Biol. Cybern. 64, 7–14 (1990)

Biological Cybernetics
© Springer-Verlag 1990

The "molecular" traveling salesman

W. Banzhaf*

Central Research Laboratory, Mitsubishi Electric Corporation, 1-1, Tsukaguchi Honmachi 8-chome, Amagasaki, Hyogo, 661 Japan

Received March 27, 1990/Accepted in revised form May 30, 1990

Abstract. We consider a method for optimization of NP-problems motivated by natural evolution. The basic entity is a population of individuals searching in state space defined by the problem. A message exchange mechanism between individuals enables the system to proceed fast and to avoid local optima. We introduce the concept of isolated evolution to maintain a certain degree of variance in the population. The global optimum can be approached to an arbitrary degree. The method is applied to the TSP and its behavior is shown in a couple of simulations.

1 Introduction

In recent years interest has raised to apply search strategies motivated by natural evolution to complicated technical optimization problems. Though algorithms simulating evolution were proposed over at least 30 years (Bremermann et al. 1965; Rechenberg 1973; Holland 1975; Schwefel 1981), the application to combinatorial optimization problems only recently became an actual research topic. A large number of papers on evolutionary optimization of NP-problems has been published since (Brady 1985; Grefenstette et al. 1985; Boseniuk et al. 1987; Fontana and Schuster 1987; Wang 1987; Mühlenbein et al. 1988; Fogel 1989).

A classical problem of this kind is the "traveling salesman problem" (TSP) where the shortest tour on a given distribution of cities is looked for which visits all cities once and returns to the starting point (Lawler et al. 1985). This problem is of particular interest, since it is one member out of the class "NP-complete", the members of which are

i) transformable into each other
ii) classically unsolvable for higher dimensions.

The former means that if a method for the solution of one of these problems is known it naturally carries over to other problems of the class. The latter amounts to an exploding computation time (with an increase greater than any polynomial in the problem dimension N) which hinders an effective solution for arbitrary dimensions. The reason for this poor behaviour is the exponential growth of the number of locally optimal solutions with growing dimensionality.

A similar development like in the case of evolutionary algorithms happened after the proposition of another dynamical optimization method, the Monte-Carlo-algorithm originally suggested by Metropolis et al. (Metropolis et al. 1953) in the late forties. The breakthrough to applications on NP-problems has been achieved there after a seminal paper on simulated annealing (SA) by Kirkpatrick et al. (Kirkpatrick et al. 1983).

The advantage of evolutionary search methods compared to thermodynamical methods (like SA) is that the former ones are operating far from equilibrium. The SA method is based on the introduction of a parameter analogous to the temperature of a physical system. A stochastic relaxation process is created to reach Boltzmann distribution starting from any intial configuration in state space. By departing from a high value of the temperature parameter and slowly cooling it down, it is possible to find local or global minimal-energy-states of the simulated system. In this approach, however, one critical point is, that one has to wait during any cooling step until the equilibrium distribution of states is reached. Being clearly a time consuming process, this contrasts sharply to simulated evolution methods.

In natural evolution we find three main ingredients to guarantee the success of the optimization process which it embodies:
1.) randomness
2.) selection
3.) populations of searching subjects.
Let us discuss these points a bit further before coming to our concrete system.

* On leave from: Institut für Theoretische Physik und Synergetik, Universität Stuttgart, Pfaffenwaldring 57/IV, D-7000 Stuttgart 80, Federal Republic of Germany

Ad 1.): Randomness – often associated with mutational effects only – appears in nature in different phenomena: Firstly, as various sorts of mutations, i.e. point mutations, gene deletions, chromosome mutations, and so on, secondly, in random mating behavior of individuals and, thirdly, even as abrupt changes of environmental conditions. Consequently, local (diffusion or tunneling like), intermediate and global (Monte-Carlo-like) random effects are needed if a simulating system designed to optimize functions should reflect evolution.

Ad 2.): Limited availability forces the individuals of a species in nature to compete for resources. This gives rise to a selection of those individuals who are better suited for a given task. In a simulation system, the selective pressure is imposed by limited computational resources in terms of equipment (processor time, storage capacity) and this pressure in turn generates an uphill (downhill) motion in the abstract landscape of fitnesses (cost functions) to be searched. Though different scenarii are reasonable, the common effect of selection is comparable to gradient ascent (descent) in a simple landscape (i.e. without local optima). Progress velocity over time typically decays exponentially in systems with random forces working under a selection rule. This is an often observed phenomenon in evolutionary algorithms and it seems to be independent of the detailed structure of the mutational and/or selective operations applied.

