Journal of Computational Physics **159,** 139–171 (2000)
doi:10.1006/jcph.1999.6413, available online at http://www.idealibrary.com on IDEAL®

Record Breaking Optimization Results Using the Ruin and Recreate Principle

Gerhard Schrimpf,* Johannes Schneider,† Hermann Stamm-Wilbrandt,* and Gunter Dueck*

*IBM Scientific Center Heidelberg, Vangerowstr. 18, D-69115 Heidelberg, Germany; †Faculty of Physics, University of Regensburg, D-93040 Regensburg, Germany
E-mail: schrimpf@de.ibm.com, Johannes.Schneider@physik.uni-regensburg.de, stammw@de.ibm.com, dueck@de.ibm.com

Received December 24, 1998; revised November 16, 1999

A new optimization principle is presented. Solutions of problems are partly, but significantly, ruined and rebuilt or recreated afterwards. Performing this type of change frequently, one can obtain astounding results for classical optimization problems. The new method is particularly suited for more complex optimization problems ("discontinuous" ones, problems with hard-to-find admissible solutions, problems with complex objectives or many constraints). The method is an all-purpose-heuristic. Numerical results are given for the Traveling Salesman Problem, for the Vehicle Routing Problem with time windows, and for network optimization. Numerical evidence for the quality of the proposed principle is given. For most of the instances of a research library of problems, the ruin and recreate (R&R) implementation achieved the best published results. For many instances, better or much better solutions could be found.

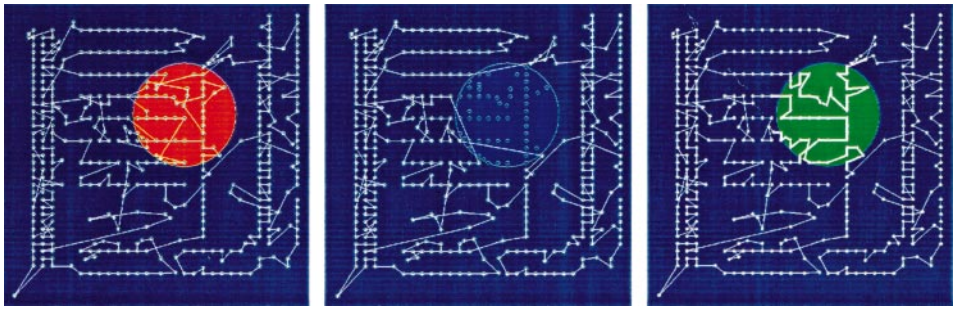
Key Words: combinatorial optimization; Monte Carlo, threshold accepting; global optimization; Traveling Salesman Problem; Vehicle Routing Problem; network optimization.

I. RUIN AND RECREATE, A FIRST LOOK AT THE PRINCIPLE

Before we give a more systematic introduction, we want to give the reader a quick feeling for this new class of algorithms we introduce here. The basic element of our idea is to obtain new optimization solutions by a considerable obstruction of an existing solution and a following rebuilding procedure. Let's look at a famous Traveling Salesman Problem which was often considered in the literature (PCB442 problem of Grötschel [1–8]). Suppose we have found some roundtrip through all of the 442 cities like in Fig. 1.

That's our initial or current solution of the problem. We "ruin" now a significant part of the solution. That's the easy part of it. When ruining the solution, think of a major

0021-9991/00 $35.00



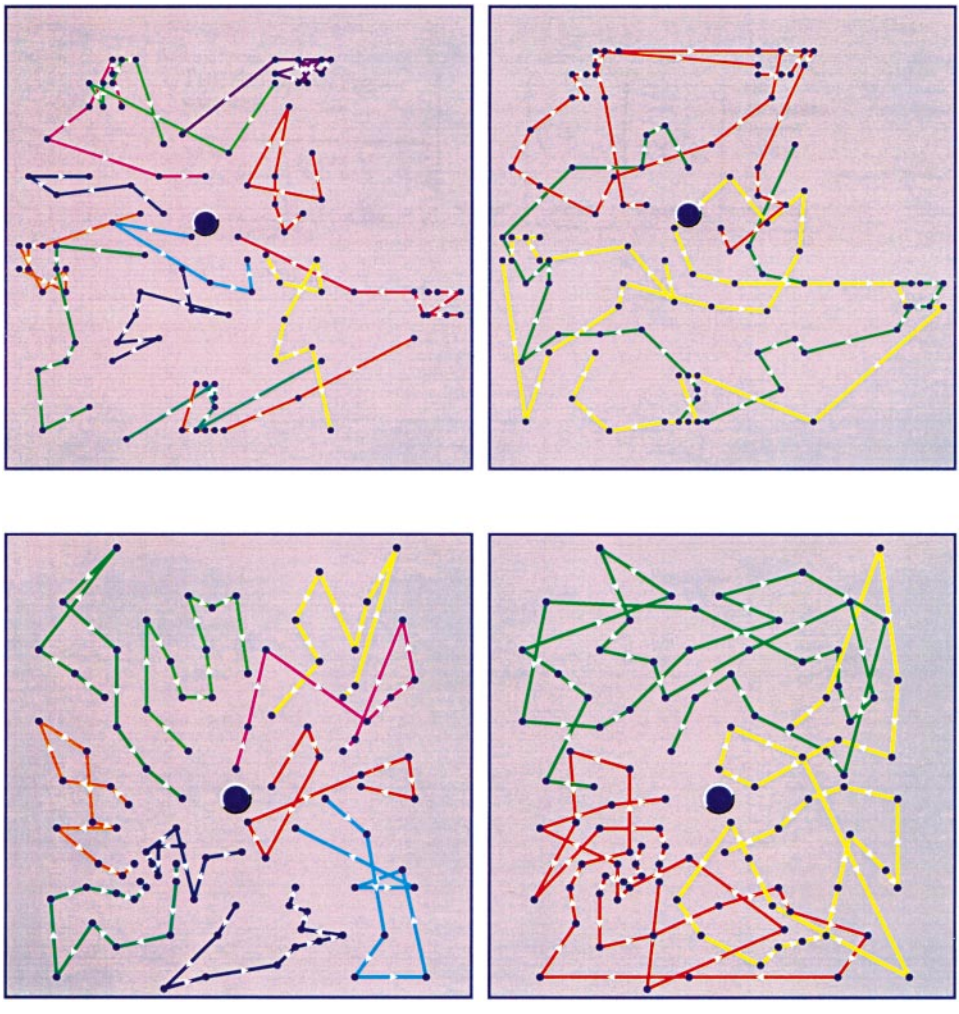


FIG. 1. TSP-instance PCB442. Left, rather bad; middle, after a radial ruin; right, recreated.

FIG. 2. Best solutions for 4 instances from Solomon's library of VRPTW problems. Left top, RC105; right top, RC206; left bottom, R107; right bottom, R202.

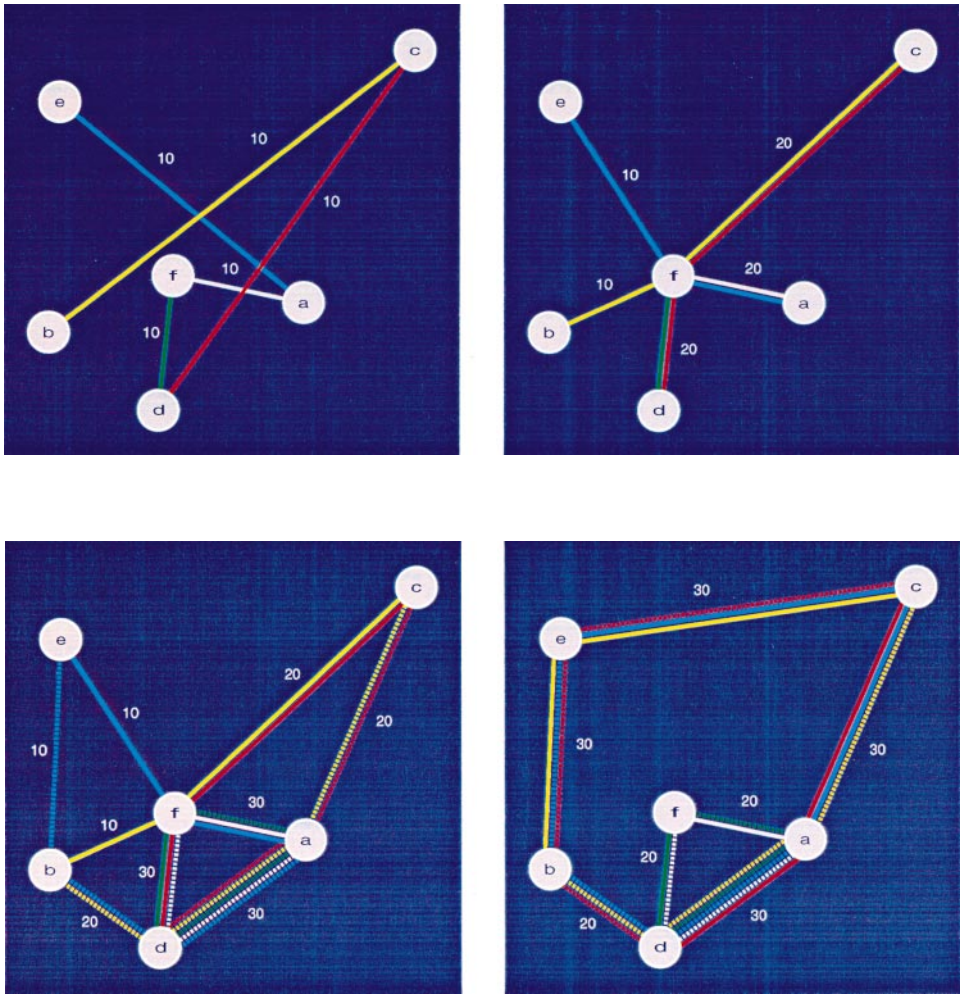


FIG. 3. NOPR on 6 locations with 5 demands of 10 Mbps. Left, input-demands and 1-hop NOP optimum with total length 885; right, 2-hop NOP optimum with total length 562.

FIG. 4. Left, 2-step 2/3-hop NOPR optimum with total length 1048; right, 2/3-hop NOPR optimum with total length 861.

disintegration or devastation. Mathematically speaking, we take some cities in the shaded area out of the current tour and connect the remaining (or surviving) cities to a shorter round trip. We say, "the cities which are disintegrated by the ruin move are not any longer served by the traveling salesman." In the final step, we recreate this partial solution after its ruin to a full TSP solution again. That's the harder part of the algorithm. There are many ways to recreate the ruined part of the solution, such that this is obviously an important point to be discussed in this paper.

Now we have a first imagination of how the ruin and recreate principle works. If we had to write a R&R based optimization routine we have to think about the kind and size of the disintegration steps and to argue about how to recreate ruined parts of the solution. We can overlay a decision rule whether we should accept the rebuilt structure or rather keep the original one. We could only accept better solutions ("Greedy Acceptance") or proceed according to Simulated Annealing [9–11], Threshold Accepting [3], or Great Deluge [4] methods.

In the next section, we discuss R&R in more detail, however, in general. We present tables which show results of R&R kind optimization for vehicle routing giving overwhelming evidence for the power of the R&R principle. After that, we proceed with a very short overview of possible solution change acceptance rules to be used in connection to R&R changes. Then we turn to the discussion of the vehicle routing problem, the network optimization problem, and the detailed TSP studies.

II. RUIN AND RECREATE

A. *Strategy*

The *ruin and recreate* method proposes using the well-known concepts of Simulated Annealing [9, 10] or Threshold Accepting [3] with bold, large moves instead of smaller ones. For "simple structured" problems like the Traveling Salesman Problem there is no real need to use large moves, because algorithms usually deliver near-to-optimum solutions with very small moves already. Dealing with complex problems, however, we encountered in our research team severe difficulties using these classical algorithms. If we considered wide area networks, or very complex vehicle routing tasks, we faced troubles.

- Complex problems often can be seen as "discontinuous": If we walk one step from a solution to a neighbor solution, the heights or qualities of these solutions can be dramatically different, i.e., the landscapes in these problem areas can be very "uneven."
- Solutions of complex problems often have to meet many constraints, and it is often even hard to get just admissible solutions. Neighbor solutions of complex schedules, for instance, are usually inadmissible solutions, and it may be very hard to walk in such a complex landscape from one admissible solution to another neighbored admissible solution. Many forms of the classical algorithms try to avoid the "admissibility problem" by modeling artificial penalty functions, but they typically can get stuck in "slightly inadmissible" solutions which might not be allowed at all.

Throughout this paper, we will "think" in a new paradigm: *ruin and recreate*. We ruin a quite large fraction of the solution and try to restore the solution as best as we can. Hopefully, the new solution is better than the previous one.

The R&R approach will show an important advantage in this paper: If we have disintegrated a large part of the previous solution we have a lot of freedom to create a new one.

