A local search template (extended abstract)

EINDHOVEN UNIVERSITY OF TECHNOLOGY
Department of Mathematics and Computing Science

Memorandum COSOR 92-11

A Local Search Template

(Extended Abstract)

R.J.M. Vaessens
E.H.L. Aarts
J.K. Lenstra

Eindhoven, May 1992
The Netherlands

Eindhoven University of Technology
Department of Mathematics and Computing Science
Probability theory, statistics, operations research and systems theory
P.O. Box 513
5600 MB Eindhoven - The Netherlands

Secretariate: Dommelbuilding 0.03
Telephone: 040-47 3130

ISSN 0926 4493

A Local Search Template

(Extended Abstract)

R.J.M. Vaessens[1], E.H.L. Aarts[2,1], J.K. Lenstra[1,3]

1. Department of Mathematics and Computing Science, Eindhoven University of Technology, P.O. Box 513, 5600 MB Eindhoven
2. Philips Research Laboratories, P.O. Box 80000, 5600 JA Eindhoven
3. CWI, P.O. Box 4079, 1009 AB Amsterdam

Abstract

A template is presented that captures the majority of local search algorithms proposed in the literature, such as iterative improvement, simulated annealing, threshold accepting, tabu search, and genetic algorithms. The template leads to a classification of existing local search algorithms and suggests directions for designing new types of local search approaches.

Key words: *local search, iterative improvement, simulated annealing, threshold accepting, tabu search, genetic algorithms.*

1 Introduction

Local search is a generally applicable approach that can be used to find approximate solutions to hard combinatorial optimization problems. The basic idea is to start from an initial solution and to search for successive improvements by examining neighboring solutions. Local search approaches date back to the late 1950's, when Bock [1958] and Croes [1958] developed their link exchange procedures for the traveling salesman problem. Ever since, a large variety of local search algorithms has been proposed, each aiming at different remedies to the risk of getting stuck in poor local optima. Yannakakis [1990] gives a survey of the area.

At present, there is a proliferation of local search algorithms, which, in all their different guises, seem to be based on a few basic ideas only. We present a local search template that has been designed to capture most of the variants proposed in the literature. The aim of our framework is to provide a classification of the various existing local search algorithms. It should also be sufficiently general to capture new approaches to local search and thereby to suggest novel variants.

The organization of this paper is as follows. Section 2 considers the basic iterative improvement algorithm. Section 3 describes our local search template, and Section 4 shows how many algorithms proposed in the literature fit into this template. Section 5 contains some conclusions and suggestions for future research.

2 Deterministic iterative improvement

The basic local search algorithm is the so-called deterministic iterative improvement algorithm. Given an instance of a combinatorial optimization problem and a neighborhood function, the deterministic iterative improvement algorithm starts from an initial solution and then continually searches the neighborhood of the current solution for a solution of better quality. If such a solution is found, it replaces the current solution. The algorithm terminates as soon as the current solution has no neighboring solutions of better quality, at which point a locally optimal solution has been identified.

To discuss deterministic iterative improvement in more detail, we have to define several notions more formally. A *combinatorial optimization problem* is either a maximization or a minimization problem specified by a class of problem instances. An *instance* is defined by the implicit specification of a pair $(\mathcal{S}, f)$, where the *solution space* $\mathcal{S}$ is the set of all feasible solutions, and the *cost function* f is a mapping $f: \mathcal{S} \rightarrow \mathbb{R}$. Without loss of generality we will restrict ourselves to minimization problems. The optimal cost f_{opt} of an instance is defined by $f_{\text{opt}} = \min\{f(i) | i \in \mathcal{S}\}$, and the set of optimal solutions is denoted by $\mathcal{S}_{\text{opt}} = \{i \in \mathcal{S} | f(i) = f_{\text{opt}}\}$. The objective is to find some solution $i_{\text{opt}} \in \mathcal{S}_{\text{opt}}$. In the remainder of the paper, we assume that an instance $(\mathcal{S}, f)$ of such a combinatorial minimization problem is fixed and that $\mathcal{S}$ is finite.

To be able to apply a local search algorithm to this instance, we define a *neighborhood function* $\mathcal{N}$ as a mapping $\mathcal{N}: \mathcal{S} \rightarrow \mathcal{P}(\mathcal{S})$. A solution $i \in \mathcal{S}$ is called a *local minimum with respect to* $\mathcal{N}$ (or in general a *local optimum*) if $f(i) \leq f(j)$ for all $j \in \mathcal{N}(i)$. Furthermore, to distinguish between local optima and elements of $\mathcal{S}_{\text{opt}}$, we call the latter ones *global* optima.

The pseudo-Pascal procedure given in Figure 1 represents the basic part of the deterministic iterative improvement algorithm. Here, the procedure GENERATE NEIGHBOR deterministically generates a solution j from the neighborhood $\mathcal{N}(i)$ of the current solution i, such that

every $j \in \mathcal{N}(i)$ is generated only once as a neighbor of i. At the termination of the procedure DETERMINISTIC ITERATIVE IMPROVEMENT, a solution i is returned that is locally optimal with respect to the neighborhood function $\mathcal{N}$.

```
procedure DETERMINISTIC ITERATIVE IMPROVEMENT (i ∈ S);
  { input:  i ∈ S
    output: i ∈ S }
  begin
    repeat
      GENERATE NEIGHBOR (i, j, N);
      if f(j) < f(i) then i := j
    until ∀j ∈ N(i) : f(j) ≥ f(i)
  end;
```

Figure 1. The procedure DETERMINISTIC ITERATIVE IMPROVEMENT.

