the selected customers has to be less than or equal to the capacity of the vehicle. The first and the second criteria are combined so that the assignment of customers to groups is performed by increasing the angular coordinate from 0 to the value where capacity of the assigned vehicle for that cluster is exhausted. The last step optimizes each vehicle route (cluster) separately by solving the corresponding TSP.

2.2.1.4 Clark and Wright Heuristic for CVRP

This method is one of the first originally developed heuristics for CVRP and it is frequently used. The algorithm starts from the initial solution where each route has only one customer and a corresponding vehicle. At the start, the number of vehicles is equal to the number of customers. Every new iteration should reduce the number of vehicles unifying two routes that give maximal savings, e.g. reduction of overall distance or time. There are two variants of algorithm: one with sequential and other with parallel construction of routes. The parallel version yields better results (Clarke & Wright, 1964).

2.2.1.5 Solomon's Sequential Insertion Heuristic I1 for CVRPTW

The seed customer for a new route can be set on various criteria (e.g. the farthest unrouted customer, unrouted customer with the earliest deadline or unrouted customer with the biggest demand). For each unrouted customer the feasible insertion place in the emerging route with its minimal insertion cost as a weighted average of additional distance and time is computed first. The next step is to select a customer for whom the cost difference between insertion in a new route and in the emerging route is the largest. The selected customer is then inserted in the route and the new calculation and selection is repeated until the time or capacity resource is exhausted. New resources are generated with new route/vehicles. It is not trivial to find the suitable weighted average for real-world problems. A good starting point for the tuning algorithm can be found in the original paper (Solomon, 1987).

2.2.1.6 Coefficient Weighted Distance Time Heuristics for CVRPTW

Based on the assignment of weights to the closing part of a time windows and distances to the serving places, the Coefficient Weighted Distance Time Heuristics (CWDTH) has been developed (Galić et al., 2006b; Carić et al., 2007). In each iteration the algorithm simultaneously searches for the customer with the soonest closing time of requested delivery and minimum distance from the current vehicle position. The route is designed starting with one vehicle. In each subsequent iteration the customer who best matches the given criteria is served. When the vehicle has used all of the available capacity that can be utilized regarding the amount of demands, it returns to the depot. A new vehicle is engaged and the described process is repeated. At the moment when all customers have been served, the algorithm stops. Automatic parameter adjusting is implemented for the weighting of distance over delivery closing time.

2.2.2 Local search

The local search starts from the initial solution (e.g. provided by constructive heuristic method) and subsequently moves from the present solution to a neighbouring solution in the search space where each solution has only a relatively small number of feasible neighbour solutions and each of the moves is determined by neighbourhood's operators.

The library includes two groups of operators. Operators from the first group move one or more customers from one position in the route to another position in the same route and are called Intra Route operators, Fig. 3.