Ad 3.): Nature generally adapts populations of individuals, not individuals themselves. This is the most important ingregient gleaned from natural evolution. The intrinsic parallelism (Holland 1975) of evolutionary algorithms stems from a parallel search in state space which is done simultaneously by an entire population of individuals. In order to profit from each others' knowledge, the individuals should be able to exchange messages – at least from time to time. Usually, the message exchange is done by applying recombination or cross-over operations between subjects. Yet, this results in a decreasing variance of the population. Since maintaining a considerable variance is of immense importance during the search for a global optimum, it turns out that the influence of the recombination frequency is a crucial parameter as far as the convergence behavior of the algorithms is concerned.

In the following we shall apply some of the operational recipes nature used in the molecular evolution of macromolecular strings. Our intention, however, is not to model closely natural evolution but to apply the principles to technical optimization problems. Thus, we are free to take advantage of arbitrary sets of operations existing in natural evolution. We shall combine operations suitable for solving the traveling salesman problem without consideration whether their actual combination occurs in nature or not.

More specifically, we shall study a population of data-strings searching the underlying state space of the problem by means of operations which a couple of machines (processors in technical systems) is applying on them. Data-strings are virtually realized in a computer, thus the algorithm is freed from the necessity to introduce more and more parallel processors.

2 The "molecular" system

The individuals in our system are pieces of data lumped together to form data-strings of a certain length. In a sense, they resemble the strings of macromolecules formed in prebiotic evolution. Every string carries *all* the informations necessary to put together a particular solution of the optimization problem we want to get solved. This principle of full representation is a necessary prerequisite to the totally local character of the search processes to be introduced below.

Like in macromolecular evolution a soup of organic material and strings of aminoacids was formed, the ensemble of all individuals in our data population forms a pool out of which specialized machines pick up exemplars to perform various sorts of operations. The machines – analoguous to enzymes working on macromolecules – operate serially on those strings they have isolated out of the pool, in a way determined by their respective internal cycle length and time scale. The requirement of minimal bias dictates that the machines' operations are controlled mainly by random number generators ensuring that various trials are made during the search process. Obviously, some knowledge must be incorporated into the action of machines, so that they are specialized in the sense of performing different sorts of operations which are usually adapted to the optimization problem to be solved. Figure 1 gives a sketch of the overall system.

A strict locality principle is invoked for the search process which states that every machines performs operations on the strings it has picked up and *does not care* about what the others are doing. This has as one of its immediate consequences that reproduction in our algorithm is *not* based on a global ranking of fitness

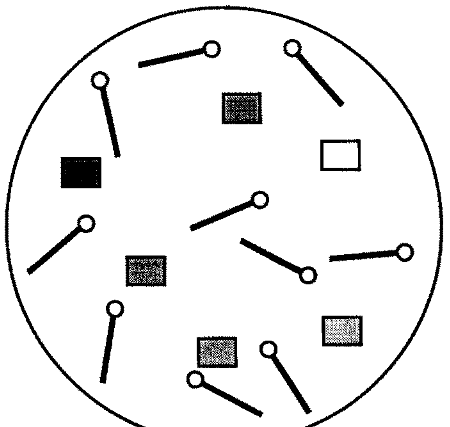


Fig. 1. The overall system, a pool of data-strings. Every string is equipped with all the information necessary to produce a solution. Especially, the strings carry their quality signal (circle). The strings are subject to changes imposed on them by machines (squares) picking them up at random

values in the entire population. Rather, the fitness value of the existing individual is compared to the trial solution generated by the action of the machine. On the *local* basis, a selection is performed in order to release only one string back into the pool. Another consequence is that at the same time even different problems of the same sort could be solved simultaneously.

More explicitely, a local operation cycle of one particular machine consists of the following serial steps:
1. picking up the number n_{op} of strings necessary to perform the prescribed operation,
2. performing the operation controlled by a random generator,
3. evaluating the newly generated strings(s) according to a predefined fitness criterion,
4. comparing it (them) to the original string(s),
5. releasing the n_{op} best one(s) into the population.

Let us now assume that we have a number, p, of distinct machines. We are free to apply as many copies of each machine as we want, with the following restriction, yet: The effective number of machines should be smaller than the number of data-strings in order to maintain a truly parallel search. We discriminate between the number of machines and the effective number of machines, since machines may act at different frequencies, and indeed this will be used later on extensively. We therefore define:

$$p_{eff} = \sum_{j=1}^{p} t_j \times m_j \tag{1}$$

m_j: number of machines of sort j.

A natural order in time for the succession of individuals is provided by a generation counter t. Since we are simulating the system on a serial computer, a generation is defined here as the number c of local operation cycles which suffices to give every string a chance to be changed, i.e. $t \rightarrow t+1$ after $c = \lfloor M/p_{eff} \rfloor$ operations were performed. ($\lfloor x \rfloor$ symbolizes the next smaller integer to x.)