We can reasonably hope that, in this large space of solutions, it is possible to find again an *admissible* solution. Hopefully we get "discontinuous" problems (problems with very complex objective functions, problems where the solutions have to meet many side conditions) which are more tractable using special large moves.

We demonstrate the power of the new paradigm by numerical results for the vehicle routing problem with time windows. We chose this problem area because we felt that this problem is the "easiest of the complex" problems. It is hard enough to recognize that the classical algorithms may be not really suited here anymore. It is "easy" enough to find some published problem instances which are already extensively studied in the literature. We took more than 50 problem instances from the library of Solomon [12] and tested our R&R implementation on them.

Tables II to V present our complete numerical results on the Solomon library. In most cases, our R&R implementation gave solutions at least as good as the currently published record. In many cases, we could achieve better and much better results. We highlight here four rather complex examples out of the Solomon library. Figure 2 shows our best R&R solutions for the problems R107, R202, RC105, and RC206. Here, we could achieve significant improvements: instance R107, 1119.93 (down from 1159.86); instance R202, 1195.30 (down from 1530.49); instance RC105, 1633.72 (down from 1733.56); instance RC206, 1152.03 (down from 1212.64). Now, the final results are clear. Before we can come to the main part of the paper, where we explain "how it really works," we add a remark from the practical point of view.

B. *Remarks for Applications*

Throughout this paper we will study R&R algorithms which build one *admissible* solution after each other. We don't use artificial constructions like penalty terms in the objective function at all. This property of the R&R principle has quite significant implications for, say, commercial vehicle routing systems. At every time during running a R&R optimization you have a fully admissible solution available. This is in contrary to many vehicle routing implementations where you often achieve only solutions with small violations of the given restrictions which you have to resolve in practice by neglect, by tolerance, by wiping them out by hand, by brute force—take a full additional truck for a small packet—or by calling a customer for a more suitable time window.

III. CLASSICAL IMPROVEMENT HEURISTICS

A. *Simulated Annealing and Its Relatives*

Simulated Annealing [9–11], Threshold Accepting [3], the Great Deluge Algorithm [4, 5], and related Monte Carlo-type optimization algorithms apply ideas of statistical physics and applied mathematics to find near-to-optimum solutions for combinatorial optimization problems. These are all iterative improvement algorithms. They start with an initial configuration and proceed by small exchanges in the actual or current solution to get a tentative new solution. The tentative new solution is evaluated, i.e., its objective function, e.g., its total cost, is computed. The algorithmic decision rule is applied. It is decided if the tentative new solution is kept as the current solution; in case of acceptance the new solution is taken as the new current solution. Of course, we can use all these acceptance strategies within a R&R optimization. Usually, optimization algorithms work with very small or very

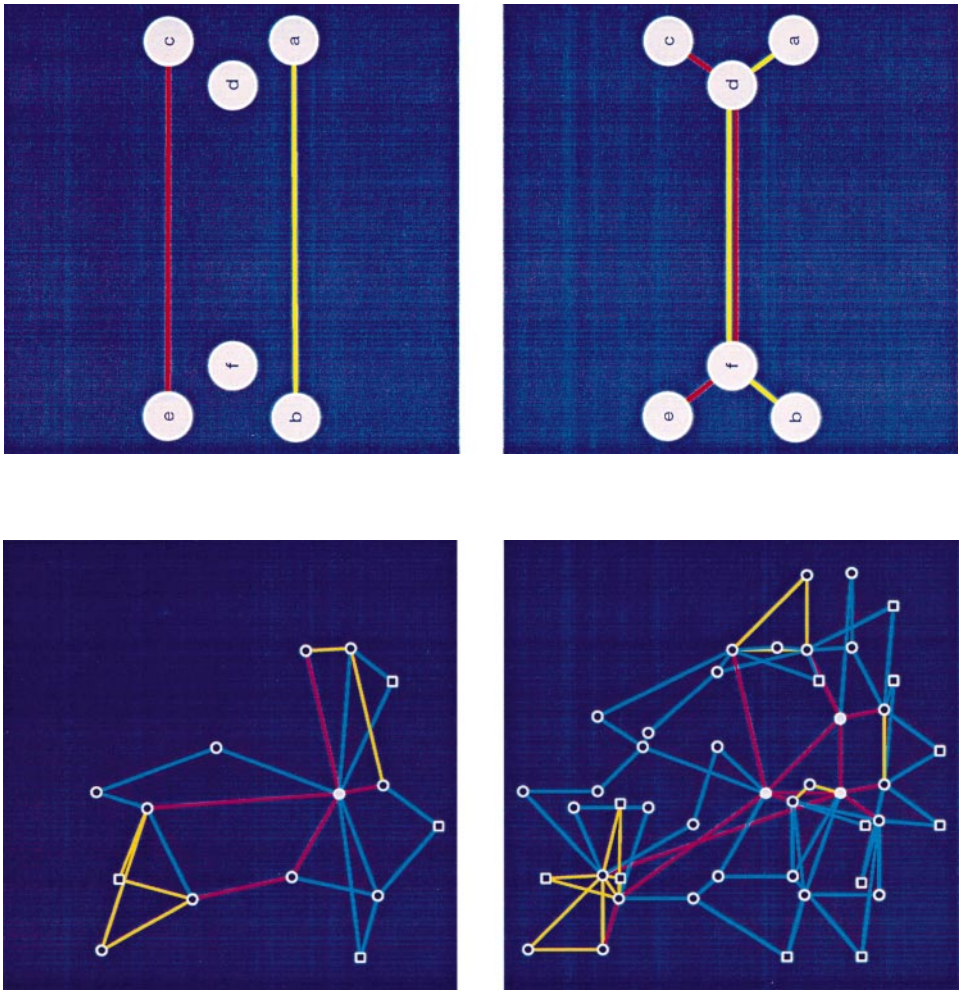


FIG. 5. Left, 1-hop NOP optimum with total length 82; right, 3-hop NOP optimum with total length 55.

FIG. 6. Topologies of solutions for systems *N15* (left) and *N45* (right). Computing centers *C* are colored in grey. Circles denote locations with switching facility; squares denote locations without switching facility. Trunk bandwidths are 64 Kbps (cyan), 128 Kbps (gold), and 2 Mbps (magenta). Total costs are 11,437 units for system *N15* and 25,304 units for system *N45*, corresponding to a synergy of 26.3%.

local changes in a current solution. In this paper we'll also use these type of acceptance rules, however, with considerable changes "made by meteorites."

1. Decision rules. The different algorithms work with this same structure, but they use different decision rules for acceptance/rejection. In a Random Walk (RW) *every* new solution is accepted. The Greedy Acceptance (GRE) accepts every solution which is *better* than the current solution. Simulated Annealing (SA) procedures accept every better solution and, with a certain probability, also solutions being worse than the current solution. Threshold Accepting (TA) [3] accepts every solution which is *not much worse* than the current solution, where "not much" is defined by a threshold. The Great Deluge Algorithm (GDA) [4, 5] rejects every solution below a required quality level (the "waterline"). This principle is related to a Darwinian approach. Instead of "only the fittest will survive" the deluge principal works with "only the worst will die."

2. Mutations. Of course, the definition of an exchange in a current solution depends on the optimization problem. Let us look, for instance, at the Traveling Salesman Problem. In order to modify the current solution to get a new tentative chosen solution, different types of local search mutations are commonly applied. An *exchange* exchanges two nodes in the tour. The *Lin-2-opt* cuts two connections in the tour and reconstructs a new tour by insertion of two new connections, which can be shown to provide better results [13]. A *node insertion move* (NIM, or Lin-2.5-opt [14]) removes a node from the tour and reinserts it at another position. Moreover, *Lin-3-opt, Lin-4-opt, and Lin-5-opt* [15, 16] are sometimes applied, cutting three, four, and five connections and choosing one of 4, 25, and 208 possibilities to recreate the new tour, respectively.

B. *Set Based Algorithms*

Simulated annealing and related techniques have in common that a new configuration is generated based on the actual one. No information about former configurations is used. Genetic algorithms and evolution strategies both use a large set of configurations as individua of a population. Tabu Search saves information about former configurations in its Tabu List and therefore also depends on a set of configurations. Searching for Backbones reduces the complexity of a problem by eliminating parts which are supposed to be already optimally solved.

1. Genetic algorithms and evolution strategies. Genetic algorithms mostly use different kinds of crossover operators generating children from parent configurations, while evolution strategies concentrate on mutations altering a member of the population [17]. With both techniques new configurations are produced; various implementations of these algorithms only differ in the type of the used mutations and in the choice which configurations are allowed to reproduce or to mix with each other or forced to commit suicide.

2. Tabu Search. Tabu Search [18] is a memory based search strategy to guide the system being optimized away from parts of the solution space which were already explored. This can be achieved either by forbidding solutions already visited or structures some former solutions had in common, which are stored in a Tabu List. This list is updated after each mutation according to some proposed rules, which have to guarantee that the optimization run never reaches a solution again which was already visited before, that the Tabu List size does not diverge, and that a good solution can be achieved.

3. Searching for Backbones. Searching for Backbones [6] compares results of independent optimization runs for equal parts. These parts are supposed to be optimal, i.e., to be parts of the optimum solution. This information is considered in the next series of optimization runs in which these parts remain unchanged. The new solutions are supposed to be better than the previous ones because the optimization could concentrate on parts which are more difficult to solve optimally. This algorithm is repeated iteratively until all optimization runs produce the same solution.

IV. R&R FOR VEHICLE ROUTING

A. *Introduction*

We turn to the optimization of vehicle routing. We only consider problems where a *fleet of vehicles* starts from a *central depot*. All the vehicles have a given *maximum capacity*. They serve a *set of customers* with known *demands*. The solution of the VRP (vehicle routing problem) consists of a minimum cost set of routes of the vehicles satisfying the following conditions: Each vehicle starts and ends its route at the central depot. Each customer is served exactly once. The sum of the demands of all customers served by a single vehicle does not exceed its capacity. The cost of a set of routes for the VRP is the sum of the length of all routes. Each customer may add a *time window* or *time interval* restriction to the problem, that is, for every customer there is an earliest and a latest time where the vehicle is allowed to serve the customer. If such restrictions are imposed, we speak of the VRPTW (vehicle routing problem with time windows).

In the literature there is a well known collection of 56 VRPTW instances from Solomon [12] which is used by many researchers for evaluation of their VRPTW solving systems. These problems can be classified into 3 groups, with distinct characteristics of the distribution of the customer locations: random (R), clustered (C), and a mixture of both (RC). Furthermore, each of these groups can be split into problems with low vehicle capacity (type 1) or high vehicle capacity (type 2). One therefore generally speaks of six problem sets, namely R1, C1, RC1, R2, C2, and RC2. Members of a single set vary both in the distribution of the time windows and in the demands of the single customers (see Table I).

All problems consist of 100 customer locations and one depot. Both the distances and the travel times between the customers are given by the corresponding euclidean distances. Therefore, this library incorporates many distinguishing features of vehicle routing with

TABLE I
Classification of the VRP-Parts and Service Times of Solomon's Library

Set	Coord.	Demand	Maximum vehicle capacity	Service time
R1	a	A	200	10
C1	b	B	200	90
RC1	d	C	200	10
R2	a	A	1000	10
C2	c	B	700	90
RC2	d	C	1000	10

Note. The a–d and A–C are 101/100-dimensional vectors representing the coordinates (a–d) and demands (A–C) of the customers.

time windows: fleet size, vehicle capacity, spatial and temporal customer distribution, time window density, time window width, and customer service times. The objective of the problem is to service all customers while first minimizing the number of vehicles and second minimizing the travel distance.

Results in the literature are not completely comparable. While some authors used euclidean distances, others truncated the distances to one decimal place. The reason for the truncation is that some exact algorithms for solving VRPTW are integer based, like dynamic programming. As in the recent publications [19, 20] we used real, double-precision distances. Desrochers *et al.* [21] used a LP relaxation of the set partitioning formulation of the problem and solved it by column generation. The LP solutions obtained are excellent lower bounds. With a branch-and-bound algorithm they solved 7 out of the 56 problems exactly. Potvin *et al.* [22] used Genetic Search to solve the VRPTW. The basic principle they used is the creation of a methodology for merging two vehicle routing solutions into a single solution that is likely to be feasible with respect to the time window constraints. Thangiah *et al.* [19] applied a technique where customers are moved between routes defining neighborhood solutions. This neighborhood is searched with Simulated Annealing and Tabu Search. The initial solution is obtained using a push-forward insertion heuristic and a genetic algorithm based sectioning heuristic. They solved $56 + 4$ problems from the literature (Subsection IV.A, 4 other problems), and for 40 of these they obtained new optimum solutions. For 11 out of the 20 remaining problems they obtained solutions equal to those best known. Later, Rochat *et al.* [20] adopted their probabilistic technique to diversify, intensify, and parallelize the local search to the VRP and VRPTW. They used a simple first-level tabu search as the basic optimization technique and were able to significantly improve its results by their method. In using a post-optimization procedure they improved nearly 40 of the 56 instances. This shows that their method has in most cases significant advantages over the previous methods.