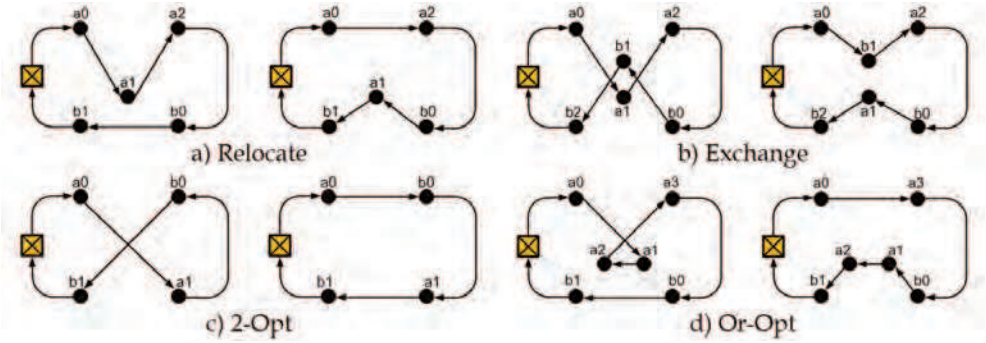


Fig. 3. Intra Route operators for CVRPTW

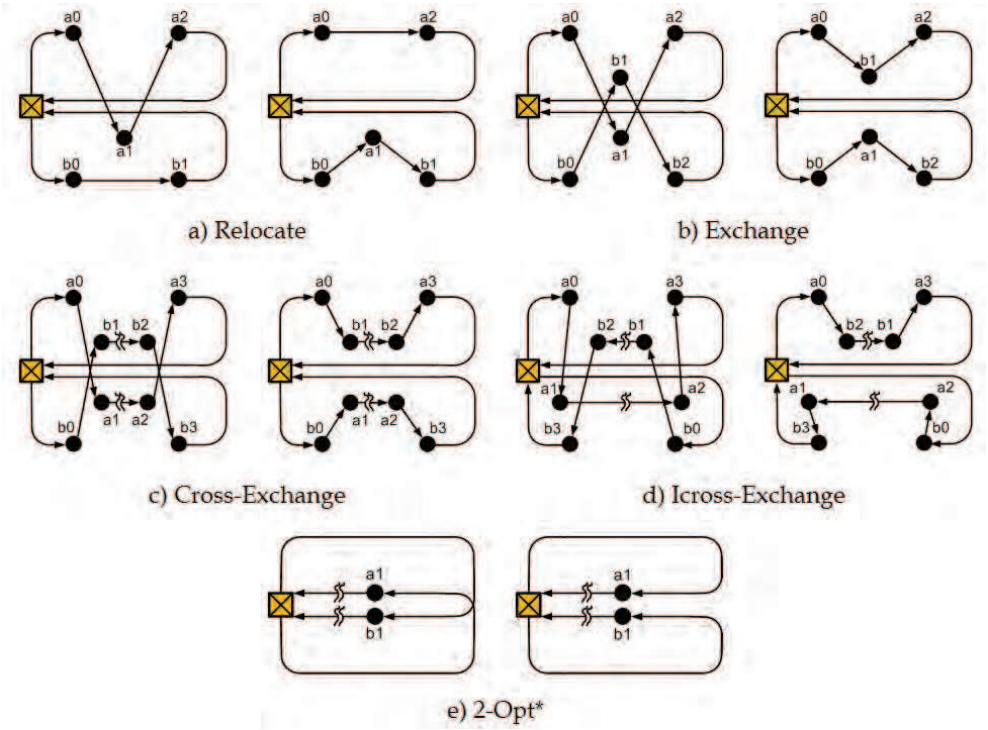


Fig. 4. Inter Route operators for CVRPTW

They are used for the reduction of the overall distance. The other group, called Inter Route operators, work with two routes, Fig. 4. They are used to reduce overall distance but in some cases they can reduce the number of vehicles as well.

2.2.2.1 Intra Route Relocate for CVRPTW

To be served in the new order between b0 and b1 the customer a1 is relocated from the original position between a0 and a2, (see Fig. 3a). Relocation is performed only when the saving (reduction of length of the route) is positive. The saving is calculated by maximizing the result of subtraction *x*-*y* where *x* is derived as the result of three arc deleting operations (a0, a1), (a1, a2) and (b0, b1) and *y* is derived as the result of three arc adding operations of (a0, a2), (b0, a1) and (a1, b1).

2.2.2.2 Intra Route Exchange for CVRPTW

Intra exchange operator swaps the position of two customers a1 and b1 (see Fig. 3b). To be served in the new order between a0 and a2 customer b1 is relocated from the original position between b0 and b2. Also, customer a1 is relocated from the original position between a0 and a2 to the new position between b0 and b2. The exchange is performed only when the saving (reduction of length of the route) is positive. The saving is calculated by maximizing the result of subtraction *x*-*y* as in the Intra Relocate operator. This operator could be considered as execution of two relocate operators, but sometime because of hard time windows the intermediate solution is not feasible.

2.2.2.3 Intra Route 2-Opt for CVRPTW

Intra route 2-Opt operator transforms the intersection of arcs if savings exist after we change the direction of the arcs between a1 and b0 and delete and add the appropriate arcs (see Fig. 3c).

2.2.2.4 Intra Or-Opt for CVRPTW

If savings exist, the intra route operator Or-Opt transforms the intersection of arcs with reordering customers on a route (see Fig. 3d). This operator is practical because it is very fast. The alternative slower scenario is two relocate operator execution.

2.2.2.5 Inter Route Relocate for CVRPTW

The inter route Relocate operator moves customer a1 from one route to another between b0 and b1 if savings exist (see Fig. 4a).

2.2.2.6 Inter Route Exchange for CVRPTW

The Exchange operator swaps two customers a1 and b1 from two different routes if savings exist (see Fig. 4b).

2.2.2.7 Inter Route Cross-Exchange for CVRPTW

The Cross-Exchange operator swaps two groups of customers from one route to another (see Fig. 4c). The groups consist of one up to maximally five customers. Bigger groups are inefficient mainly because of slow execution time. To prevent neighbourhoods from being interlaced (exchange and cross-exchange) only one group a1-a2 or b1-b2 can have only one customer in the group. In that case a1=a2 or b1=b2.

2.2.2.8 Inter Route Icross-Exchange for CVRPTW

The Icross-Exchange operator swaps two groups of customers the same way as Cross-Exchange but reverse in order of customers in both groups (see Fig. 4d). Further extension of Icross and Cross operator can be to leave the order of one group and to reverse the order of another.

2.2.2.9 Inter Route 2-Opt for CVRPTW*

Inter Route 2-Opt* can be considered like Cross-Exchange where b2 and a2 customers are depot (see Fig. 4e).

2.2.3 Escaping mechanism

By applying only the neighbourhood's operators in local search in most of the cases leads the optimization to the local optima where operators cannot yield better solutions any more. To escape from the local optima the escaping mechanisms such as Simulated Annealing, Iterated Local Search and Variable Neighbourhood Search are implemented and their scripts in FSL can be found in the Appendix.

2.2.3.1 Simulated Annealing

Simulated Annealing (SA) is a stochastic relaxation technique that finds its origin in statistical mechanics (Kirkpatrick et al., 1983; Cerny, 1985; Metropolis et al., 1953). Simulated Annealing uses stochastic approach to guide the search. The method allows the search to continue in the direction of the neighbour even if the cost function gives inferior results in that direction. The starting solution is obtained by constructive heuristics, described in section 2.2.1

2.2.3.2 Iterated local search

The local search process is started by selecting an initial candidate solution and then proceeds by iteratively moving from one candidate solution to the neighbouring candidate solution, where the decision on each search step is based on a limited amount of local information only. In Stochastic Local Search (SLS) algorithms, these decisions as well as the search initialization can be randomized (Hoos & Stützle, 2005). Generally, in the Iterated Local Search (ILS) two types of SLS steps are used (Laurenço & Serra, 2002). One step for reaching the local optima as efficiently as possible and the other step for efficiently escaping local optima.

2.2.3.3 Variable Neighbourhood Search

Another way to escape local optima is the idea that one solution which is a local optima for one neighbourhood generated by one operator need not be a local optima for another neighbourhood generated by some other operator. The procedure of Variable Neighbourhood Search (VNS) approach is to change the neighbourhood (operator) whenever local optima is reached. The starting neighbourhood is usually generated by the simplest operator in the pool of available operators. When currently selected operator does not produce improvement the next operator is selected and process continues. If any of the available operators produce an improvement, whole process starts again with first (simplest) operator from the pool. In case when neither one of the operators (including last one) produces improvement the shake move (perturbation) is executed to escape local optima. A new cycle starts again from first i.e. simplest operator. Function DoOperator (see Appendix) is called from VNS function to execute desirable operator by passing integer variable that represents index of operator.