Due to the independent operation of machines, the contribution of each sort of machines is distributed stochastically and reflects the success of the respective machine operation given the particular state of the optimization problem. The frequency of successfully applied operations, i.e. those which improve the string population, is left to be completely determined by the system itself. Thus the system is forced to self-organize in order to approach the global optimum.

To simplify things a bit, we restrict ourselves to

$$m_j = 1 \; \forall j.$$

The only parameters, then, to be chosen are t_j. Those will be the object of our particular attention below, since one of the time scale parameters is related to a possible isolation or non-isolation of individuals in the course of development.

The system we introduced so far is applicable to a large variety of optimization problems and it needs some further specification in terms of the operations applied to get it doing the concrete optimization task. Therefore, some knowledge about the problem domain and some intuition about possibly useful operations are necessary before starting.

Constraining ourselves to the TSP now, the particular system consists of tours symbolized by data-strings $\vec{s}$,

$$\vec{s} = \{s_0, s_1, \ldots, s_N\}$$

where

$$s_0 \in R^+$$

and

$$s_i \in \{1, \ldots N\}, \quad i = 1, \ldots, N$$

represents the actual position of one of the N cities on the tour $\vec{s}$. For explicational purposes, let $N = 4$, then the string

$$\vec{s} = (l, 1, 3, 4, 2)$$

represents the tour of visiting city 3 after city 1 and city 4 after city 3. The tour must be closed, so that city 1 is visited after city 2. The first component of the string, s_0, is the quality signal which serves as selection criterion. The TSP criterion is the distance between cities

$$l = \sum_{i=1}^{N} D_{s_i, s_{i+1}} \tag{2}$$

where $D_{i,j}$ is the euclidean distance between cities i and j, in our case 2-dimensional

$$D_{i,j} = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$$

and the cyclic constraint

$$s_{N+1} \equiv s_1$$

is invoked.
There are $N!$ different strings realizable with a certain geometric distribution of cities and the aim is to find those strings which have lowest tour-length.

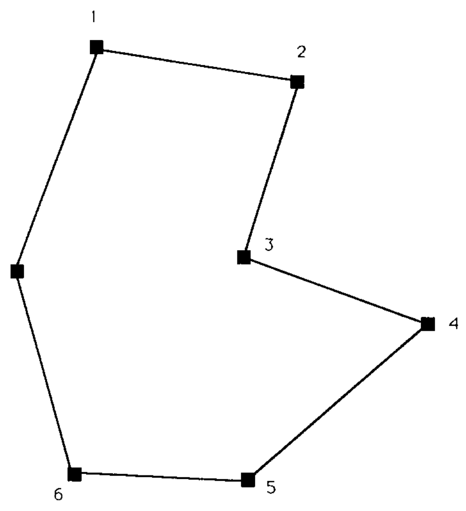
To make use of the intrinsic parallelism of evolutionary search we introduce a population of M data-strings (the pool)

$$\vec{s}^{\mu}(t), \quad \mu = 1, \ldots, M$$

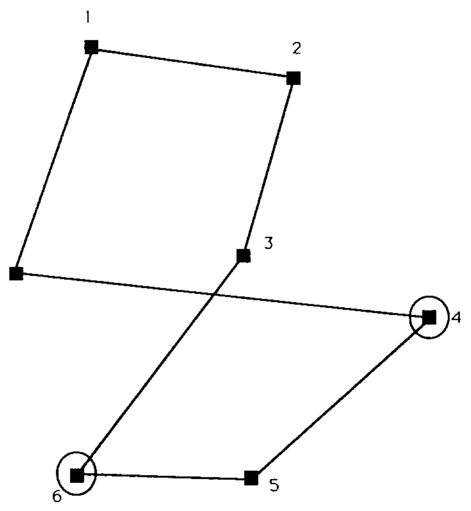
on which search operations are performed by various machines. Therefore, the datastrings are subject to changes and should be described by time dependent quantities $\vec{s}^{\mu}(t)$.

Our four sorts of machines are called the C-, I-, R- and E-machines performing cutting, inversion, recombination and exchange-2 operations. Depending on the size of the string population, machines of the same sort could act in a parallel manner.

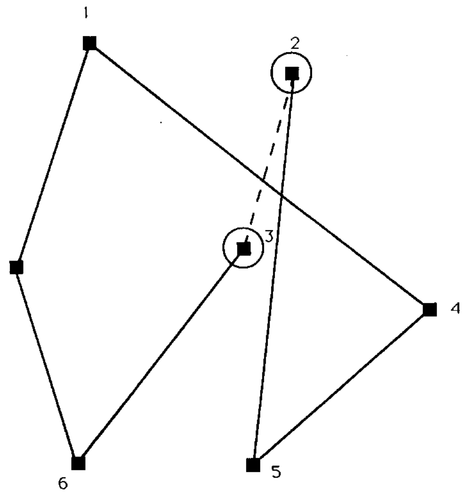
Let us now consider in more detail the different effects the machines have on strings:
The E-machine just fixes two (randomly chosen) cities and inverts their order in the tour, compare Fig. 2b.
The C-machine again fixes two cities and transposes the part of the tour lying inbetween them behind of a third (randomly chosen) city, compare Fig. 2c.