B. *Ruin*

In the first section, we introduced the concept of "ruin" with a discussion of the Traveling Salesman Problem: We removed all those cities from a round trip that were located in the radial deletion area. Let's use a different wording for "a city being deleted from the tour." We would like to say, "a city is not served anymore by the traveling salesman." After the ruin the salesman is serving only all the cities that remain after the ruin step. If we deal with vehicle routing, we shall disintegrate a solution by removing destinations or customers or packets to be delivered. We say that these customer destinations are not serviced anymore after a ruin. When we discuss the recreation step, we say that these customers are trying to be serviced again, this time by the most appropriate vehicle. There are many ways to ruin a solution. Below we give some exact definitions of particular ruin strategies used in our implementation. Of course, you may feel free to invent new ones. Let's browse briefly through some obvious ideas.

For the TSP, you could ruin according to the first section. We call this procedure *radial* ruin. We can remove cities from the service by flipping coins: every city is removed with a certain probability. We can remove a shorter or longer string of cities within a round trip. For the vehicle routing problems, there are many more promising fashions to disintegrate solutions. Since every vehicle rides along a round trip, any TSP destruction method can be used. But we have a broader spectrum of possibilities. Of course, we can remove all customers inside a disk in a plane. This is a type of "space deletion" or "space ruin." We could also remove every customer which is serviced in the current solution inside a certain

time interval: this is a "time deletion." Furthermore, we could apply "volume deletions" or "weight deletions" that remove customers receiving packets whose weights or volumes are in a certain range. All these ruins remove customers who are adjacent in some sense. For example, many years ago we studied knapsack problems [23] and found already at that time that packet exchanges are most useful if they are restricted to packets of similar size. Within the new framework we present here we would say that we have used *volume ruins* of small size.

Let's start here to define some kinds of ruins: some packets to be delivered or customers to be served or cities to be visited are removed from the system T. They are not served any longer, or we say, they are put into a bag B of unserved items. *Radial ruin*: This is the classical ruin from which the imaginative picture is derived. Select randomly a node c from the set T of all $\mathcal{N}$ nodes (packets, customers, cities). Select a random number $\mathcal{A}$ with $\mathcal{A} \leq [\mathcal{F} \cdot \mathcal{N}]$, $\mathcal{F}$ being a fraction, a number between 0 and 1. Remove c and its $\mathcal{A}-1$ nearest neighbors from T and put them into the bag B. The "nearest neighbors" are defined according to a certain metric. For our vehicle routing instances we use the euclidean distance. *Random ruin*: Select a random number $\mathcal{A} \leq [\mathcal{F} \cdot \mathcal{N}]$, $0 \leq \mathcal{F} \leq 1$. Remove $\mathcal{A}$ randomly selected nodes from T and put them into B. Note that random ruin is a global strategy whereas radial ruin is a more local one. *Sequential ruin*: Remove $\mathcal{A} \leq [\mathcal{F} \cdot \mathcal{N}]$ succeeding nodes from a single, randomly selected round trip.

C. *Recreate*

Suppose we have ruined a solution. This means that we have a set of customers which are no longer serviced (by a salesman or by a vehicle). The recreation of the solution means the reinsertion of these customers into the system. Ideally, we could try to invent an algorithm such that the solution is recreated exactly optimally. On the other hand, we could try to take every customer, one after another, and insert them into the system in a more or less clever way. It can be seen that there is a whole universe of methods to recreate the system. In this paper, we want to present the *principle* of R&R, the pure idea. We have restricted ourselves to studying the most obvious recreation of all: *best insertion*. Best insertion means that we add all customers out of service successively in the best possible way to the system. Do not violate any restriction (e.g., time window constraints), such that every recreation ends up in a fully admissible solution. Let us emphasize another point: We used only this rule of best insertion, and we could achieve record results by exclusively using this raw principle. Of course, it is possible to study more elaborate, more sophisticated hybrid algorithms to achieve better results. However, this is not the intention of the present paper. We just want to state the power of the most simple form of R&R, which is the following: In a partially disintegrated routing solution we have a certain number of customers (or cities) put into the bag. Now, we take these customers in a random order out of the bag and perform best insertion: The customer asks every vehicle if and at which position it is possible to serve him on its tour and what the additional costs would be. The minimum cost insertion is chosen. It may not be possible at all to insert a customer into the solution due to the capacity or time window constraints. In this case, an additional vehicle is inserted into the system.

D. *Overall Optimization Procedure*

Throughout this paper, we exclusively use radial R&R ($\text{R\&R}_{\text{rad}}$), random R&R ($\text{R\&R}_{\text{ran}}$), and sequential R&R ($\text{R\&R}_{\text{seq}}$), which connect the chosen ruin mode with the best insertion technique. The scheme should now be clear:

1. Start with an initial configuration.
2. Choose a ruin mode.
3. Choose a number $\mathcal{A} \leq [\mathcal{F} \cdot \mathcal{N}]$ of nodes to be removed.
4. Ruin.
5. Recreate.
6. Decide if you accept the new solution according to a decision rule (Simulated Annealing. Threshold Accepting, etc.). If you accept, proceed with (2), using the new solution, else restart with (2) using the current (old) solution.

E. *Details of the Implementation*

A route of a vehicle is represented by a sequence of customers $C_1, C_2, \ldots, C_k$. For a given route C_1 and C_k are pseudo-customers representing the starting and ending at the central depot. The only point left unspecified in the Ruins and Recreates is the representation of time. We choose the representation as time intervals at each customer (including the pseudo-customers). This has the advantage that all possible realizations in time of a given customer sequence in a route are represented simultaneously. Let C_i^{first} and C_i^{last} be the first and last time the customer C_i allows the start of the service and C_i^{job} be its service time (the service time is 0 for the pseudo-customers). Now C_i^{early} and C_i^{late} are always updated to represent the first and last time the service may start at C_i inside the actual route. A time window conflict exists at customer C_i if and only if $C_i^{\text{early}} > C_i^{\text{late}}$. The travel time between two customers C_i and C_j is denoted by $d(C_i, C_j)$. If customer C_i is inserted newly into the route it gets initialized by $C_i^{\text{early}} = C_i^{\text{first}}$ and $C_i^{\text{late}} = C_i^{\text{last}}$. Then the *early* and *late* entries are updated similarly, here demonstrated for the *early* entries,

$$\textit{for } i := 2 \textit{ to } k \textit{ do } C_i^{\text{early}} = \max\left\{C_i^{\text{early}}, C_{i-1}^{\text{early}} + C_{i-1}^{\text{job}} + d(C_{i-1}, C_i)\right\}.$$

After removal of a customer (due to an R&R ruin) resulting in the route $C_1, C_2, \ldots, C_k$ the early part of the update is done by

$$\textit{for } i := 2 \textit{ to } k \textit{ do } C_i^{\text{early}} = \max\left\{C_i^{\text{first}}, C_{i-1}^{\text{early}} + C_{i-1}^{\text{job}} + d(C_{i-1}, C_i)\right\}.$$

The late part of the update is done similarly.

For problems where it is not easy to achieve the desired number n_T of tours in a solution as known optimum/best from the literature we used the following modification of the normal approach: For each vehicle used in a configuration exceeding n_T we charged a constant amount of 50 units and scaled its costs by a factor of 5. For the configurations with at most n_T vehicles used this does not change anything, and for others this will lead the search into the right direction.

All runs were performed using Threshhold Accepting as the decision rule. The initial threshold T_0 was set to be half of the standard deviation of the objective function during a random walk. We used an exponential cooling schedule for the threshold T of the form

$$T = T_0 \cdot \exp(-\ln 2 \cdot x/\alpha), \tag{4.1}$$

where the half-life α was set to 0.1. The schedule variable x was increased from 0 to 1 during the optimization run. We applied a 1 : 1 mixture of $\text{R\&R}_{\text{rad}}$ and R&R random, using $\mathcal{F} = 0.3$ and $\mathcal{F} = 0.5$, respectively. The single computations were performed with 40,000 mutations,

consuming approximately 30 minutes of CPU time on a RS 6000 workstation, model 43P, 233 MHz. The neglect of time windows increased the computation speed approximately by a factor of three.

An important point to mention is the applicability of the R&R approach to very large vehicle routing problems. This is due to its inherent parallelizability and its ability to achieve comparable results even after localizing some computations. Most time consuming is the calculation of the cost of acceptance for a customer by a vehicle, especially for the recreate steps. The recreate step consumes about 90% of the whole computing time. These calculations can easily be parallelized on the vehicle basis, since the single vehicle calculations are independent. This is even valid for the single tests on different potential positions inside a vehicle tour.

F. *Results*

The work on VRPTW can be split into two groups: work on exact algorithms or heuristics and work on meta heuristics. The results prior to Rochat *et al.* [20] fall into the first category; the work of Rochat *et al.* itself falls into the second one. R&R is a strategy to solve complex combinatorial problems. The application of R&R to VRPTW is a heuristic and therefore falls into the first category.

We compare our results to the work prior to Rochat *et al.* [20] in Tables IV (left) and V (left) and to the work of Rochat *et al.* [20] in Tables IV (right) and V (right). In Tables IV (left) and V (left) there are 8 cases where our results are worse, 12 cases where our results are equal, and 36 cases where our solutions are better. There are only 3 cases where we missed the minimum number of tours, but 5 where we are one tour better than the best known result. Comparison of the results with the work of Rochat *et al.* shows that in 24 cases we receive the same results and have better results in 31 cases. Only 1 problem was solved better by Rochat *et al.* With two exceptions, where our solutions are one tour better, we always received solutions with the same number of tours. The large number of equal results may indicate that many of these 24 problems cannot be improved further.

The problems from groups C1 and C2 are solved by most authors and us with nearly the same quality, and for 5 out of the 9 problems from C1 optimally. Thus we consider these problem sets as "easy." For each of the other groups we present the best improvement for a problem we found in Table II.

The main aim of an optimization algorithm can either be to achieve a new best solution or to be used in practice. In the latter case a small variance in the (good) results is even more important than the average quality or the best solution that can be found by an algorithm. Table III shows the statistics over 50 runs of the 4 problems mentioned above. The value of ρ is the probability of the event, that a solution consists of the best known number of tours. It differs widely for these problems. For the first 3 problems even the worst solutions found are better than the previous optimum solution. For problem RC206 all runs resulted in solutions with the best known number of tours used. Here, the worst solution is worse than the best known, but even here the mean value of these solutions is better than the actual best known.

V. R&R FOR NETWORK OPTIMIZATION

A. *Problem Definition*

In this section we introduce the hard problem of network optimization (NOP). Attempting to get reasonably good solutions for customers, we realized that the simple annealing

TABLE II
Tour Sequences for Instances R107, R202, RC105, and RC206

Instance	n_C	Sequence	l	l_{tot}
R107	9	28-50-76-40-53-68-29-24-80	115.86	119
	9	33-81-65-71-9-66-20-51-1	128.77	142
	9	48-47-36-64-49-19-82-18-89	126.00	167
	9	12-54-39-23-67-55-4-25-26	132.61	166
	12	2-57-43-15-41-22-75-56-74-72-73-21	103.08	129
	9	27-69-30-79-78-34-35-3-77	111.35	118
	11	60-83-45-46-8-84-5-17-61-85-93	113.47	151
	11	52-7-11-62-88-31-10-63-90-32-70	116.34	138
	12	95-97-42-14-44-38-86-16-91-100-37-98	105.74	181
	9	94-96-92-59-99-6-87-13-58	66.70	147
R202	37	96-59-92-98-85-91-14-42-2-21-72-39-23-15-38-44-16-61-99-18-8-84-86-5-6-94-95-97-43-74-13-37-100-93-17-60-89	392.26	560
	29	50-33-65-34-29-3-28-27-69-76-67-73-40-53-87-57-41-22-75-56-4-54-55-25-24-80-12-26-58	354.70	376
	34	83-45-48-47-36-63-64-11-19-62-88-30-71-78-79-81-9-51-90-49-46-82-7-10-20-32-66-35-68-77-1-70-31-52	448.34	522
RC105	8	72-71-81-41-54-96-94-93	127.54	120
	8	92-95-62-67-84-51-85-89	141.21	96
	8	82-12-11-87-59-97-75-58	137.55	172
	5	90-53-66-56-91	74.54	59
	8	65-83-64-99-52-86-57-74	116.71	124
	7	2-45-5-7-79-55-68	108.97	147
	9	98-14-47-15-16-9-10-13-17	121.02	149
	9	42-61-8-6-46-4-3-1-100	144.53	132
	7	63-23-19-22-49-20-77	160.78	143
	5	69-88-78-73-60	81.70	95
	9	39-36-44-38-40-37-35-43-70	132.62	183
	10	31-29-27-30-28-26-32-34-50-80	134.62	183
	7	33-76-18-48-21-25-24	151.62	111
RC206	33	69-98-2-45-5-44-42-39-38-36-40-41-61-88-53-78-73-79-7-6-8-46-4-3-1-43-35-37-54-96-93-91-80	334.39	592
	34	65-83-82-11-14-12-47-15-16-75-59-52-99-64-84-67-71-94-81-90-66-56-50-34-32-26-89-20-24-48-25-77-58-74	475.23	554
	33	72-92-95-62-31-29-27-28-30-33-63-85-51-76-18-21-23-19-49-22-57-86-87-97-9-10-13-17-60-55-100-70-68	342.42	578