2.3 Examples of using the framework scripting language

2.3.1 Solving the travelling salesman problem

In order to demonstrate the scope and the usage of FSL, a simple example of solving a TSP problem (one-vehicle VRP) by two algorithms from library is described, Fig. 5. The initial solution is calculated by the nearest neighbour constructive heuristic, described in section 2.2.1.1, and further improvements are done by simplified Intra Route Relocate operator, described in 2.2.2.1. In line 3, method *Clear()* deletes all the previous routes (if there are any) and prepares the problem for solving from scratch. In line 4, the initial solution construction is obtained by *TspNearestNeighbour* function call. From line 5 to line 7, the current solution is improved in a loop which breaks after the IntraRelocate operator has yielded no improvement. Inside of the *TspNearestNeighbour* method, the *select* statement in line 11 is used for finding one customer among all *Customers* that satisfies two conditions (customer is not *Depot* and not served at the moment) and minimizes the objective function. The result of the search is stored in the variable of *Customer* data type called *nearest*.

The objective function defined in line 14 represents the distance between the current position of vehicle *v* and candidate customer *nearest*. The objective function is calculated only for those *Customers* for which both conditions have been fulfilled and the selecting process is ended with the *Customer* which has the lowest value of objective function. In other words, this query (lines 11-15) selects the nearest customer to the current position of the vehicle which is not depot and which is not being served at the moment. Despite similar syntax the FSL statement *select* should not be confused with SQL SELECT statement. Statements *v.MoveTo(nearest)* and *v.ReturnToGarage()* are examples of methods that have

impact on the current solution of the active problem. By moving vehicle v to the nearest customer or its garage (default is depot), the state of the active problem solution will be changed. For example, by moving vehicle v to customer A, the vehicle route will be updated and the attribute of its position will gain value of A.

```
Vehicle truck = Vehicles[0]
boolean success
Clear()
TspNearestNeighbour(truck)
repeat
   success = IntraRelocate(truck)
until not success
function TspNearestNeighbour(Vehicle v) returns nothing
   Customer nearest
   while true
      select Customer nearest from Customers where
         : nearest != v.Garage
         : nearest.Served = false
         minimize Distance(v.Position, nearest)
      endselect
      if not Found(nearest) then
         break
      endif
      v.MoveTo(nearest)
   endwhile
   v.ReturnToGarage()
endfunction
function IntraRelocate(Vehicle v) returns boolean
   double x, y, improvement
   Customer a0, a1, a2, b0, b1
   select Customer a1 from v.Route, Customer b1 from v.Route where
         : a1.Index != Depot.Index
         : b1.Index != Depot.Index
         : abs (a1.PositionInRoute - b1.PositionInRoute) > 1
         a0 = v.Route[a1.PositionInRoute-1]
         a2 = v.Route[a1.PositionInRoute+1]
         b0 = v.Route[b1.PositionInRoute-1]
         x = (a0.DistanceTo(a1) + a1.DistanceTo(a2) +
         b0.DistanceTo(b1))
         y = (b0.DistanceTo(a1) + a1.DistanceTo(b1) +
         a0.DistanceTo(a2))
         improvement = x - y
         : improvement > 0.000000001
         maximize improvement
   endselect
   if not Found(a1) then
         return false
   endif
   v.RemoveFromRoute(a1)
   v.InsertBefore(b1.PositionInRoute, a1)
   return true
endfunction
```

Fig. 5. Complete script of simple Travelling Salesman Problem solved in Framework Scripting Language

The statement *select* from line 26 to 38 execute a selective search for two customers a1 and b1 from the route of vehicle *v* with the aim of maximizing the objective function defined in line 37. Lines between *select* and *endselect* labelled by colon are search constraints. Unlabelled lines are regular statements which are executed for each iteration of search to update the variables used as part constraints or objective function. Constraints in lines 27 and 28 request that customers, a1 and b1, cannot be equal to depot. Constraint in line 29 does not allow the case when customers a1 and b1 are two neighbours on the route. The final constraint, defined in line 36 assures that every improvement has to be positive and it resolves the problem regarding acceptable error in comparison to the floating point numbers. The objective function is calculated only for those values of the required variables at which all the set conditions have been fulfilled. If one of the conditions is never to be satisfied, the search will be unsuccessful, and will result in non-initialized variables. In that case, after the statement *select* has been performed, the variables can be tested and the decision about the next action can be made. It should be noted that at the end of the search, the variables a1 and b1 acquire values for which the conditions are satisfied and for which the variable *improvement* yields maximal value. In other words, the results of the search are customer a1 and the position located before customer b1 on the same route for which we obtain maximal saving by performing the relocate operation. With the statements *v.RemoveFromRoute(a1)* and *v.InsertBefore(b1.PositionInRoute, a1)* the relocation is made.