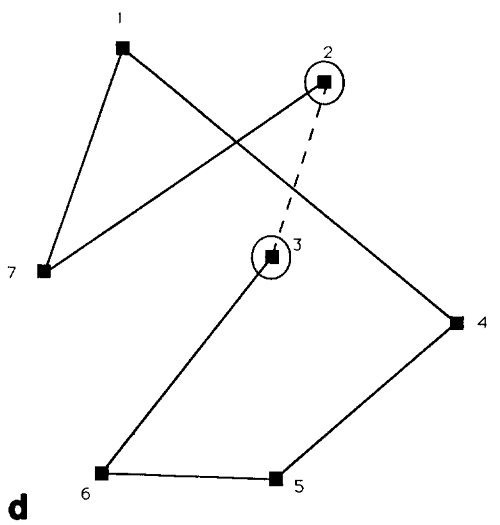
a



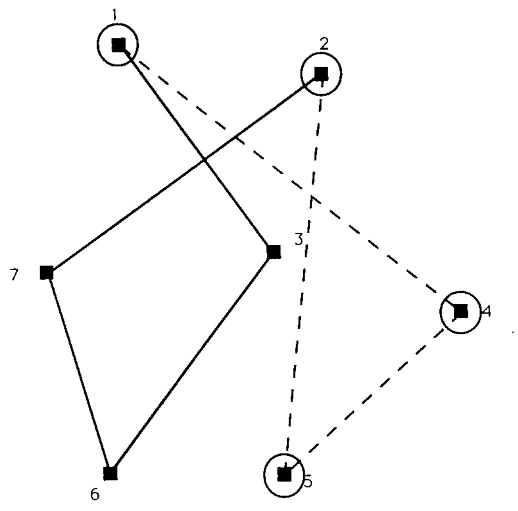
b



c



d



e

Fig. 2. a A sample traveling salesman tour. Original tour. **b–e** The actions of four different machines **b** Exchange; **c** Cut; **d** Cut-inverse; **e** Recombine on the sample tour

The I-machine performs the same operation as the C-machine, except that the part cut is inserted in inverted order, compare Fig. 2d.
The recombination machine takes two strings and cuts a randomly chosen part of the first string fixing it at the overlapping city of the second string. In order to guarantee a valid tour, all cities inserted from the first string and already present at the second must be deleted there simultaneously, compare Fig. 2e.

As may be seen here, it is the *local* activity of machines working on n_{op} strings without any reference to the global state of the system that leads to continuous progress in terms of a decreasing mean population tour-length $\langle l^{\mu} \rangle$.

3 Simulation

In this section we report on a simulation with the previously described system. Simulation 1 is aimed at a demonstration of different features of the algorithm. Simulation 2 studies the TSP with randomly distributed cities in greater detail.

Concerning the first simulation, we have chosen a particular toy problem, the global solution of which is well known: A ring of $N = 30$ cities (points) should be connected in the order of minimal tour-length. On a ring, the obvious solution is a connection of any city to its two next neighbors. Actually, the ring solution is the only one, so there are no local minima in this special case.

A population of $M = 9$ strings searches the state space of this problem. Initially, the data-strings do not possess any knowledge about the city-configuration. Rather, the initial tours are generated randomly. Figure 3a shows the resulting initial tours as represented by the data-strings. Figure 3b monitors the state of the strings after 1000 generations have passed. Still, all individuals perform their own way towards the optimal state.

In this experiment, we have fixed the time-scales of our different machines as follows:

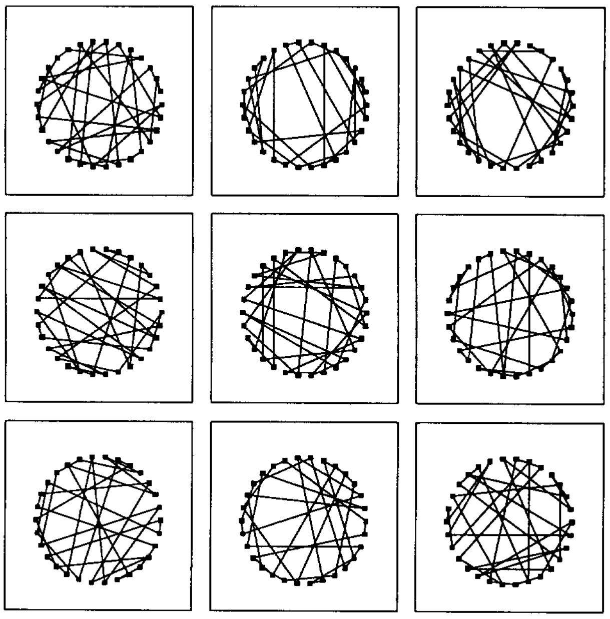
$$t_C = t_I = t_E = 1, \quad t_R = \frac{1}{100}.$$

This ensures that developments in one part of the state space are able to evolve before recombination unifies them with strings of other parts. In biology, there exists a similar mechanism which is called isolated evolution. It enables subpopulations to evolve rather independent before merging again to evolve in an entire population.