Note. For the single vehicle tours the number of customers, n_C, its length l, and the total loading due to the serviced customers l_{tot} are given.

and threshold accepting methods were not satisfactory or failed, if you prefer this harder formulation. The landscape of this harder problem seems to be extremely "discontinuous." At this point, frustrated with the results of classical algorithms, we had the final ruin and recreate idea. We state the problem. In a wide area network (WAN) you have the task of transmitting certain amounts of information over an undirected graph, *the network*. Different nodes are connected by so-called *links*. Each link consists of zero or more communication lines, so-called *trunks*, each having a *bandwidth*, which is measured in bits per second (bps), kilobits per second (Kbps), or megabits per second (Mbps). If you want to communicate

TABLE III

Distribution of the Final Lengths, Calculated from 25 R&R, for Problems R107, R202, RC105, and RC206

Problem instance	Best known		R&R				
	n_T	l_{min}	ρ	l_{min}	$\langle l \rangle$	l_{max}	σ_l
R107	10	1159.86	0.24	1119.93	1125.84	1136.05	5.33
R202	3	1530.49	0.64	1195.30	1243.43	1316.48	30.73
RC105	13	1733.56	0.12	1633.72	1646.77	1663.03	12.18
RC206	3	1212.64	1.00	1152.03	1198.54	1256.62	29.84

Note. ρ is the probability of a R&R solution to consist of n_T tours, l is the best known length of the problem instance, and $l_{\min}$, $\langle l \rangle$, $l_{\max}$, and σ_l are the minimum, mean, maximum final length, and the standard deviation of our runs consisting of n_T tours.

TABLE IV

Comparison of the R&R Results with Literature Data for Problem Sets R1, C1, and RC1

Problem instance	Best solution previous to Rochat *et al.*			Best R&R solution		Problem instance	Best solution by Rochat *et al.*		Best R&R solution	
	n_T	l		n_T	l		n_T	l	n_T	l
R101	18	1607.7*	[21]	19	1645.7*	R101	19	1650.80	19	1650.80
R102	17	1434.0*	[21]	17	1481.2*	R102	17	1486.12	17	1486.12
R103	13	1207	[19]	13	1296.19	R103	14	1213.62	13	**1296.19**
R104	10	1048	[19]	10	**981.23**	R104	10	982.01	10	**981.23**
R105	14	1420.94	[22]	14	**1377.11**	R105	14	1377.11	14	1377.11
R106	12	1350	[19]	12	**1252.03**	R106	12	1252.03	12	1252.03
R107	11	1146	[19]	10	**1119.93**	R107	10	1159.86	10	**1119.93**
R108	10	989	[19]	9	**966.40**	R108	9	980.95	9	**966.40**
R109	12	1205	[22]	11	**1210.66**	R109	11	1235.68	11	**1210.66**
R110	11	1105	[19]	10	**1121.46**	R110	10	1080.36	10	1121.46
R111	10	1151	[19]	10	**1122.76**	R111	10	1129.88	10	**1122.76**
R112	10	992	[19]	10	**953.63**	R112	10	953.63	10	953.63
C101	10	827.3*	[21]	10	827.3*	C101	10	828.94	10	828.94
C102	10	827.3*	[21]	10	827.3*	C102	10	828.94	10	828.94
C103	10	835	[19]	10	**828.06**	C103	10	828.06	10	828.06
C104	10	840	[19]	10	**824.78**	C104	10	824.78	10	824.78
C105	10	828.94	[22]	10	828.94	C105	10	828.94	10	828.94
C106	10	827.3*	[21]	10	827.3*	C106	10	828.94	10	828.94
C107	10	827.3*	[21]	10	827.3*	C107	10	828.94	10	828.94
C108	10	827.3*	[21]	10	827.3*	C108	10	828.94	10	828.94
C109	10	828.94	[22]	10	828.94	C109	10	828.94	10	828.94
RC101	14	1669	[19]	15	1623.58	RC101	15	1623.58	15	1623.58
RC102	13	1557	[19]	13	**1477.54**	RC102	13	1477.54	13	1477.54
RC103	11	1110	[19]	11	1261.67	RC103	11	1262.02	11	**1261.67**
RC104	10	1204.07	[22]	10	**1135.52**	RC104	10	1135.83	10	**1135.52**
RC105	14	1602	[19]	13	**1633.72**	RC105	13	1733.56	13	**1633.72**
RC106	12	1485.67	[22]	12	**1384.26**	RC106	12	1384.92	12	**1384.26**
RC107	11	1274.71	[22]	11	**1230.54**	RC107	11	1230.95	11	**1230.54**
RC108	10	1281	[19]	10	**1147.26**	RC108	10	1170.70	10	**1147.26**

Note. Left, comparison to work prior to Rochat *et al.*: right, comparison to Rochat *et al.* Results, where R&R gives better results than known, appear in boldface. Lengths are calculated by euclidean distances. Exceptions with truncation to one decimal place are marked by an asterisk.

TABLE V

Comparison of the R&R Results with Literature Data for Problem Sets R2, C2, and RC2

Problem instance	Best solution previous to Rochat *et al.*			Best R&R solution		Problem instance	Best solution by Rochat *et al.*		Best R&R solution	
	n_T	l		n_T	l		n_T	l	n_T	l
R201	4	1354	[19]	4	**1252.37**	R201	4	1281.58	4	**1265.74**
R202	3	1530.49	[22]	3	**1195.30**	R202	4	1088.07	3	**1195.30**
R203	3	1126	[19]	3	**947.63**	R203	3	948.74	3	**947.63**
R204	2	914	[24]	2	**848.91**	R204	2	869.29	2	**848.91**
R205	3	1128	[19]	3	**994.43**	R205	3	1063.24	3	**1053.37**
R206	3	833	[19]	3	906.14	R206	3	912.97	3	**906.14**
R207	3	904	[19]	3	**811.51**	R207	3	814.78	3	**811.51**
R208	2	759.21	[22]	2	**726.82**	R208	2	738.60	2	**726.82**
R209	2	855	[19]	3	915.16	R209	3	944.64	3	**915.16**
R210	3	1052	[19]	3	**939.37**	R210	3	967.50	3	**963.67**
R211	3	816	[19]	2	**904.32**	R211	2	949.50	2	**904.32**
C201	3	591.56	[22]	3	591.56	C201	3	591.56	3	591.56
C202	3	591.56	[22]	3	591.56	C202	3	591.56	3	591.56
C203	3	591.55	[22]	3	**591.17**	C203	3	591.17	3	591.17
C204	3	590.60	[22]	3	590.60	C204	3	590.60	3	590.60
C205	3	588.88	[22]	3	588.88	C205	3	588.88	3	588.88
C206	3	588.49	[22]	3	588.49	C206	3	588.49	3	588.49
C207	3	588.32	[22]	3	**588.29**	C207	3	588.29	3	588.29
C208	3	588.49	[22]	3	**588.32**	C208	3	588.32	3	588.32
RC201	4	1249	[19]	4	1415.33	RC201	4	1438.89	4	**1415.33**
RC202	4	1221	[19]	4	**1162.80**	RC202	4	1165.57	4	**1162.80**
RC203	3	1203	[19]	3	**1051.82**	RC203	3	1079.57	3	**1051.82**
RC204	3	897	[19]	3	**798.46**	RC204	3	806.75	3	**798.46**
RC205	4	1389	[19]	4	**1302.02**	RC205	4	1333.71	4	**1302.02**
RC206	3	1213	[19]	3	**1152.03**	RC206	3	1212.64	3	**1152.03**
RC207	3	1181	[19]	3	**1068.86**	RC207	3	1085.61	3	**1068.86**
RC208	3	919	[19]	3	**829.69**	RC208	3	833.97	3	**829.69**

Note. Left, comparison to work prior to Rochat *et al.*; right, comparison to Rochat *et al.* Results, where R&R gives better results than known, appear in boldface. Lengths are calculated by euclidean distances.

from point *a* to point *b* in a network, you can subscribe to a trunk with a sufficient bandwidth. Your ISDN channel, for instance, is such a trunk and has 64 Kbps. Your provider may offer trunks with different bandwidths. In Germany, for instance, you can get 9.6, 19.2, 64, 128 Kbps trunks, or even 2 and 34 Mbps trunks. Depending on the length of the trunk and of its bandwidth you have to pay a fee per time unit, typically per month. The prices are non-linear in length and bandwidth. Usually, there is a basic fee for any communication; shorter trunks are much more expensive per mile than longer ones where you get a discount in length, etc. Since there are many recently founded new telecommunication providers, there is no hope any longer for simple price structures.

Suppose you have an enterprise with six offices in a country as shown in Fig. 3. From *c* to *b*, from *d* to *c*, from *f* to *d*, from *f* to *a*, and from *e* to *a* you need a communication bandwidth of 10 Mbps to satisfy your communication demands. The left-hand picture of Fig. 3 is then called your *demand graph*. You could now order the trunks exactly corresponding to this

demand graph to build your *communication network*. That solves your task immediately. However, it's easy to argue that there might be cheaper ways to link your offices. See the right-hand picture in Fig. 3. If we order these trunks from the telecom, the total trunk length is reduced from 885 units in the straightforward solution to 562 units. If we communicate from c to b in this new network, we will say our messages will be *routed* over f. We say, from c to b over f, our message is routed with two *hop counts*. Location f has to be provided with switching technology.

Network optimization is the mathematical problem to find the cheapest possible communication network for your demand graph. In the following we state the problem a bit more precisely and state some necessary conditions on solutions. It is convenient to state the demands as a matrix rather than to visualize them in a graph. For instance, the example problem in Fig. 3 has the demand matrix

$$D = \begin{pmatrix} \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ 0 & \cdot & \cdot & \cdot & \cdot & \cdot \\ 0 & 10 & \cdot & \cdot & \cdot & \cdot \\ 0 & 0 & 10 & \cdot & \cdot & \cdot \\ 10 & 0 & 0 & 0 & \cdot & \cdot \\ 10 & 0 & 0 & 10 & 0 & \cdot \end{pmatrix}.$$

The maximum hop counts may be restricted for every single demand, because, for example, telephone paths are routed better over less or equal than two hop counts because of possible echo effects. This is due to the delay caused by intermediate nodes. In such a case the hop count matrix H may look like

$$H = \begin{pmatrix} \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ 0 & \cdot & \cdot & \cdot & \cdot & \cdot \\ 0 & 2 & \cdot & \cdot & \cdot & \cdot \\ 0 & 0 & 2 & \cdot & \cdot & \cdot \\ 2 & 0 & 0 & 0 & \cdot & \cdot \\ 2 & 0 & 0 & 2 & 0 & \cdot \end{pmatrix}.$$

For every site of the network, we know if it may possess switching functionality or not. The vector $S = (1, 1, 1, 1, 1, 1)$ indicates for our small example that every site has a switching functionality (no switching functionality is denoted by 0).

For a network optimization problem we have to know further the vector P of coordinates of the geographical locations of the sites and the set of possible trunk bandwidths. In our example, the positions are given by $P = ((160, 60), (0, 50), (220, 210), (70, 0), (10, 180), (80, 80))$, and only trunks with a bandwidth of 34 Mbps may be used. The price or cost of a single 34-Mbps trunk is equal to its length, e.g., the euclidean distance between a pair of sites. In practice, the price for a trunk with a specified bandwidth depends on the distance in a more complex form.

Figure 3 (right) shows the optimum network for this problem. Topologically it consists of a so-called *star* of five links, each link containing a single 34-Mbps trunk. The numbers at the links in Fig. 3 denote the actual bandwidth needed. The routing of the demands in that network is graphically presented by paths of different colors. The routing matrix is given

by

$$R = \begin{pmatrix} \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \epsilon & \cdot & \cdot & \cdot & \cdot & \cdot \\ \epsilon & cfb & \cdot & \cdot & \cdot & \cdot \\ \epsilon & \epsilon & dfc & \cdot & \cdot & \cdot \\ efa & \epsilon & \epsilon & \epsilon & \cdot & \cdot \\ fa & \epsilon & \epsilon & fd & \epsilon & \cdot \end{pmatrix},$$

with ϵ denoting the empty path. By looking at a simple example we have defined the *network optimization* problem. In summary, its parameters are:

The Network Optimization Problem (NOP).
Input:

- a number of sites and their geographical coordinates
- switching facilities at the sites
- demand matrix with demand entries from site to site
- hop count matrix with hop count restrictions for each demand
- a set of trunk types (bandwidths) that can be ordered
- pricing table depending on bandwidth and distance

Output:

- graph of links consisting of certain trunks that meets all demands and restrictions
- routing matrix with route entries for every demand

Objective function: minimum price for an admissible network

That's the mathematical problem. In practice, our customers don't like very "abstract looking" or "mathematical looking" solutions which don't look "intuitive." So we have to rearrange the solutions a little bit. In addition, many alternatives are compared with different switching facilities to save network equipment.