2.3.2 Operators and guiding optimization

In order to demonstrate the proposed programming style of module "gluing" a simple local search mechanism coded in FSL is presented in Fig. 6. This local search is declared as *LocalSearch* function that returns the solution. The call of *LocalSearch* function preserves the

```
1   function LocalSearch(Solution s) returns Solution
2      Solution current = CurrentSolution()
3      SetCurrentSolution(s)
4      Solution improved = s
5      while(true)
6         SetCurrentSolution(Relocate())
7         SetCurrentSolution(Exchange())
8         SetCurrentSolution(Cross())
9         SetCurrentSolution(Icross())
10        SetCurrentSolution(TwoOptInter())
11        if better(CurrentSolution(), improved) then
12           intraRoute()
13           improved = CurrentSolution()
14        else
15           break
16        endif
17     endwhile
18     SetCurrentSolution(current)
19     return improved
20  endfunction
```

Fig. 6. Example of library components gluing for Local Search used in Simulated Annealing and Iterated Local Search escaping mechanisms

current state of the problem. Before the local search is started the problem solution is stored in the variable *current*, line 2, and restored at the end of the function, line 18. The initial

solution for *LocalSearch* function is passed by argument *s*, and the result of the search is returned, line 19, by variable *improved*. The local search is defined in *while* loop, line 5-17, which is stopped when the current solution obtained by the operators is not better than the one stored in variable *improved*, line 11. In the body of the loop, line 6-10, the operators Relocate, Exchange, Cross, Icross, TwoOptInter, one by one, try to improve the solution until local optima is reached. All of these operators return the first best feasible solution from their neighbourhoods. In line 12, the result is additionally improved by intra route operators Relocate, Exchange, 2-Opt and Or-Opt.

3. Performance measure protocol

The results of most heuristics and metaheuristics for Vehicle Routing Problems depends on the initialisation (i.e. starting solution, seed solution, etc.) of the algorithm, so that multiple starts with different initialisation can lead to better solutions than a single run. Additionally, the search process of most metaheuristics is influenced by explicit or implicit stochastic decisions (i.e. starting solution, selecting a candidate solution in a neighbourhood for the next iteration, mutation operators in Evolutionary Algorithms (EA), kick moves in ILS, shake moves in VNS, etc.). Therefore, we are using a performance measure for multi-start approaches that is able to handle both deterministic and stochastic algorithms (Reinholz, 2003). This performance measure is motivated by following question: How often do we have to run an algorithm with a concrete parameter setting so that the resulting solutions are equal or better than a requested quality threshold at a requested accuracy level (i.e. 90%, 95%, and 99%). The lowest number of runs that assures these requests is called multi-start factor (*MSF*).

DEF 1:

The multi-start factor *MSF* of an algorithm *A* with concrete parameters *P*, accuracy level *AL*, quality threshold *T*, and success probability $p(T)$ is defined by

$$MSF(A, P, T, AL) := \min(k \in \mathbf{N} \text{ with } 1 - (1 - p(T))^k \geq AL)$$

The *MSF* multiplied by the average runtime of the fixed parameterized algorithm is the performance measure *PM* that has to be minimized.

DEF 2:

The performance measure *PM* of an algorithm *A* with concrete parameters *P*, average runtime *AvRT* (*A, P*), accuracy level *AL*, and quality threshold *T* is defined by

$$PM(A, P, T, AL) := MSF(A, P, T, AL) \times AvRT(A, P)$$

The estimation of the *MSF* in a statistical method is based on the fact that the success probability *p* of being better than the requested threshold quality in one run is Bernoulli-distributed. Therefore, we can use a parameterized maximum likelihood estimator to determine the success probability *p* for one run. This implies that the success probability for *k* runs (in *k* runs there is at least one successful run) is exactly $1 - (1 - p)^k$ and that the *MSF* for reaching a requested accuracy level *AL* can be easily computed using a geometrical distribution with success probability *p*. The accuracy of the estimation of *PM* and *MSF* depends on the number of runs that are used to estimate *AvRT* (*A, P*) and $p(T)$.

Two important key parameters of iterative multi-start algorithms are the number of algorithm restarts and the maximal number of iterations. The statistic method for estimating the performance measure *PM* for a requested quality threshold and accuracy level can be

used in an elegant way to determine the best combination out of these two parameters by simply computing the *PM* for the intermediate results after each iteration and identifying the iteration with the best *PM* value.
This shows again the importance for the output of intermediate results of an algorithm when making an empirical investigation. Having done *R* runs of an algorithm for *I* iterations with the output of intermediate results, then you have also the data out of *R* runs for a statistical analysis of the algorithm with stopping criteria 1, 2, … , *I*.
In this paper we have used the statistical data out of 30 runs for each algorithm and the problem instance to estimate the success rates and the average runtimes for all the stopping criteria up to 100 iterations. The statistical analysis was applied to a series of combinations out of three accuracy levels and two quality thresholds. For the accuracy levels we have used the predefined values 90%, 95%, and 99%. The quality thresholds were chosen out of the data by the following procedure: The first quality threshold *T1* was defined by the quality value that was reached by the worst out of 25% of the best runs after 100 iterations. The second quality threshold *T2* was defined by the quality value that was reached by the worst out of 10% of the best runs.

4. Computational results

4.1 Benchmark results

The efficiency of VRP solver is usually measured by cumulative result of the Solomon benchmarks (Solomon, 1987). Three different scripts for ILS, SA and VNS are tested in order to check the relevance of the proposed Framework library and language, Table 1.

	R1	R2	R3	R4	R5	R6	CM	CPU
HG	12.08 1211.67	2.82 950.72	10.00 828.45	3.00 589.96	11.50 1395.93	3.25 1135.09	408 57422	P400 3 / 1.6
BC	12.08 1209.19	2.73 963.62	10.00 828.38	3.00 589.86	11.50 11389.22	3.25 1143.70	407 57412	P933 1 / 512
PR	11.92 1212.39	2.73 957.72	10.00 828.38	3.00 589.86	11.50 1387.12	3.25 1123.49	405 57332	P3000 10 / 2.4
MBD	12.00 1208.18	2.73 954.09	10.00 828.38	3.00 589.86	11.50 1387.12	3.25 1119.70	406 56812	P800 1 / 43.8
ILS	13.08 1192.87	3.27 936.25	10.00 828.38	3.00 589.86	12.88 1371.97	4.00 1073.73	442 56353	P2000D 5 / 9
SA	13.00 1193.51	3.27 933.82	10.00 828.38	3.00 589.86	12.88 1373.05	4.00 1067.13	441 56290	P2000D 5 / 9
VNS	12.92 1200.84	3.27 951.94	10.00 832.46	3.00 598.46	12.88 1379.63	4.00 1091.96	440 56934	P2000D 5 / 9