In order to see the effect of the different operators applied during search we performed 4 comparative runs for the same initial conditions but with a varying number of machines.

Table 1a displays the results for the ring-distribution of cities. As can be seen immediately, the recombination operation has considerable influence on the number of necessary generations as well as on the computation time. The speed-up is considerable. In fact, one can see that invoking recombination enables the population to profit from different individuals' progress.

In order to estimate this influence further, we list in Table 1b the results of different runs starting with


a

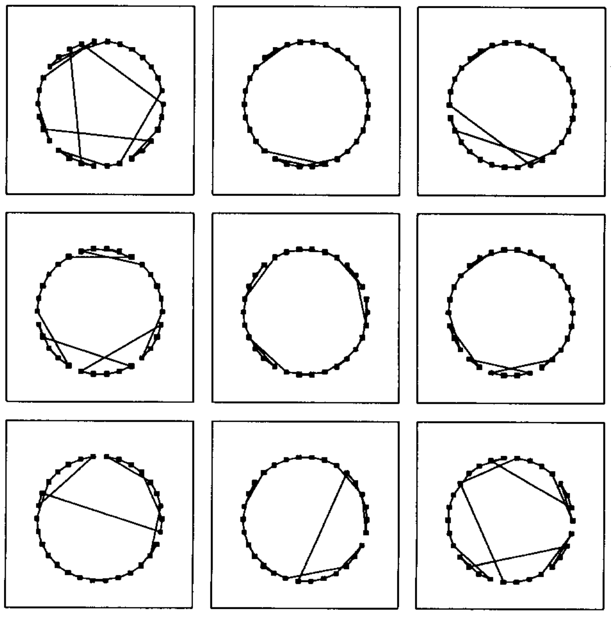

b

Fig. 3a, b. The toy problem of Simulation 1: A ring distribution of $N = 30$ cities. **a** State of a population of $M = 9$ data strings at the starting time, **b** after 1000 generations have passed. Any individual follows its own path towards the optimum due to a low recombination frequency of $t_R = \frac{1}{100}$

Table 1. a Effects of the different machines applied to a ring problem. Sample of $N = 30$, $M = 9$, $t_R = \frac{1}{100}$. Evaluated after reaching an average quality 10% above optimum (time in seconds)

Machines operating	CPU-time	Number of generations
E	20.93	3846
E+C+I	24.27	5447
E+R	7.59	1322
E+C+I+R	19.47	4358

Table 1. b Acceleration through application of recombination operations. Same conditions as in **a**

T_R	CPU-time	Number of generations
1/1000	23.63	5000
1/500	29.74	6110
1/100	19.47	4358
1/50	18.51	3700
1/10	12.52	2556
1/5	8.21	1600
1/1	5.62	738

identical initial conditions. We have varied the recombination frequency ranging from 1 in 1000 generations up to 1 in every generation. Quite evidently, the higher the frequency, the faster the optimization process. The deviation from this rule is a statistical fluctuation due to the influence of initial conditions.

There is, however, one serious drawback of adjusting a higher and higher recombination frequency. This has to do with the disappearance of variance in the population. Since it can be understood best in the context of a real NP problem with many local minima, we now turn to simulation 2 which deals with those problems, consisting of randomly distributed city configurations.

Table 2a gives again, now for randomly distributed cities, the effects of the different operations applied during search. We discover that without the use of the recombination operation, things get not only slower, but become even impossible. For, the use of local search operations leads to traps in bad local minima, the use of all operations except recombination leads at least to local minima. The use, however, of recombination additionally in either case allows to reach the preset quality criterion which still has a distance of nearly 1% from the global optimum.

Table 2b summarizes the effect of rising the recombination frequency. We can observe that a limit for suitable recombination frequencies exists which allows the search process to converge to the preset quality. At the lower end of the recombination frequency table, however, the search failed due to a collapse in variance. Imposing additionally a certain, yet arbitrary time limit could result in another failure at the upper end of the table.