Another serious property which is very often requested in practice is *redundancy*. Treating redundancy in network optimization is the consideration of possible failures of network components (trunks and machines). One would like to be able to run networks even in case of such failures. Therefore, many networks are built with one of the two following properties:

- for every demand there are defined two alternative routings which do not have any link in common
- for every demand there are defined two alternative routings which do not have any knot in common.

Networks with these features are called *link redundant* or *knot redundant*, respectively, if the following conditions are satisfied: If a link or knot fails and if in this case all affected routings of demands are changed to their alternative routing, then this new routing is an admissible solution for the original network problem. Note that knot redundancy implies link redundancy, so that knot redundancy is the harder restriction. In normal networks we have a trunk availability of more than 99%. Failure is therefore a rare case in terms of time (not in subjective anger, of course). In these few intermediate failure times it is certainly tolerable to use alternative routings with larger hop counts. For redundancy optimized networks the problem has, for this reason, a hop count matrix H^{ord} and H^{alt} for the ordinary case and for

the case of single failure, respectively. In our mini-example we use

$$H^{ord} = H = \begin{pmatrix} \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ 0 & \cdot & \cdot & \cdot & \cdot & \cdot \\ 0 & 2 & \cdot & \cdot & \cdot & \cdot \\ 0 & 0 & 2 & \cdot & \cdot & \cdot \\ 2 & 0 & 0 & 0 & \cdot & \cdot \\ 2 & 0 & 0 & 2 & 0 & \cdot \end{pmatrix}, \qquad H^{alt} = \begin{pmatrix} \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ 0 & \cdot & \cdot & \cdot & \cdot & \cdot \\ 0 & 3 & \cdot & \cdot & \cdot & \cdot \\ 0 & 0 & 3 & \cdot & \cdot & \cdot \\ 3 & 0 & 0 & 0 & \cdot & \cdot \\ 3 & 0 & 0 & 3 & 0 & \cdot \end{pmatrix}.$$

Current network optimization tools approach the problem in two steps. First, a good basic network is designed. Then all the links necessary to achieve redundancy are added in a clever way. Figure 4 (left) shows the best solution we obtained for this kind of approach. To the "star-like" solution for the basic network we added redundancy ending with a total length of 1048. When we optimize the network just *all-in-one*, in a one-step approach we can achieve a much better solution of total length 861 for this small example, which means a 21.7% improvement. The routing of Fig. 4 (right) is given by the matrices

$$R^{act} = \begin{pmatrix} \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \epsilon & \cdot & \cdot & \cdot & \cdot & \cdot \\ \epsilon & c-e-b & \cdot & \cdot & \cdot & \cdot \\ \epsilon & \epsilon & d-a-c & \cdot & \cdot & \cdot \\ e-c-a & \epsilon & \epsilon & \epsilon & \cdot & \cdot \\ f-a & \epsilon & \epsilon & f-d & \epsilon & \cdot \end{pmatrix},$$

and

$$R^{red} = \begin{pmatrix} \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \epsilon & \cdot & \cdot & \cdot & \cdot & \cdot \\ \epsilon & c-a-d-b & \cdot & \cdot & \cdot & \cdot \\ \epsilon & \epsilon & d-b-e-c & \cdot & \cdot & \cdot \\ e-b-d-a & \epsilon & \epsilon & \epsilon & \cdot & \cdot \\ f-d-a & \epsilon & \epsilon & f-a-d & \epsilon & \cdot \end{pmatrix},$$

with ϵ denoting an empty path. Dotted lines indicate the redundant routes of the demands. The numbers at the links give the actual bandwidth used. Note that every demand in our example is 10 Mbps and look at Fig. 4 (right). From d to a, there are now five different routings, each with a demand of 10 Mbps, over a single 34-Mbps line. This doesn't seem admissible at first sight. Let us give a clarifying argument. For example, the yellow routing and the green routing don't have a link or an intermediate node in common. Thus, if there is a single network failure, the dotted yellow and the dotted green line will never be used simultaneously. If you go carefully through these arguments you see that the solution is in fact feasible. In addition, you see that the simple mini-network we present here for an introduction already is an intriguing and intricate network optimization problem if redundancy is considered.

The Network Optimization Problem with Redundancy (NOPR).
Additional Input:

- kind of redundancy (knot or link)
- hop count constraints for alternative routings

Additional Output:

- routing matrix with failure-case alternative routes for every demand

B. *Ruin and Recreate*

1. Starting solution. The construction of a feasible solution is easy, because you obviously always can construct a very expensive solution which satisfies all aforementionend constraints.

2. Ruin. Ruin means that certain demands are removed from the system. This involves downsizing the bandwidth of those links that are used by the active and redundant path of each demand having been removed.

3. Recreate—Collective recreate. Normal best insertion should be clear. A demand to be inserted in chosen. Its active path is inserted in the cheapest possible way, then its redundant path is added in a cost-optimal manner. However, for network optimization we observed a rather frequent difficulty which we have to overcome by a somewhat more intelligent recreate technique: *collective best insert*. Let us explain what's not easy with single best insertion. Look at Fig. 5 (left), a starting solution of a simple network optimization problem: Two demands, from a to b, and from c to e, 10 Mbps each, three hops allowed, only 34-Mbps trunks available. The coordinate vector is $P = ((41, 0), (0, 0), (41, 8), (38, 4), (0, 8), (3, 4))$. If you ruin this starting solution by deleting one or both demands you will always return to the starting solution. Obviously, however, the right-hand network of Fig. 5 is a much better solution. In order to overcome the deficiency of the single best insertion strategy it is necessary to do further research on more sophisticated recreate techniques.

At present we are quite successfully experimenting with strategies which could be called "collaborative insertions." We don't yet have a simple or clean technique, which could be nicely communicated here. Let us just give a rough idea how we work for our current customers in practice. We try to group disintegrate demands which have to connect along "similar directions" in the plane. In the trivial example of Fig. 5 we would feel that the demands from e to c and from a to b are, viewed geographically, "in a similar direction." We insert such demands again in a better, collaborative way by lowering artifically the cost of trunks in these directions. This means in the example that we assume that trunks in the direction from a to b, or, for instance, trunks from f to d, are offered for a lower price or even for free. A lower price in a promising direction encourages the algorithms to choose solutions which look like they are collaborative even though they are computed independently.

C. *Details of the Implementation*

How to reinsert the active and redundant path of a demand into the design is being decided by length-restricted cheapest path algorithms. We obtain the length restriction by the active and redundant hop count restriction of a demand. The edge costs for the cheapest path graph algorithm are defined to be the additional costs of transport of the demand's bandwidth. Therefore the edge cost determination incorporates the following two problems.

1. Bandwidth calculation. For the NOP, the determination of the bandwidth for a given link, sufficient for dealing with its actual design bandwidth and the bandwidth of the demand to be reinserted, is an easy task: just add both values.

The problem becomes more difficult in case of the NOPR as already seen in the discussion of the bandwidth of link a–d in the previous example. Here a *conflict graph* is determined

with vertices for each demand's redundant path and edges between two vertices, if the corresponding redundant paths may be needed at the same time due to some network failure, i.e., the corresponding active paths share a link (or an internal location in case of knot redundancy).

The task now is to determine groups of isolated vertices, such that the sum of the group's maximum demands is minimal. The maximum bandwidth of such a group is sufficient for the whole group, since by the group's definition at most one of its bandwidths is needed due to network failure at the same time. Since in the non-failure case all active demands use their bandwidth, the sum of a link's active demands needs to be allocated. Since there is no interaction of the active paths and the redundant paths over the same link, the bandwidth B necessary for a link L with a set of active demands L_{act} and redundant demands L_{red} is given by

$$B = \sum_{D_{ij} \in L_{act}} D_{ij} + \min \left\{ \sum_{i=1}^{k} \max\{d \mid d \in J_i\} \mid L_{red} = J_1 \dot{\cup} \ldots \dot{\cup} J_k,\ J_i \ \textit{conflict free} \right\}. \quad (5.1)$$

The problem of determining the partition of the demands above is NP-complete by itself: if all redundant demands are the same it is the minimum-clique-cover problem on the complementary graph of the conflict graph. We therefore use a *first fit decreasing* like heuristic to deal with bandwidth calculation efficiently. Exact algorithms are out of scope since there are cases with often more than 30 redundant paths over a certain network link and the calculation must be done very fast.

2. Trunk set cost determination. Typical telecommunication providers have tariffs depending on a mixture of link distance, tariff zones, and bandwidths to be transported. We used a memory-based tariff database to deal with the efficient answering of questions of the following type: For a given bandwidth B, a distance between two locations dist and a tariff zone Z, what is the cheapest arrangement of trunks satisfying the needs and how much does it cost? Since these calculations should take only a very small part of computing time (we want to optimize), we use a hashing approach to guarantee that each call to the database is calculated the first time only. The actual calculation consists of answering the following (NP-complete) problem:

Problem. Minimum Weighted Cover

Instance. A set of trunks $T = \{(v_1, c_1), \ldots, (v_k, c_k)\}$, $v_i, c_j \in \mathcal{N}_+^0$, $1 \le i, j \le k$, each being a volume/cost pair and a volume $V \in \mathcal{N}_+^0$.

Question. Find a set $I \subseteq T$ with $\sum_{(v,c)\in I} v \ge V$ and $\sum_{(v,c)\in I} c$ minimal.

Normally the number of different trunks is not small since each different CIR (commited information rate) of a trunk with a certain bandwidth (for links with service included) gives different pairs to the above problem.

The actual calculation is done with an efficient branch and bound algorithm, making it possible to answer, e.g., all 650,000,000 calls to the database during a 12-h optimization run with a total of 1 minute of CPU time.

D. *Real Life Example*

In this section we would like to publish a network example for further research. We are particularly interested in how your algorithms perform on this problem. The example $N15$

we propose consists of 15 nodes, one of them acting as a "center." There is no "any-to-any" communication in this network but only an "any-to-center" communication. This is a very typical case in reality. Branch offices of insurances or banks communicate with the center and usually not among each other. Example $N15$ describes such a real life case. In our optimization service for our customers we are frequently asked what the synergy effects might be if networks are managed jointly. For example, three banks with centers nearby (in a large city) are connected to their branch offices in the surrounding region. If they manage their three networks separately it will cost them some amount x for the sum of the cost of their networks. Suppose they decide to operate a *joint* network. Then, of course, the resulting network can be designed cheaper than the sum of their original costs. If the joint network is y% cheaper than the original sum, we say the synergy is y%. We present here such a synergy problem. From $N15$ we constructed two equivalent networks by translating the original network by a vector in a plane. This way we have two equivalent copies of the original $N15$. Consider now the new network of 45 nodes which is generated by three $N15$ copies. This new "synergy problem" we call $N45$ (Fig. 6). The task is to optimize $N15$ alone, and then try to compute the best synergy network $N45$. What is the resulting synergy?

Below we provide the problem description (see Tables VI–IX).

System N15.

Input:

- 15 locations, 11 locations with switching facilities
- one demand (32-96 Kbps) from each location to the "computing center" (a single destination)
- knot redundancy for all demands
- maximum hop count of 3 for active and redundant path
- trunk bandwidths of 64 Kbps, 128 Kbps, and 2 Mbps can be ordered
- costs are 1, 2, and 3 units per length unit, respectively

TABLE VI
Parameters of System *N15* and Optimization Results

Node	x	y	S	D	C	r_{ord}	r_{alt}
1	715	488	1	—	—	—	—
2	200	450	1	32	1	2-1	2-6-10-1
3	771	47	0	32	1	3-1	3-4-1
4	818	214	1	64	1	4-10-1	4-1
5	833	511	1	96	1	5-1	5-14-8-1
6	320	205	1	32	1	6-10-1	6-13-2-1
7	982	401	0	64	1	7-5-1	7-4-1
8	626	872	1	96	1	8-1	8-14-5-1
9	385	613	1	32	1	9–1	9-12-2-1
10	587	265	1	64	1	10-1	10-6-2-1
11	858	789	0	64	1	11-1	11-14-5-1
12	64	494	1	32	1	12-9-1	12-2-1
13	77	68	1	96	1	13-6-10-1	13-2-1
14	746	878	1	64	1	14-1	14-5-1
15	125	260	0	96	1	15-6-10-1	15-2-1

Note. Listed are the coordinates x and y of the single nodes, the switch facility S, its bandwidth need (in Kbps) to the computing center C, and the optimized routings r_{ord} and r_{alt} for the normal case and for the failure case, respectively.