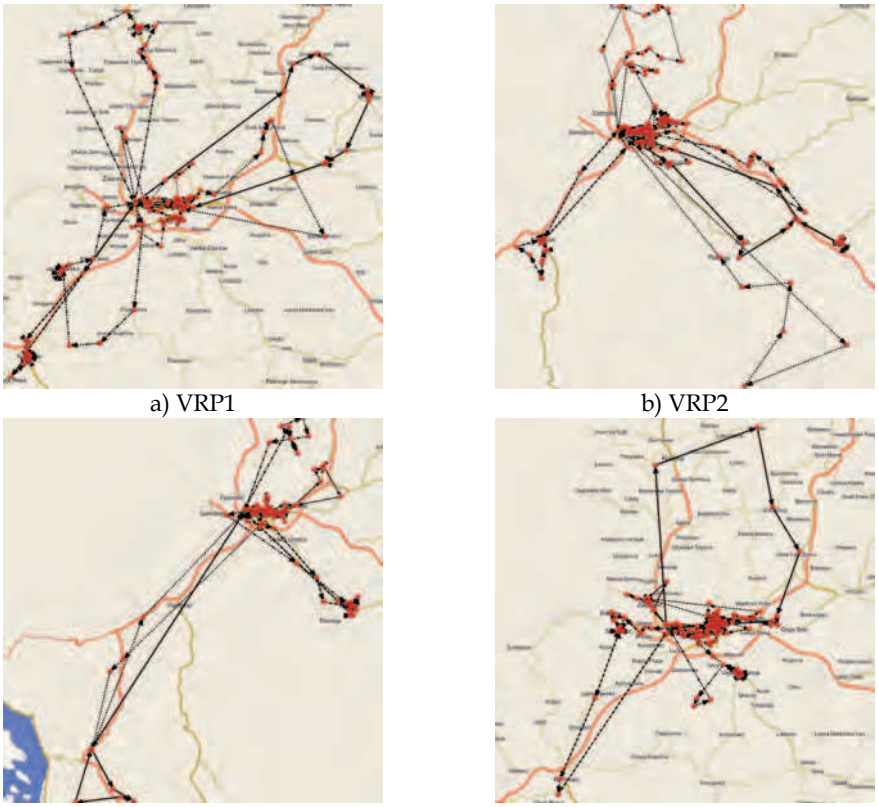
Table 1. Comparison of the results for the number of vehicles and distances obtained by ILS, SA and VNS to the best recently proposed results for Solomon's VRPTW problems. CM = cumulative values, P = Intel Pentium, D = duo, r / m = number of run(s) / minutes. HG = (Homberger & Gehring, 2005), BC = (Le Bouthillier & Crainic, 2005), PR = (Pisinger & Röpke, 2005), MBD = (Mester et al., 2007)

The modules (Solomon I1, Intra and Inter operators) of the framework are glued in two different ways (*LocalSearch* and *DoOperator)* with three different escaping mechanisms (SA,

ILS and VNS). The outline of the tested scripts is shown in the Appendix. Further development for reaching a better cumulative result of the Solomon benchmarks should focus on the methods for reducing the number of vehicles like ejection pool (Lim & Zhang, 2007) or ejection chain (Bräysy & Dullaert, 2003).

4.2 Characteristics of real-world problems

The four real-world VRPTW problems VRP1, VRP2, VRP3 and VRP4 are considered for optimization. Distribution of customers and vehicle routes are shown in Fig. 7. The set of standard VRP problems found in the literature and used to validate the performance of VRP solving algorithms use the Euclidian metric of distances. In contrast, solving the real-world VRP problems, due to the traffic rules and transport network topology requires the use of the traffic matrix. In the bidirectional traffic matrix the distances between the pairs of points stored in the transport layer of the Geographic Information System are not necessarily symmetric. This is most obvious for the routes in the urban areas while routes between urban areas are mostly symmetric. Solving the time constrained problems, as in the case of VRPTW, an additional matrix containing forecasted travel times data between each pair of customers has to be available.



a) VRP1 b) VRP2

c) VRP3 d) VRP4

Fig. 7. Maps showing customer's locations and vehicle routes

Therefore the problems are defined by two traffic matrices: the distance asymmetric look-up matrix and the related forecasted travel time matrix. The calculation of travel time matrix is based on the average velocity on a particular street or road segments. If such information is not available then the calculation is based on the rank of the road segments. In the example the road ranking follows the classification which divides them into sixteen categories. Customers for problems VRP1, VRP2 and VRP4 are located in the area of the city of Zagreb, the capital of Croatia and customers for VRP3 are located in a wider area of Zagreb. All described problems have heterogeneous fleet with two types of vehicles regarding different transport capacities (7 vehicles 2500 kg, 3 vehicles 3500 kg). The road networks are spread within big urban area, small cities and rural parts which gravitate to the capital and along inter-city highways. The number of customers varies for each problem (154, 234, 146 and 162). Overall loads per each problem are 11.1, 23.11, 14.8 and 14.6 tonnes of goods for delivery. Most of the customers have wide time widows from 7:00 a.m. till 2:00 p.m., except for a few customers who are located in the downtown area. The average customer service time is 10 minutes with little variation.

4.3 Real-world problems results

Table 2 shows which algorithm produces the best result for each of the real-world problems. Cost function was calculated by multiplication of a number of vehicles and overall distance. The quality thresholds *T1* and *T2* that are used in the comparative analysis are calculated by the procedure described in Section 3. Table 2 also shows average running time for each algorithm on each problem. That average time has important role in finding optimal number of iteration in performance measure protocol.