In order to observe the degree of variance in a population of datastrings we have to define the overlap or similarity in the population. This is done here invoking the following definitions: Let the sum of the (upper half) adjacency matrices $Adj^{*}_{i,j}$ (Lawler 1976) of tours represented by datastrings be called the population matrix P:

$$P_{i,j} = \sum_{m=1}^{M} Adj^{*\,m}_{i,j} \tag{3}$$

Then the percentage of overlapping edges in the population O may be defined as

$$O = \frac{\Sigma_{i,j}(P^T)_{ij}(P)_{ij}}{M^2 N} \tag{4}$$

which evidently is a measure of the population variance.

Figure 4 shows an examples of a run with $N = 100$ cities at a state where 90% overlap between the solutions in the population have been reached.

Figure 5 is a more detailed record of one search process. The contributions of different machines to the progress are reported separately. Starting out with a randomly generated population, the progress velocity as shown in Fig. 5 reaches a plateau of small values after nearly 10^5 generations. Before that, it shows a behaviour which is a strong reminescent of exponential progress decay. The similarity between strings increases not very smoothly, interrupted at certain points by a decrease due to fast progress of a few individuals.

As stated before, the different machines contribute differently during the search process: At the very beginning the E-machine is contributing mostly, whereas later on the non-local search operations (C and I-

Table 2. **a** Effects of the different machines applied to a random city problem of size $N = 30$, $M = 9$. Evaluated after a certain average quality was reached of the time limit was exceeded

Machines operating	CPU-time	Number of generations
E[a]	1400[a]	270000[a]
E + C + I[a]	1400[a]	100000[a]
E + R[b]	146.99	27667
E + C + I + R	112.42	24900

[a] Preset quality not reached
[b] $t_R = 1/1000$

Table 2. **b** Acceleration effect and variance collapse is due to different recombination frequencies

t_R	CPU-time	Number of generations
1/1000	489.13	105000
1/500	166.38	34000
1/100	112.42	24900
1/50	42.77	8650
1/10	22.09	4035
1/5	15.73	2767
1/1[a]	49.25[a]	10000[a]

[a] Variance breakdown

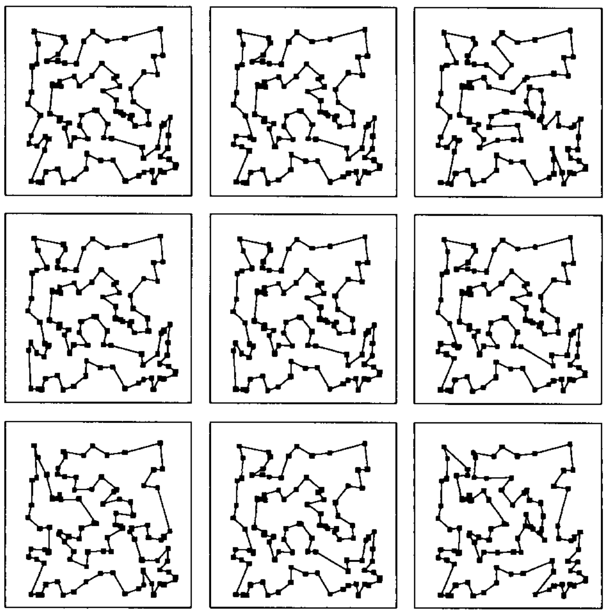


Fig. 4. Final state of population of $M = 9$ strings on a $N = 100$ cities problem in Simulation 2. Evaluation was stopped when 90% overlap has been reached. $t_R = \frac{1}{100}$ and at least 2 different solutions are observable

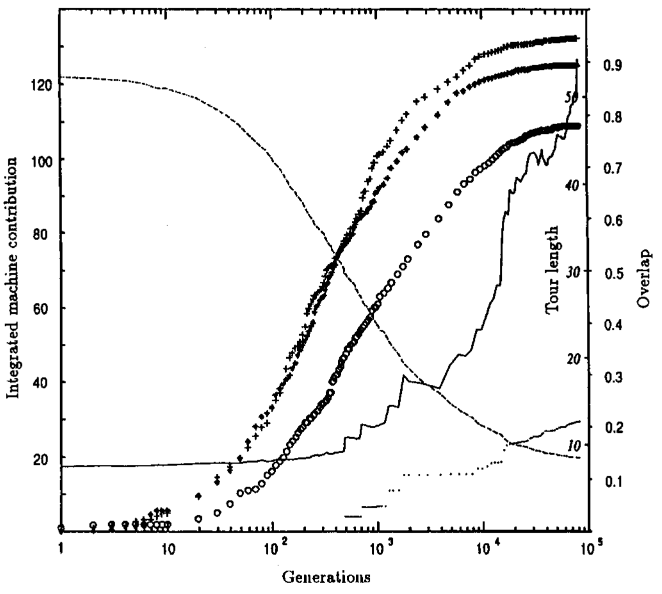


Fig. 5. Sample run with $N = 30$ and $M = 9$, random distribution of cities. Contributions of the different sorts of machines are shown with different markers ($t_R = \frac{1}{100}$).
+: Cut machine; O: Exchange machine;
×: Cut and Inverse machine; .: Recombination machine.
Solid line: Overlap in the population; Dashed line: Average quality of the solutions obtained

machine) take over, followed even later on by the recombination operation. This precisely reflects the state of the population under a given city distribution.