TABLE VII
The 24 Links Used in the Optimized System *N15*

Link	b_{ord}	b_{alt}	b_t	Costs
1-2	32	288	2000	1551
1-3	32	0	64	445
1-4	0	64	64	293
1-5	160	96	2000	363
1-8	96	96	2000	1185
1-9	64	0	64	353
1-10	352	32	2000	774
1-11	64	0	64	334
1-14	64	0	64	392
2-6	0	64	64	273
2-12	0	64	64	143
2-13	0	128	128	804
2-15	0	96	128	410
3-4	0	32	64	174
4-7	0	64	64	249
4-10	64	0	64	237
5-7	64	0	64	186
5-14	0	96	128	756
6-10	224	64	2000	822
6-13	96	32	128	558
6-15	96	0	128	406
8-14	0	96	128	242
9-12	32	32	64	343
11-14	0	64	64	144

Note. Listed for each link is the accumulated bandwidth b_{ord} due to the normal routings r_{ord}, the accumulated bandwidth b_{alt} due to the routings in case of failure r_{alt}, the bandwidth b_t of the trunk needed, and its cost. All bandwidths are given in Kbps.

- distance between two nodes is euclidean (rounded up to integer)
- specification in Table VI

Output:

- graph of links consisting of certain trunks (Table VII)
- routing matrix with route entries for every demand (Table VI)

Best solution: minimum sum of trunk costs found is 11437 units

System N45.
Input:

- 45 locations, 33 locations with switching facilities
- consisting of three *N15* subsystems with translation vectors of (0, 0), (0, 200), and (−200, 0). Details in Table VIII

Output:

- graph of links consisting of certain trunks (Table IX)
- routing matrix with route entries for every demand (Table VIII)

Best solution:

- minimum sum of trunk costs found is 25304 units
- synergy, compared to three isolated *N15* systems, is 26.3%

TABLE VIII
Parameters for System *N45* and Optimization Results

Node	x	y	S	D	C	r_{ord}	r_{alt}
1	715	488	1	—	—	—	—
2	200	450	1	32	1	2-36-31-1	2-32-28-1
3	771	47	0	32	1	3-19-1	3-10-1
4	818	214	1	64	1	4-19-1	4-34-1
5	833	511	1	96	1	5-1	5-20-16-1
6	320	205	1	32	1	6-36-28-1	6-40-10-1
7	982	401	0	64	1	7-19-1	7-5-1
8	626	872	1	96	1	8-16-1	8-38-31-1
9	385	613	1	32	1	9-31-1	9-21-28-1
10	587	265	1	64	1	10-25-31-1	10-1
11	858	789	0	64	1	11-20-16-1	11-5-1
12	64	494	1	32	1	12-42-28-1	12-39-31-1
13	77	68	1	96	1	13-36-31-1	13-28-1
14	746	878	1	64	1	14-8-16-1	14-20-5-1
15	125	260	0	96	1	15-28-1	15-36-31-1
16	715	688	1	—	—	—	—
17	200	650	1	32	16	17-24-8-16	17-39-31-16
18	771	247	0	32	16	18-19-1-16	18-5-20-16
19	818	414	1	64	16	19-1-16	19-25-31-16
20	833	711	1	96	16	20-16	20-5-1-16
21	320	405	1	32	16	21-28-1-16	21-9-31-16
22	982	601	0	64	16	22-5-1-16	22-20-16
23	626	1072	1	96	16	23-8-16	23-38-31-16
24	385	813	1	32	16	24-8-16	24-38-31-16
25	587	465	1	64	16	25-31-16	25-19-1-16
26	858	989	0	64	16	26-8-16	26-20-16
27	64	694	1	32	16	27-38-8-16	27-39-31-16
28	77	268	1	96	16	28-1-16	28-36-31-16
29	746	1078	1	64	16	29-16	29-14-20-16
30	125	460	0	96	16	30-28-1-16	30-36-31-16
31	515	488	1	—	—	—	—
32	0	450	1	32	31	32-2-36-31	32-28-1-31
33	571	47	0	32	31	33-6-36-31	33-34-1-31
34	618	214	1	64	31	34-25-31	34-1-31
35	633	511	1	96	31	35-25-31	35-1-31
36	120	205	1	32	31	36-31	36-28-1-31
37	782	401	0	64	31	37-19-1-31	37-25-31
38	426	872	1	96	31	38-31	38-8-16-31
39	185	613	1	32	31	39-27-38-31	39-31
40	387	265	1	64	31	40-31	40-10-1-31
41	658	789	0	64	31	41-8-16-31	41-38-31
42	−136	494	1	32	31	42-28-1-31	42-12-39-31
43	−123	68	1	96	31	43-13-36-31	43-28-1-31
44	546	878	1	64	31	44-8-16-31	44-38-31
45	−75	260	0	96	31	45-28-1-31	45-36-31

Note. Listed are the coordinates x and y of the single nodes, the switch facility S, its bandwidth need D (in Kbps) to the computing center C, and the optimized routing r_{ord} and r_{alt} for the normal case and for the failure case, respectively.

TABLE IX
The 79 Links Used in the Optimized System *N45*

Link	b_{ord}	b_{alt}	b_t	Costs	Link	b_{ord}	b_{alt}	b_t	Costs
1-5	160	160	2000	363	13-43	96	0	128	400
1-10	0	64	64	258	14-20	0	64	64	189
1-16	608	96	2000	600	14-29	0	64	64	200
1-19	320	64	2000	381	15-28	96	0	128	98
1-28	512	288	2000	2025	15-36	0	96	128	112
1-31	416	224	2000	600	16-20	160	160	2000	363
1-34	0	64	64	291	16-29	64	0	64	392
1-35	0	96	128	172	16-31	192	288	2000	849
2-32	32	32	64	200	17-24	32	0	64	247
2-36	64	0	64	258	17-39	0	32	64	40
3-10	0	32	64	286	18-19	32	0	64	174
3-19	32	0	64	370	19-25	0	64	64	237
4-19	64	0	64	200	19-37	64	0	64	39
4-34	0	64	64	200	20-22	0	64	64	186
5-7	0	64	64	186	20-26	0	64	64	280
5-11	0	64	64	280	21-28	32	32	64	279
5-18	0	32	64	272	23-38	0	96	128	566
5-20	0	128	128	400	24-38	0	32	64	72
5-22	64	0	64	175	25-31	288	128	2000	228
6-33	32	0	64	297	25-34	64	0	64	253
6-36	64	0	64	200	25-35	96	0	128	132
6-40	0	32	64	90	25-37	0	64	64	206
7-19	64	0	64	165	27-38	64	0	64	404
8-14	64	0	64	121	27-39	32	32	64	146
8-16	544	96	2000	615	28-30	96	0	128	396
8-23	96	0	128	400	28-32	0	64	64	198
8-24	64	0	64	249	28-36	32	96	128	154
8-26	64	0	64	260	28-42	64	0	64	311
8-38	32	96	128	400	28-43	0	96	128	566
8-41	64	0	64	89	28-45	96	0	128	306
8-44	64	0	64	81	30-36	0	96	128	512
9-21	0	32	64	218	31-36	320	384	2000	1458
9-31	32	32	64	181	31-38	128	352	2000	1185
10-25	64	0	64	200	31-39	0	64	64	353
10-40	0	64	64	200	31-40	64	0	64	258
11-20	64	0	64	82	33-34	0	32	64	174
12-39	0	64	64	170	36-45	0	96	128	406
12-42	32	32	64	200	38-41	0	64	64	247
13-28	0	96	128	400	38-44	0	64	64	121
13-36	192	0	2000	432					

Note. Listed for each link is the accumulated bandwidth b_{ord} due to the normal routings r_{ord}, the accumulated bandwidth b_{alt} due to the routings in case of failure r_{alt}, the bandwidth b_t of the resulting trunk needed, and its cost. All bandwidths are given in Kbps.

VI. SYSTEMATIC STUDIES ON THE TSP

In this section we present R&R for the Traveling Salesman Problem PCB442, based on single configurations, such that decision rules as SA, TA, GRE (and even Random Walk) are applicable to guide the search in the configuration space. We will provide results for all important parts of an optimization run using the R&R method. We will then compare these results to well-known results for local search optimization using Lin-2-opt as mutation.

A. *Initial Solution*

Usually, a random configuration serves as a starting point for an optimization run with local search using "non-intelligent" mutations of small order. This choice corresponds to a typical solution produced by non-intelligent mutations in a Random Walk. In contrast, the R&R optimization run starts with a configuration resulting from a creation of the whole system ($R\&R_{all}$). Figure 7 compares the distribution of randomly created configurations to the distribution of the R&R starting configurations. The mean length of a random configuration is approximately 770,000. This compares to the mean length of a $R\&R_{all}$ configuration of 58,140, which is only 15% above the optimum solution (50,783.5). This is a first advantage of the R&R method: the optimization starts much closer to the optimum. We therefore save the calculation time that an optimization run with small mutations needs to descend from these high values of a random solution and can concentrate on reducing this "15%."

B. *Optimization Run*

A typical Monte Carlo optimization run using the meta-heuristics SA, TA, or GDA starts with a Random Walk, and ends with Greedy Acceptance. Here, we want to discuss the particular behavior of the R&R mutations for these marginal decision rules. Finally, we present results with TA.

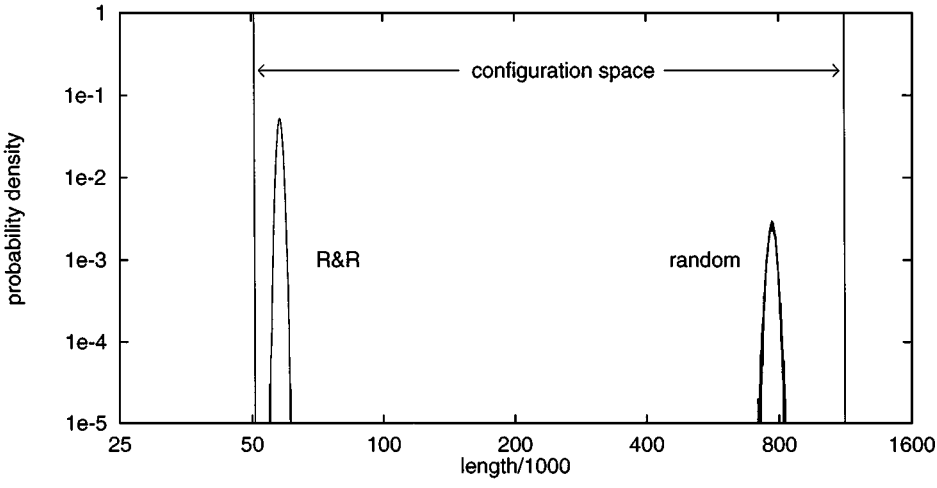


FIG. 7. Distribution of the lengths received both for 100,000 randomly created solutions and for 100,000 solutions generated with the best-insertion heuristics for the PCB442 problem. Left border, minimum at 50,783.5; right border, maximum tour length is approximately 1,130,000. Both distributions are obviously log-gaussian; the distribution generated by the best-insertion heuristics is much smaller and has therefore a higher peak, furthermore, its peak is close to the optimum.

1. Random Walk. It can be easily shown that a Random Walk using mutations of small order without "intelligence" like Lin-2-opt produces the same distribution of configurations as a random generation of configurations (Fig. 7). However, one might ask whether R&R mutations with $\mathcal{A} < \mathcal{N}$ change the distribution of the lengths of the solutions generated by $R\&R_{all}$. Figure 8 displays the distributions resulting from a $R\&R_{all}$ followed by 100