	ALG	*VEH*	*DIST*	*T1*	*T2*	*ARTA*		
						ILS	*SA*	*VNS*
VRP1	VNS	9	941117	8562195	8543250	4,42	4,43	2,97
VRP2	SA	10	1344551	14365330	14216120	13,68	14,16	8,87
VRP3	ILS	9	1287100	11659950	11626146	3,73	3,91	2,51
VRP4	ILS	9	847891	7711902	7655945	6,18	6,42	4,54

Table 2. Best results, thresholds and average running time of algorithms ILS, SA and VNS for real-world problems VRP1, VRP2, VRP3 and VRP 4. *ALG* = best performing algorithm, *VEH* = number of vehicles in solution, *DIST* = overall distance in meters, *T* = value of cost function, *ARTA* = average running time of algorithm in minutes

4.4 Comparative analysis of real-world problems results

The final results of the conducted experiments of performance measure protocol (see Section 3.) are shown in Table 3 and Table 4. The examination pool of results was constructed by 360 runs of the developed ILS, SA and VNS algorithms.

In order to determine which strategy needs less time, i.e. number of restarts multiplied by the number of iterations, to produce a solution below the threshold with some accuracy, each problem was solved 90 times with 30 runs of each algorithm.

Table 3 shows optimal parameters of the winning strategy for all problems. Parameters from Table 3 guarantee reaching of the threshold interval *T1* or *T2* in minimal time with 90% accuracy. For example, VRP1 needs to be restarted 4 times with halting criteria set to 89

iterations per start for ILS algorithm to reach threshold *T1* with 90% accuracy. If we increase the accuracy level the number of restarts increases.

Table 4 shows which algorithm, multi-start factor and halting number of iterations are optimal to reach threshold *T1* or *T2* with accuracy level 90%, 95% or 99.9% respectively and obtained by performance measure protocol. The thresholds are defined in such a way that all the results obtained by ILS, SA and VNS are sorted in a list where the value of the objective function on the last iteration is the number on which the sorting is done. Threshold *T1* is calculated so that 25% of the runs in the sorted list are in the *T1* threshold interval. Threshold *T2* has 10% of the best runs.

AL	**VRP1**			**VRP2**			**VRP3**			**VRP4**		
90%	***ALG***	***MSF***	***IT***	***ALG***	***MSF***	***IT***	***ALG***	***MSF***	***IT***	***ALG***	***MSF***	***IT***
T1	ILS	4	89	SA	6	99	ILS	68	5	VNS	34	6
T2	ILS	9	93	VNS	22	95	ILS	68	5	VNS	34	30

Table 3. Optimal tuning parameters for real-world benchmark problems VRP1, VRP2, VRP3 and VRP4. *AL* = accuracy level, *T* = threshold, *ALG* = algorithm, *MSF* = multi-start factor, *IT* = optimal number of iterations per each run

	VRP1						**VRP2**					
AL	***T1***			***T2***			***T1***			***T2***		
	ALG	***MSF***	***IT***	***ALG***	***MSF***	***IT***	***ALG***	***MSF***	***IT***	***ALG***	***MSF***	***IT***
90.0%	ILS	4	89	ILS	9	93	SA	6	99	VNS	22	95
95.0%	ILS	4	99	ILS	12	93	SA	10	75	SA	21	81
99.9%	ILS	10	89	ILS	26	93	SA	23	75	VNS	66	95

	VRP3						**VRP4**					
AL	***T1***			***T2***			***T1***			***T2***		
	ALG	***MSF***	***IT***	***ALG***	***MSF***	***IT***	***ALG***	***MSF***	***IT***	***ALG***	***MSF***	***IT***
90.0%	ILS	68	5	ILS	68	5	VNS	34	6	VNS	34	30
95.0%	ILS	89	5	ILS	89	5	VNS	44	6	VNS	44	30
99.9%	ILS	204	5	ILS	204	5	VNS	101	6	VNS	101	30

Table 4. Multi-start factors for real-world benchmark problems VRP1, VRP2, VRP3 and VRP4. *AL* = accuracy level, *T* = threshold, *ALG* = algorithm, *MSF* = multi-start factor, *IT* = optimal number of iterations per each run

The statistical analysis of 30 runs of each algorithm ILS, SA and VNS on each of the problems VRP1, VRP2, VRP3, VRP4 reveals that number of iterations which is 100 in the considered experiments is acceptable value for problems VRP3 and VRP4. In the case of other two problems VRP1 and VRP2 all optimal numbers of iterations are clearly grouped near the number 100 what leads us to the conclusion that the convergence of algorithms is not finished. The empirical study should be continued under the same protocol with the increased number of iterations for the problems VRP1 and VRP2. From the results in a Table 4, we can state that ILS is best performing algorithm for solving VRP3 which converges very early, but the number of restarts should be very large. With the large

numbers of restarts from 68 to 204 and only 5 iterations per run, the time duration to reach the threshold with ILS is more acceptable than time duration obtained by SA and VNS algorithms. The best performing algorithm for VRP4 problem is VNS algorithm.

5. Conclusion

For real-world application that solves the Vehicle Routing Problem it is essential to perform a fast selection of methods (constructive heuristics, neighbourhood operators and escaping mechanisms) which produce the desired improvement of the objective function. To speed up the prototyping a new flexible VRP framework has been developed and described.
The framework consists of the Framework Scripting Language (FSL), a library of coded methods and real-world benchmarks. Even new optimization ideas can use the advantages of this programming environment with tools considering commands that operate with VRP entities like moving vehicle, change customer position in routes, displaying the solving process of the current problem graphically, etc. The framework offers a set of programming tools to speed up the development, testing and tuning of heuristic algorithms.
The knowledge of solving practical problems by known methods is stored in the library which can be easily adapted for tailor-made application.
A new statistical approach for estimating the optimal number of restarts and iterations of the implemented algorithms is described and integrated in a general performance measure. This performance measure calculates for every solver the expected time that is necessary to compute solutions above a requested quality thresholds with respect to a demanded accuracy level.
The framework and the performance measure protocol are implemented on the standard benchmark and practical VRPTW problems.