Figure 6 shows the transition between different quality levels in a $N = 30$, $M = 100$ problem. We observe population waves sweeping through the different quality levels on their way downwards. The search process slows down due to a deceleration of the transition between levels.

In simulation 2, the recombination frequency was fixed again at

$$t_R = \frac{1}{100}\,.$$

We claim that this parameter regulating the "degree of isolation" in a population can be adjusted to arbitary small values as to ensure that the true global optimum of the problem is reached.

4 Statistics

The proposed algorithm possesses a strong element of randomness. One may, therefore, raise the objection that most of the results produced so far are the result of very special conditions provided by the random number generators. In order to study the global behavior of this algorithm we have processed hundreds of runs under similar conditions.

Our experiments centered arround the question which influence the population size exerts on the problem solution. We have chosen one random city configuration of $N = 30$ cities, the global optimum of which was known. We then performed 100 runs with different

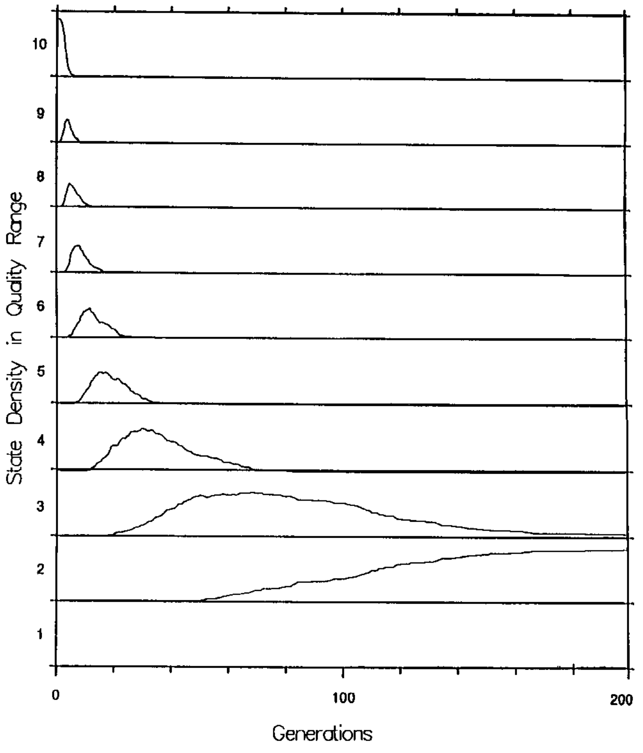


Fig. 6. Development of the population density in certain quality ranges, $M = 100$, otherwise same conditions as in Fig. 5. A quick decay in the worst quality range is followed by slower and slower transitions towards better quality levels (shorter tour lengths). Lowest level not yet reached

initial strings and checked whether and how fast it reached the global solution. The results are given in Table 3a for different sizes of the population. The certainty of finding the global optimum is near 90%, whereas the reason for failure changes from (irreversible) variance collapse for smaller populations to time-limit-excess in larger ones. The average computation time needed to optimize one string approaches a value well below 12 s.

In Table 3b we show (for 50 TSP problems of size $N = 100$) the computation time and the average quality found after the overlap between strings, i.e. the similarity between solutions in the populations, has reached 90% (cf. Fig. 4). Since we had no ab initio information about the true global optimum for this problems, we have chosen the intrinsic criterion of similarity as a measure of quality. Such a criterion proves useful as a general method of ranking the actual stages of the search process (in other optimization problems as well).

How does our result compare to other statistical results? It is well known from the literature (Bonomi and Lutton 1984) that the average TSP has its global optimum at

$$\bar{l} = 0.739\sqrt{N} \text{ for } N \to \infty$$

In the worst case of Table 3b, $M = 9$, we compare this theoretical limit with the best string found by the evolutionary algorithm. The deviation is smaller than 10%. In other words, our stopping criterion allows us to approach the global optimum to at least 10%. One should, however, keep in mind that the statistical result

Table 3. **a** 100 different starting strings for a $N = 30$ random city configuration for which the global optimum was known. Influence of different recombination frequencies on the security of finding the global optimum

Population size	Recombination frequency	Variance collapse	Time limit exceeded	⟨cpu-time⟩ per string
M = 9	1/100	9	2	14.66
M = 18	1/50	1	3	12.36
M = 36	1/25	0	4	12.07
M = 72	1/12	0	10	11.89

Table 3. **b** 50 different $N = 100$ random city configurations. Average of best solutions found at the state when the overlap reached 90%

Population Size	Recombination frequency	⟨cpu-time⟩ per string	Best quality found
M = 9	1/100	181.5	8.093
M = 18	1/50	232.9	7.980
M = 36	1/25	240.8	7.892

Table 4. Scaling of computational time needed until 90% overlap is reached for different dimension of the problem

Problem Size	⟨cpu-time⟩
N = 20	35.12
N = 40	211.34
N = 60	493.94
N = 80	935.77
N = 100	1615.95

of (Bonomi and Lutton 1984) is valid in a limit we are still far away from at $N = 100$!