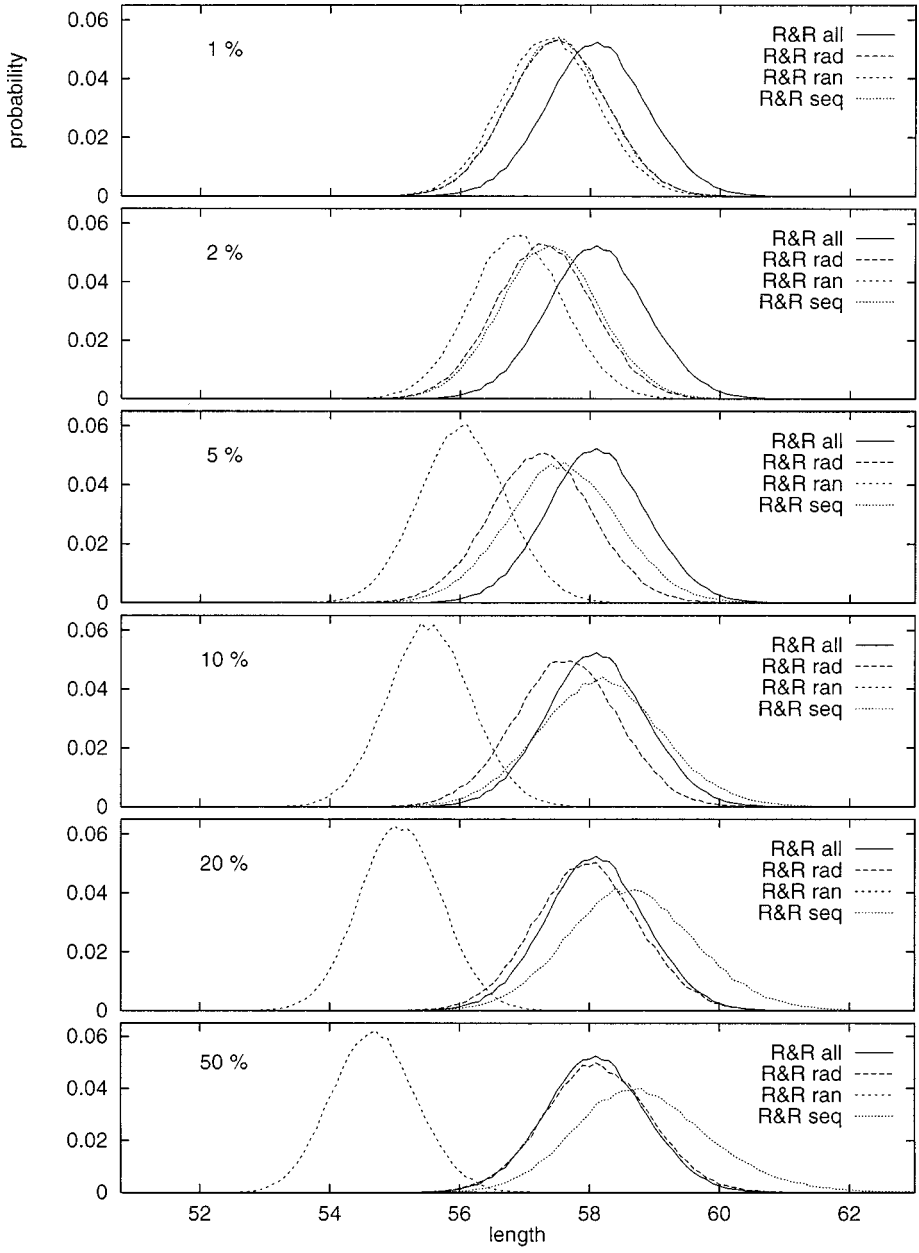


FIG. 8. Distributions of the lengths received for 100,000 solutions generated with the best-insertion heuristics and altered with 100 mutations in a Random Walk. In each figure, results for different ratios $\mathcal{F}$ are provided. We realize that for not too large $\mathcal{F}$ the peaks of the gaussian distributions are always displaced to better values. Using $R\&R_{rad}$ or $R\&R_{ran}$ one gets better results; however, the quality of the solutions is getting worse with $R\&R_{seq}$ using a large $\mathcal{F}$.

mutations in a Random Walk for different values of $\mathcal{F}$. We find that the distributions differ significantly. For small $\mathcal{F}$ the solutions are improved, which is quite clear. Due to the best-insertion strategy the system shows a Greedy Acceptance-like behavior. For a large $\mathcal{F}$ we find different results for the 3 mutation types: $R\&R_{ran}$ provides the best results for larger $\mathcal{F}$, $R\&R_{rad}$ hardly changes the distribution, and $R\&R_{seq}$ worsens the solutions. This can also be seen in Table X, which shows statistics corresponding to Fig. 8.

These results can easily be explained by the fact that $R\&R_{ran}$ can make best use of the remaining tour, since the removed nodes are uniformly distributed over the system,

TABLE X

Results for Creating a Starting Configuration with $R\&R_{all}$ and Altering It by 100 Mutations, at the Top If Using Random Walk, at the Bottom Working with Greedy

Acceptance	Ruin	$\mathcal{F}$	$\langle l \rangle$	Δl	l_{min}	l_{max}
RW	rad	1	57546	2.4	54342	61260
		2	57345	2.4	54096	61024
		5	57299	2.5	53928	60823
		10	57692	2.5	53712	61400
		20	58018	2.6	54376	61781
		50	58174	2.6	54975	61862
	ran	1	57423	2.3	54179	60908
		2	56918	2.3	53495	60071
		5	56089	2.1	53103	59046
		10	55581	2.1	52779	58874
		20	55144	2.0	52522	58022
		50	54740	2.1	52265	58061
	seq	1	57543	2.4	54656	60984
		2	57427	2.4	54090	60790
		5	57636	2.7	54090	61894
		10	58227	3.0	53890	62841
		20	58712	3.1	55013	63692
		50	58901	3.4	55139	64824
	rad	1	57344	2.3	54269	60620
		2	56635	2.2	53682	59728
		5	55425	2.0	52588	58642
		10	54762	1.8	52610	57985
		20	54490	1.6	52319	56634
		50	54758	1.5	52642	56623
GRE	ran	1	57425	2.3	54334	60864
		2	56905	2.3	53723	60290
		5	56045	2.1	53256	58973
		10	55488	2.1	52828	58628
		20	55028	2.0	52510	57881
		50	54520	2.0	51896	57278
	seq	1	57319	2.3	54124	60576
		2	56564	2.2	53868	59402
		5	55314	2.1	52719	58206
		10	54736	1.9	52510	57220
		20	54634	1.8	52482	57033
		50	55024	1.6	53045	56976

such that a good framework remains for their reinsertion. R&R$_{rad}$ completly reconstructs the system within a disc having only a few clues into the surrounding of the limiting circle. The ruin part of R&R$_{seq}$ produces a long edge in the system. Here, nodes are often inserted into other edges, such that this mutation frequently results in a long edge. Therefore, this mutation provides an example for a bad combination of a ruin and a recreate. Note that we do not speak of a bad sequential ruin; only the combination does not provide good results. For example, the combination of a sequential ruin and a recreate reinserting the nodes only between the two limiting nodes of the long edge may provide good results.

2. Greedy Acceptance. For comparison, we now discuss analogous results with Greedy Acceptance. They are shown in Fig. 9 and Table X. First, we find that nearly all GRE results are better than the RW results. The largest differences are obtained for radial and sequential ruin, whereas the improvement is only small for random ruin. The results do not differ much for small $\mathcal{F}$ because here Greedy Acceptance and Random Walk coincide: A ruin of a single node followed by best-insert always results in the same or a better configuration, and for a few nodes only very small deteriorations can happen. Second, the GRE distributions show a sharper peak for larger $\mathcal{F}$, and the peak is shifted towards smaller lengths. However, using $\mathcal{F} = 0.5$ produces results worse than $\mathcal{F} = 0.2$ when working with sequential or radial ruins. The optimum fraction $\mathcal{F}$ depends on the kind of the ruin and is close to 0.2 for radial ruin.

3. Comparison: Random Walk/Greedy Acceptance. Until now, we have only seen a trend for the single mutations. But the journey to other values of the objective function has not yet ended after 100 mutations, both for RW and for GRE. Figure 10 displays the total length, averaged over 10 independent runs, as a function of the number of mutations applied. It can be clearly seen that R&R$_{rad}$ and R&R$_{seq}$ cannot improve the R&R$_{all}$ solutions in a Random Walk with a large $\mathcal{F}$; the corresponding runs with a small $\mathcal{F}$ show nearly the same behavior as their GRE counterparts. Using the random ruin we get a completely different behavior. As shown in Fig. 8 the mean length monotonously decreases for all values of $\mathcal{F}$. For Greedy Acceptance all curves decrease sigmoidally. The best results are achieved by using a large $\mathcal{F}$, because these mutations are able to reorder the system on a larger scale, and therefore to find better improvements. Again, radial and sequential ruin are more similar to each other than to random ruin. We have to mention that in spite of the huge amount of calculation time we invested for the results shown in Fig. 10, most graphs indicate that the single systems are still not equilibrated for a certain $\mathcal{F}$. Especially when using random ruin, further improvements could be found.

4. Threshold Accepting. Although R&R achieves quite good solutions even with Greedy Acceptance, we now want to provide results for combining R&R with TA. Figure 11 (left) displays the deviation δ of the length l from the optimum length l_{opt},

$$\delta = 100 \cdot \frac{\langle l \rangle - l_{opt}}{l_{opt}}, \tag{6.1}$$

as a function of the CPU time for 6 different cooling schedules, for Lin-2-opt mutations, and for R&R mutations, respectively. The average is taken over 20 runs. One second of CPU time corresponds to 100 R&R mutations with ($\mathcal{F} = 0.2$) and to 300,000 Lin-2-opt mutations. The following cooling schedules are used:

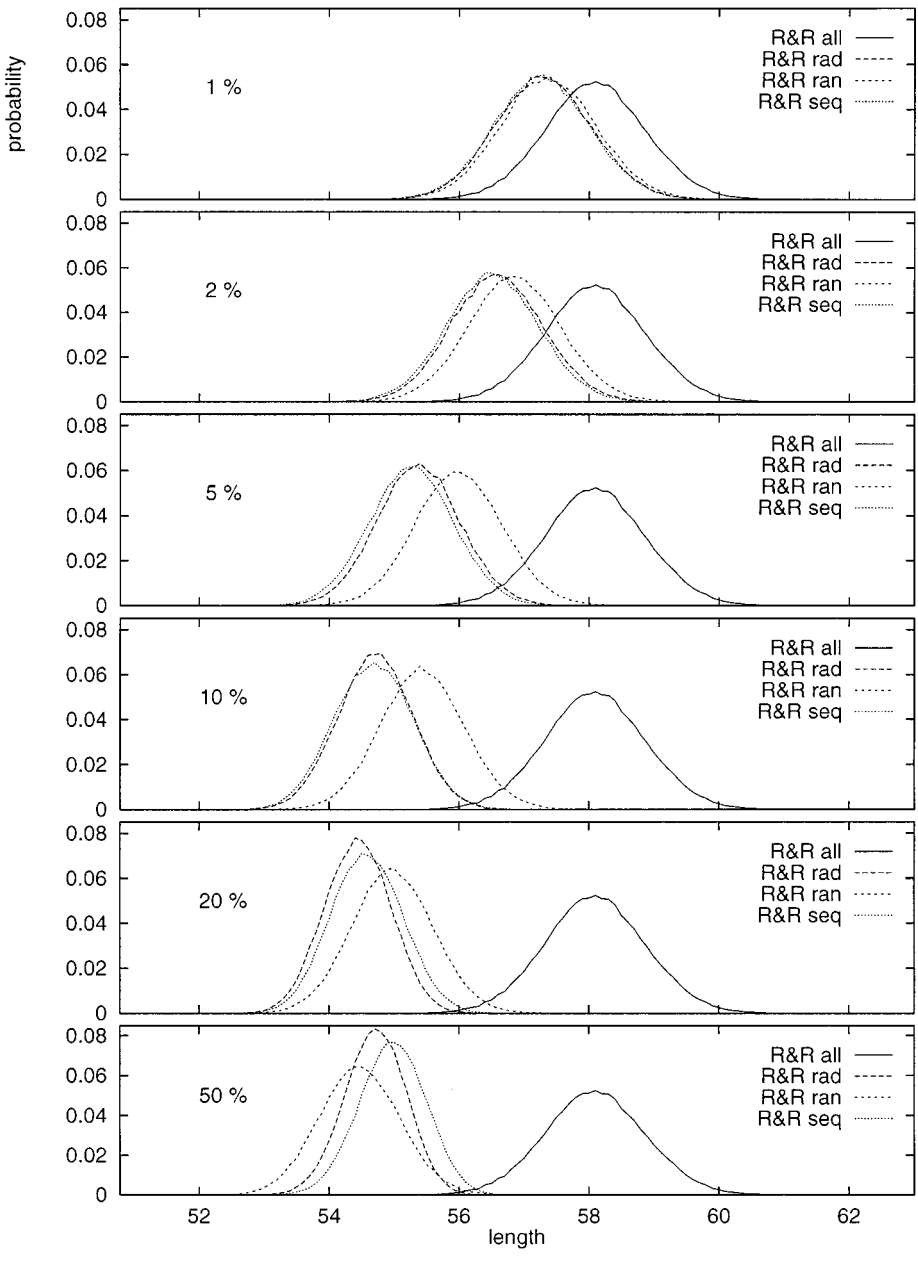


FIG. 9. Distributions of the lengths received for 100,000 solutions generated with the best-insertion heuristics and altered with 100 mutations using Greedy Acceptance. In each figure results for different ratios $\mathcal{F}$ are provided.

- Linear decay,

$$T = T_0 \cdot (1 - x). \tag{6.2}$$

- Exponential decay,

$$T = T_0 \cdot \exp(-\ln 2 \cdot x/\alpha). \tag{6.3}$$

with half-lives $\alpha = 0.2$, 0.4, 0.6, and 0.8.