6. References

Bräysy, O. & Dullaert, W. (2003). A Fast Evolutionary Metaheuristic for the Vehicle Routing Problem with Time Windows. *International Journal on Artificial Intelligence Tools*, Vol. 12, No. 2, (June 2003) pp. 153-172, ISSN 0218-2130

Bräysy, O. & Gendreau, M. (2005a). Vehicle Routing Problem with Time Windows Part I: Route construction and local search algorithms. *Transportation Science*, Vol. 39, No. 1, (February 2005) pp. 104-118, ISSN 0041-1655

Bräysy, O. & Gendreau, M. (2005b). Vehicle Routing Problem with Time Windows Part II: Metaheuristics. *Transportation Science*, Vol. 39, No. 1, (February 2005) pp. 119-139, ISSN 0041-1655

Carić, T.; Fosin, J.; Galić, A.; Gold, H. & Reinholz, A. (2007). Empirical Analysis of Two Different Metaheuristics for Real-World Vehicle Routing Problems, In: *Hybrid Metaheuristics 2007*, Bartz-Beielstein, T. et al. (Eds.), Lecture Notes in Computer Science (LNCS) 4771, pp. 31-44, Springer-Verlag, ISBN 978-3-540-75513-5, Berlin/Heidelberg

Cerny, V. (1985). A Thermodynamical Approach to the Travelling Salesman Problem: An Efficient Simulation Algorithm. *Journal of Optimization Theory and Applications*, Vol. 45, No. 1, (January 1985) pp. 41-51, ISSN 0022-3239

Clarke, G. & Wright, J.W. (1964). Scheduling of Vehicles from a Central Depot to a Number of Delivery Points, *Operations Research*, Vol. 12, No. 4, (July-August 1974) pp. 568-581, ISSN: 0030-364X

Cordeau, J.-F.; Desaulniers, G.; Desrosiers, J.; Solomon, M. & Soumis, F. (2002). The Vehicle Routing Problem with Time Windows. In: *The Vehicle Routing Problem*, Toth, P. & Vigo, D. (Eds.), pp. 157-193, SIAM Publishing, ISBN 0-89871-498-2, Philadelphia

Flood, M.M. (1956). The Traveling Salesman Problem, *Operations Research*, Vol. 4, No. 1, (February 1956) pp. 61-75, ISSN: 0030-364X

Galić, A.; Carić, T. & Gold, H. (2006a). MARS - A Programming Language for Solving Vehicle Routing Problems. In: *Recent Advances in City Logistics*, Taniguchi, E. & Thompson, R. (Eds.), pp. 48-57, Elsevier, ISBN 0-08-044799-6, Amsterdam

Galić, A.; Carić, T.; Fosin, J.; Ćavar, I. & Gold, H. (2006b). Distributed Solving of the VRPTW with Coefficient Weighted Time Distance and Lambda Local Search Heuristics. *Proceedings of the 29th International Convention on Information-Communications Technology*, pp. 247-252, Opatija, May 2006, MIPRO, Rijeka, Croatia

Gillett, B.E. & Miller, L.R. (1974). A Heuristic Algorithm for the Vehicle-Dispatch Problem, *Operations Research*, Vol. 22, No. 2, (March-April 1974) pp. 340-349, ISSN: 0030-364X

Homberger, J. & Gehring, H. (2005). A two-phase hybrid metaheuristic for the vehicle routing problem with time windows. *European Journal of Operational Research*, Vol. 162, No. 1, (April 2005) pp. 220–238, ISSN 0377-2217

Hoos, H. & Stützle, T. (2005). *Stochastic Local Search: Foundation and Application*, Elsevier/ Morgan Kauffman, ISBN 1-55860-872-9658, San Francisco

Kirkpatrick, S.; Gelatt, C.D. & Vecchi Jr., M.P. (1983). Optimization by Simulated Annealing. *Science*, New Series, Vol. 220, No. 4598, (May 13, 1983) pp. 671-680

Laporte, G. (1992). The Vehicle Routing Problem: An Overview of Exact and Approximative Algorithms. *European Journal of Operational Research*, Vol. 59, No. 3, (June 1992) pp. 345-358, ISSN 0377-2217

Laporte, G. & Semet, F. (2002). Classical Heuristics for the Capacitated VRP, In: *The Vehicle Routing Problem*, Toth, Toth, P. & Vigo, D. (Eds.), pp. 109-128, SIAM Publishing, ISBN 0-89871-498-2, Philadelphia

Laurenço, H.R. & Serra, D. (2002). Adaptive search heuristics for the generalized assignment problem. *Mathware & Soft Computing*, Vol. 9, No. 2-3, pp. 209-234, ISSN 1134-5632

Le Bouthillier, A. & Crainic, T.G. (2005). Cooperative parallel method for vehicle routing problems with time windows. *Computers and Operations Research*, Vol. 32, No. 7, (July 2005) pp. 1685-1708, ISSN 0305-0548

Lim, A. & Zhang, X. (2007). A Two-Stage Heuristic with Ejection Pools and Generalized Ejection Chains for the Vehicle-Routing Problem with Time Windows, *INFORMS Journal on Computing*, Vol. 19, No. 3, (Summer 2007) pp. 443-457, ISSN 1091-9856