Table 4 finally shows a report on increasing the dimension of the problem. 100 different runs were performed for every dimensional size to get statistically more reliable statements. We can deduce from this Table that even for serial computers the computational time needed does not explode seriously.

5 Conclusion

We have shown that a parallel optimization process inspired by natural macromolecular evolution is able to approach the optimum in the complicated NP-complete TSP. We have demonstrated the action of a search process which is totally based on local search operations thus minimizing the communication needs in a parallel implementation of the algorithm.

In making use of a population of different data-strings we have made clear that the recombination operation is crucially important in accelerating the evolutionary search process and in overcoming local optima.

We have introduced the recombination frequency parameter in analogy to the phenomenon of isolated evolution in nature as a means to control the variance conserving tendencies of the algorithm. We have varied this parameter in order to study its effects. It turned out that if the recombination frequency was too high, the variance of the population broke down too early as to allow an approach to the global optimum.

Therefore, we expect an optimal recombination frequency to exist at which the search process is maximally fast. This will depend on the problem dimension N, the number of independent search operations p, and the number, M, of individuals participating in the search.

The further examination of this question, as well as a more theoretical description of the processes involved in the proposed search method will be left to a future study.

Acknowledgements. The fundamental ideas behind this work were developed during my post-doctoral time in the Institute of Theoretical Physics and Synergetics at the University of Stuttgart, F.R.G.. I am very grateful to the Director of the institute, Prof. H. Haken, for his encouragement and continuous support. Much of the simulations were done on the SUN SPARC 1 workstation at Nakayama Lab. in the CRL of Mitsubishi Electric Corporation. I want to express my gratitude to Dr. Nakayama as well as Dr. Nara for providing me with all the resources necessary to finish this work.

References

Bonomi E, Lutton J-L (1984) The N-city traveling salesman problem: Statistical mechanics and the metropolis algorithm. SIAM Rev 26:551–568

Boseniuk T, Ebeling W, Engel A (1987) Boltzmann and Darwin strategies in complex optimization. Phys Lett 125A:307–310

Brady RM (1985) Optimization strategies gleaned from natural evolution. Nature 317:804–806

Bremermann HJ, Rogson M, Salaff S (1965) Search by evolution. In: Maxfield M, Callahan A, Fogel LJ (eds) Biophysics and cybernetic systems. Spartan Books, Washington, pp 157–167

Fogel DB (1989) An evolutionary approach to the traveling salesman problem. Biol Cybern 60:139–144

Fontana W, Schuster P (1987) A computer model of evolutionary optimization. Biophys Chem 26:123–147

Grefenstette JJ, Gopal R, Rosmaita B, Van Gucht D (1985) Genetic algorithms for the traveling salesman problem. In: Grefenstette JJ (ed) Proceedings of the International Conference on Genetic Algorithms and their Applications. Carnegie Mellon University, Pittsburgh, pp 160–168

Holland JH (1975) Adaption in natural and artificial systems. University of Michigan Press, Ann Arbor

Kirkpatrick S, Gelatt CD, Vecchi MP (1983) Optimization by simulated annealing. Science 220:671–680

Lawler E (1976) Combinatorial optimization, networks and matroids. Holt, Rinehart and Winston, New York

Lawler E, Lenstra A, Rinnooy Kan G (1985) The traveling salesman problem. Wiley, New York

Metropolis N, Rosenbluth AW, Rosenbluth MN, Teller AH, Teller EJ (1953) Equations of state calculations by fast computing machines. J Chem Phys 21:1087–1091

Mühlenbein H, Gorges-Schleuter M, Krämer O (1988) Evolution algorithms in combinatorial optimazation. Par Comp 7:65–85

Rechenberg I (1973) Evolutionsstrategien. Frommann-Holzboog, Stuttgart

Schwefel HP (1981) Numerical optimization of computer models. Wiley, Chichester

Wang Q (1987) Optimization by simulating molecular evolution. Biol Cybern 57:95–101

Dr. Wolfgang Banzhaf
Central Research Laboratory
Mitsubishi Electric Corporation
1-1, Tsukaguchi Honmachi 8-chome
Amagasaki, Hyogo
661 Japan