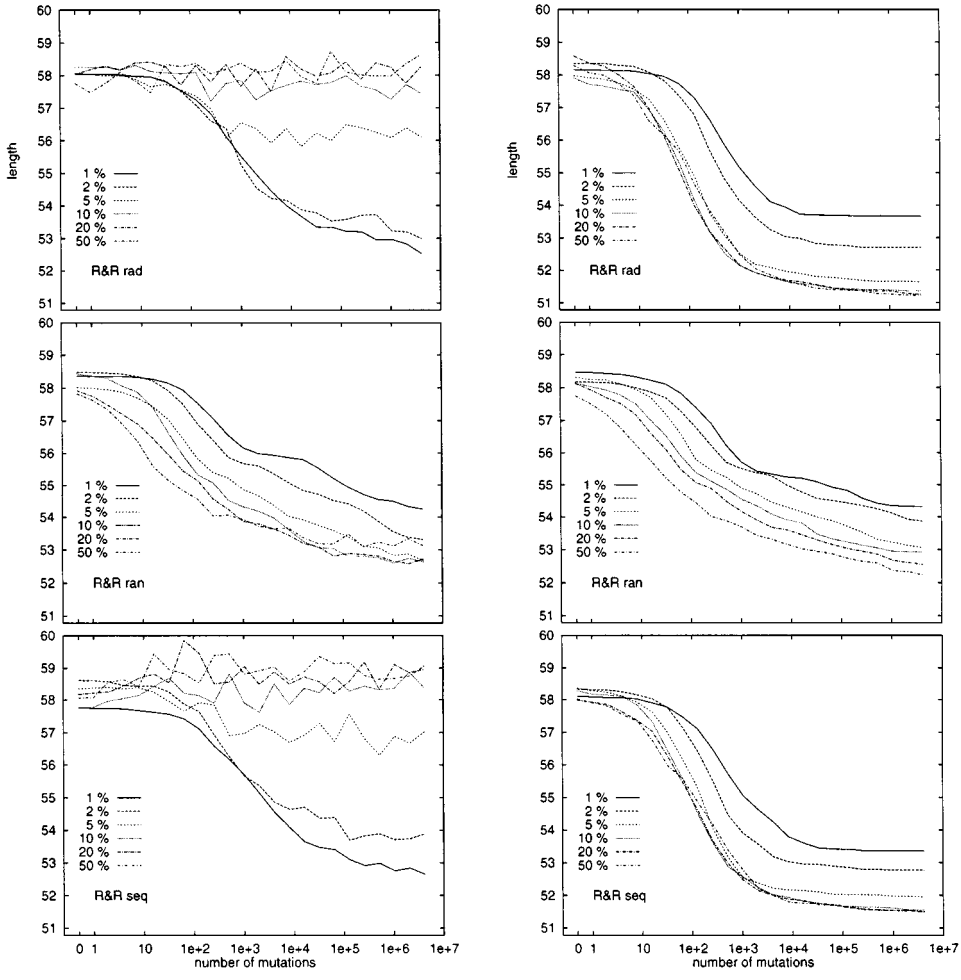


FIG. 10. Averaged length as a function of the number of the R&R mutations, starting from an $R\&R_{all}$ configuration for choosing radial, random, and sequential ruin with different fractions $\mathcal{F}$. Left, for Random Walk; right, for Greedy Acceptance.

- Greedy,

$$T = 0. \tag{6.4}$$

T is the instantaneous threshold, and x denotes the continuous schedule variable, increased from 0 to 1, which is equal to the ratio of the number of the current step to the total number of cooling steps. Each optimization run is terminated by an x-range $[1.0 : 1.1]$ using Greedy Acceptance to assure reaching a local optimum. The initial threshold T_0 for the cooling schedules is determined by the standard deviation of an initial Random Walk of 1000 mutations. Using R&R mutations with a $1 : 1$ mixture of random ruin and radial ruin with the best found fraction $\mathcal{F} = 0.2$ we obtained $T_0 = 230$. For Lin-2-opt mutations one thus gets about $T_0 = 980$. However, as proposed by Dueck *et al.* [3] we took $T_0 = 130$ for the Lin-2-opt mutations, which gives better results. Figure 11 (left) shows that the results for R&R are highly independent of the schedule applied. However, TA can improve the optimization result. On the other hand, the quality of the results is more sensitive to the schedule when working with the Lin-2-opt. Here, linear cooling and exponential cooling with a decay of

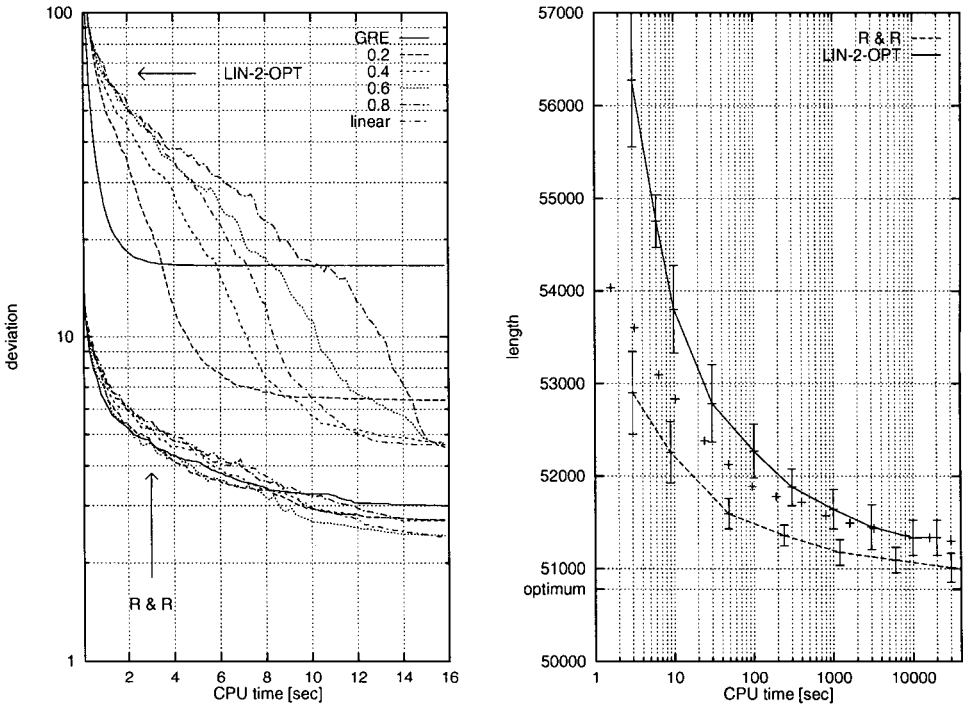


FIG. 11. Left, time development of the deviation, averaged over 50 runs, using R&R and Lin-2-opt with different cooling schedules (see text). Total CPU time for each run is 16 s. Right, optimization results, averaged over 50 runs, as a function of the total simulation time. The errorbars denote the standard deviations. Dashed, for R&R with linear cooling; solid, for Lin-2-opt with exponential cooling ($\alpha = 0.4$). Additionally, the average results for R&R with Greedy Acceptance are marked by $+$. Several times the optium value of 50,783.5 was reached using R&R with TA.

$\alpha = 0.4$ provide rather good results, close to those achieved with the cooling schedule proposed in an earlier paper [3]. Figure 11 (right) shows the results using the "optimum" value of $\alpha = 0.4$, comparing it with GRE results. In all cases the R&R is superior to Lin-2-opt.

VII. COMPUTATIONAL DETAILS

All optimization runs were performed on a RS 6000, model 43P, 233 MHz with 512MB memory. An important point to mention is the applicability of our approach to very large vehicle routing problems. This is due to the fact that inherently R&R is suitable for parallel execution. Most time consuming is the calculation of the cost of acceptance for a customer by a vehicle, especially for the recreate steps. The recreate steps consume about 90% of the whole computing time. These calculations can easily be parallelized on the vehicle basis, since the single vehicle calculations do not interdepend. This is even valid for the single tests on different potential positions inside a single vehicle tour.

VIII. CONCLUSION AND OUTLOOK

We have given computational evidence of the validity of the R&R principle. From the pure view of a mathematician, one might say, "OK, but this new method does work with larger exchange moves only, and everything is known and classical." Yes, the observation

that larger moves are essential for more difficult problems where a critical part of the task is to generate fully admissible solutions is new.

For these harder optimization problems our current approach or view seems to be really essential, scientifically and, above all, for our practical work. In our optimization services work for industrial customers we had a hard time giving solutions for flight scheduling, car sequencing, or steel mill optimization just by applying the classical simulated annealing or threshold accepting methods. When we worked with pure threshold accepting in the past years, we had the philosophy that the simplest moves are the best. Advice: use your CPU time for millions of simple and fast exchange moves! Do not try to make hybrid or "intelligent" exchange moves which consume much more CPU time. Thousands of simple moves should be better than a single complex move! This was our message that time. Time goes by, of course.

From a mathematician's point of view, as said, R&R is a generalization of SA or TA. For our thinking, however, R&R gives a different way to view difficult problems. Currently, we are scanning our optimization logbook for problems where we didn't really succeed. We'll try again, for instance, chip placement and scheduling. At present, we are successfully applying R&R to car sequencing: produce them in a smooth order, such that their individual features do not cause too much variance of the production time in each phase.

REFERENCES

1. M. Grötschel and O. Holland, Solution of large-scale symmetric travelling salesman problems, *Math. Prog.* **51**, 141 (1991).
2. G. Reinelt, TSPLIB95, University of Heidelberg, Germany, 1995.
3. G. Dueck and T. Scheuer, Threshold accepting: A general purpose optimization algorithm appearing superior to simulated annealing, *J. Comput. Phys.* **90**, 161 (1990).
4. G. Dueck, New optimization heuristics: The great deluge algorithm and the record-to-record travel, *J. Comput. Phys.* **104**, 86 (1993).
5. G. Dueck, T. Scheuer, and H.-M. Wallmeier, Toleranzschwelle und Sintflut: Neue Ideen zur Optimierung, *Spektrum Wissensch.* **3**, 42 (1993).
6. J. Schneider, Ch. Froschhammer, I. Morgenstern, Th. Husslein, and J. M. Singer, Searching for backbones—An efficient parallel algorithm for the traveling salesman problem. *Comput. Phys. Comm.* **96**, 173 (1996).
7. J. Schneider, M. Dankesreiter, W. Fettes, I. Morgenstern, M. Schmid, and J. M. Singer, Search space smoothing for combinatorial optimization problems, *Phys. A* **243**, 77 (1997).
8. J. Schneider, I. Morgenstern, and J. M. Singer, Bouncing towards the optimum: Improving the results of Monte Carlo optimization algorithms, *Phys. Rev. E* **58**, 5085 (1998).
9. N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller, and E. Teller, Equation of state calculation by fast computing machines, *J. Chem. Phys.* **21**, 1087 (1953).
10. S. Kirkpatrick, C. D. Gelatt, Jr., and M. P. Vecchi, Optimization by simulated annealing, *Science* **220**, 671 (1983).
11. K. Binder and D. W. Heermann, *Monte Carlo Simulation in Statistical Physics* (Springer-Verlag, New York, 1992).
12. M. Solomon, Algorithms for the vehicle routing and scheduling problem with time window constraints, *Oper. Res.* **35**, 254 (1987).
13. W. Schnabl, P. F. Stadler, C. Forst, and P. Schuster, Full characterization of a strange attractor: Chaotic dynamics in low-dimensional replicator systems, *Phys. D* **48**, 65 (1991).
14. E. Aarts and J. K. Lenstra, *Local Search in Combinatorial Optimization* (Wiley, Chichester, 1997).
15. S. Lin, Computer solutions to the traveling salesman problem, *Bell System Tech. J.* **44**, 2245 (1965).

16. S. Lin and B. W. Kernighan, An effective heuristic algorithm for the traveling salesman problem, *Oper. Res.* **21**, 498 (1973).
17. E. Schöneburg, F. Heinzmann, and S. Feddersen, *Genetische Algorithmen und Evolutionsstrategien* (Addison–Wesley, Bonn, 1994).
18. G. Reinelt, *The Traveling Salesman* (Springer-Verlag, Heidelberg, 1994).
19. S. R. Thangiah, I. H. Osman, and T. Sun, *Hybrid Genetic Algorithm, Simulated Annealing and Tabu Search Methods for Vehicle Routing Problems with Time Windows*, working paper, UKC/OR94/4, 1994.
20. Y. Rochat and E. Taillard, Probabilistic diversification and intensification in local search for vehicle routing, *J. Heur.* **1**, 147 (1995).
21. M. Desrochers, J. Desrosiers, and M. Solomon, A new optimization algorithm for the vehicle routing problem with time windows, *Oper. Res.* **40**, 342 (1992).
22. J.-Y. Potvin and S. Begio, *A Genetic Approach to the Vehicle Routing Problem with Time Windows*, Publication CRT-953, Centre de recherche sur les transports, Universite de Montreal, 1994.
23. G. Dueck and J. Wirsching, in *Proceedings of the Fourth European Conference on Mathematics in Industry*, edited by H. Wacker and W. Zulehner (Teubner and Kluwer Academic, Netherlands, 1991).
24. W.-C. Chiang and R. Russell, *Simulated Annealing Metaheuristics for the Vehicle Routing Problem with Time Windows*, working paper, Department of Quantitative Methods, University of Tulsa, Tulsa, OK, 1993.