Mester, D.; Bräysy, O. & Dullaert, W. (2007). A multi-parametric evolution strategies algorithm for vehicle routing problems. *Expert Systems with Applications*, Vol. 32, No. 2, (February 2007) pp. 508-517, ISSN 0957-4174

Metropolis, N.; Rosenbluth, A.W.; Rosenbluth, M.N. & Teller, A.H. (1953). Equations of State Calculations by Fast Computing Machines. *The Journal of Chemical Physics*, Vol. 21, No. 6, (June 1953) pp. 1087-1092, ISSN 0021-9606

Pisinger, D. & Röpke, S. (2005). A general heuristic for vehicle routing problems. *Technical Report*, Department of Computer Science, University of Copenhagen, Copenhagen, Denmark

Reinholz, A. (2003). Ein statistischer Test zur Leistungsbewertung von iterativen Variationsverfahren. *Technical Report 03027*, SFB559, University of Dortmund (in German)

Solomon, M. (1987). Algorithms for the Vehicle Routing and Scheduling Problems with Time Windows Constraints. *Operations Research*, Vol. 35, No. 2, (March-April 1987) pp. 254-265, ISSN 0030-364X

Appendix – Pseudo code and main FSL scripts for SA, ILS and VNS algorithms

```
procedure SA()
   T := InitialTemperature()
   initial := InitialSolution()
   s := LocalSearch(initial)
   best := s
   while not Terminate() do
      s' := Perturbate(s)
      s" := LocalSearch(s')
      if f (s") < f (s) then
          s := s"
      else
          j := rnd(0, 1)
          k := -((f(s") - f(best)) / f(best)) /  T
          if j < exp(k) then
              s := s"
          endif
      endif
          if f (s) < f(best) then
              best := s
          endif
              T := CoolingSchedule()
    endwhile
return best
end
```

```
function SA(Solution initial) returns Solution
   Solution best, s, s1, s2
   boolean Terminate = false
   int MaxIterations = 100, counter = 0
   s = LocalSearch(initial)
   best = s
   double T = InitialTemperature(), j, k
   while not Terminate
       s1 = Perturbate(s)
       s2 = LocalSearch(s1)
       if better(s2, s) then
           s = s2
       else
           j = random()
           k = -((cost(s2) - cost(best)) / cost(best)) / T
           if j < exp(k) then
               s = s2
           endif
       endif
      if better(s, best) then
           best = s
      endif
      T = CoolingSchedule(T)
      counter = counter + 1
      if counter >= MaxIterations then
           Terminate = true
      endif
   endwhile
       return best
endfunction
```

```
procedure ILS()
   initial := InitialSolution()
   s := LocalSearch(initial)
   best := s
   while not Terminate() do
      s' := Perturbate(s)
      s" := LocalSearch(s')
      if f (s") < f (best) then
          best := s"
          s := best
      endif
   endwhile
   return best
end
```

```
function ILS(Solution initial) returns Solution
   Solution s, s1, s2, best
   boolean Terminate = false
   int MaxIterations = 100, counter = 0
   s = LocalSearch(initial)
   best = s
   while not Terminate
      s1 = Perturbate(s)
      s2 = LocalSearch(s1)
      if better(s2, best) then
          best = s2
          s = best
      endif
      counter = counter + 1
      if counter >= MaxIterations then
          Terminate = true
      endif
   endwhile
   return best
endfunction
```

```
procedure VNS()
   best := InitialSolution()
   op := 1
   while not Terminate() do
      success := DoOperator(op)
      if succes then
          op := 1
      else
          op := op + 1
      if op > OperatorCount() then
          if cost(CurrentSolution()) < cost(best) then
              DoIntraRoute()
              best := CurrentSolution()
          endif
              DoPreturbate()
              op := 1
      endif
      endif
   endwhile
   return best
end
```

```
function VNS(Solution initial) returns Solution
   boolean Terminate = false, success = false
   int MaxIterations = 100, counter = 0
   int operator = 1, operatorcount = 5
   Solution best = initial
   SetCurrentSolution(initial)
   while not Terminate
      success = DoOperator(operator)
      if success then
          operator = 1
      else
          operator = operator + 1
          if operator > operatorcount then
              if better (CurrentSolution(), best) then
                  intraRoute()
                  best = CurrentSolution()
              endif
                  counter = counter + 1
                  if counter >= MaxIterations then
                      Terminate = true
                  else
                      DoPreturbate()
                      operator = 1
                  endif
          endif
      endif
   endwhile
   return best
endfunction
```

Vehicle Routing Problem

Edited by Tonci Caric and Hrvoje Gold

ISBN 978-953-7619-09-1
Hard cover, 142 pages
Publisher InTech
Published online 01, September, 2008
Published in print edition September, 2008

The Vehicle Routing Problem (VRP) dates back to the end of the fifties of the last century when Dantzig and Ramser set the mathematical programming formulation and algorithmic approach to solve the problem of delivering gasoline to service stations. Since then the interest in VRP evolved from a small group of mathematicians to a broad range of researchers and practitioners from different disciplines who are involved in this field today. Nine chapters of this book present recent improvements, innovative ideas and concepts regarding the vehicle routing problem. It will be of interest to students, researchers and practitioners with knowledge of the main methods for the solution of the combinatorial optimization problems.

InTech Europe
University Campus STeP Ri
Slavka Krautzeka 83/A
51000 Rijeka, Croatia
Phone: +385 (51) 770 447
Fax: +385 (51) 686 166
www.intechopen.com

InTech China
Unit 405, Office Block, Hotel Equatorial Shanghai
No.65, Yan An Road (West), Shanghai, 200040, China
中国上海市延安西路65号上海国际贵都大饭店办公楼405单元
Phone: +86-21-62489820
Fax: +86-21-62489821