The deterministic iterative improvement algorithm terminates as soon as a local optimum is hit upon. To enhance the quality of the solution obtained by local search, one can consider applying the following ideas.

- Generating several or all neighbors of the current solution instead of just one neighbor in each iteration. If all neighbors are generated and a best one is accepted, one obtains a steepest descent algorithm.
- Using more intricate functions to determine a new solution from the current solution and its neighbor, for instance, accepting solutions with quality worse than that of the current solution. The well-known simulated annealing and threshold accepting algorithms fall into this category.
- Replacing the single current solution by a population of current solutions. This is the basic idea of genetic algorithms.
- Alternating between two or more neighborhood functions. Such an algorithm has been proposed by Martin, Otto & Felten [1989].

Based on one or more of the above ideas, a considerable number of algorithms has been proposed in the literature. In the next section we present a generic local search template that captures most of these ideas.

3 A local search template

Our local search template generalizes the iterative improvement algorithm from the previous section in three ways:

(1) the search may proceed at several *levels*, each with its own specifications;

(2) the single current solution will be replaced by a *population* of current solutions;

(3) a neighborhood will not be associated with a single solution but with a *cluster* of solutions.

More formally, each level l has a *population size* p_l and a *cluster size* c_l, with $p_l, c_l \in \mathbb{N}$. A *population* at level l is a p_l-tuple $P \in \mathcal{S}^{p_l}$ of solutions that represents the current state of

the search at level l. We will talk about *point-based* local search if, at the first level, $p_1 = 1$, and about *population-based* local search otherwise. A *cluster* at level l is a c_l-tuple $C \in \mathcal{S}^{c_l}$ of solutions, such that with each cluster a neighborhood is associated. That is, there is a *hyper-neighborhood function* $\mathcal{N}_l : \mathcal{S}^{c_l} \to \mathcal{P}(\mathcal{S})$ which, for each cluster C, defines a set $\mathcal{N}_l(C)$ of neighboring solutions. In case $c_l = 1$, this function reduces to the standard neighborhood function from Section 2.

The local search template consists of two components. It first calls the procedure INITIALIZE, which generates an initial population $P \in \mathcal{S}^{p_1}$; this procedure usually depends on the probem type under consideration. Then, the recursive procedure LOCAL SEARCH is called with level 1 and population P as its parameters. LOCAL SEARCH works as follows.

At level l, the procedure takes a population $P \in \mathcal{S}^{p_l}$ as input and uses the hyper-neighborhood function $\mathcal{N}_l$ to produce a new population $P \in \mathcal{S}^{p_l}$ as output. This is done in two nested loops: an outer loop of *generations* and an inner loop of *iterations.*

The generation loop creates a number of generations of populations until a stopping condition is satisfied. In each generation, the procedure GENERATE CLUSTERS assembles from the current population P a finite multiset $\mathcal{C}$ of clusters $C \in P^{c_l}$. Hence, each of the c_l components $C_1, \ldots, C_{c_l}$ of C is a solution from P.

For each cluster $C \in \mathcal{C}$, the iteration loop applies a number of iterations until a stopping criterion is satisfied. Each iteration starts with a call of the procedure GENERATE NEIGHBORS, which selects a finite multiset $Q \in (\mathcal{N}_l(C))^{p_{l+1}}$. Hence, each of the p_{l+1} components of Q is a hyper-neighbor of C; note that $Q \in \mathcal{S}^{p_{l+1}}$. The procedure LOCAL SEARCH is then called recursively, with level $l + 1$ and population Q as its parameters. The result is a modified population $Q \in \mathcal{S}^{p_{l+1}}$. After this, the procedure REDUCE NEIGHBORS reduces the union of the original cluster C and the new population Q into a new cluster $C \in \mathcal{S}^{c_l}$, which then serves as input for the next iteration.

When, at level l, the iteration loop has terminated for all $C \in \mathcal{C}$, the procedure CREATE collects the solutions found in (usually) the final iteration for each $C \in \mathcal{C}$ into a single set $\hat{P} \subseteq \mathcal{S}$. The procedure REDUCE POPULATION finally merges P and $\hat{P}$ into a new population $P \in \mathcal{S}^{p_l}$.

Figure 2 shows the LOCAL SEARCH TEMPLATE and Figure 3 shows the procedure LOCAL SEARCH, both in pseudo-Pascal. We now give a short description of the procedures and functions used in the procedure LOCAL SEARCH.

- The Boolean function CONTINUE POPULATION GENERATION has the current level l as input. It returns the value TRUE as long as new generations of populations have to be created and FALSE otherwise.

- The procedure GENERATE CLUSTERS has a level l and a population P as input and a finite multiset $\mathcal{C} \subseteq \mathcal{S}^{c_l}$ as output. It clusters P into a collection $\mathcal{C}$ of c_l-tuples $C \in P^{c_l}$, either deterministically or randomly. In this way, the hyper-neighborhood function $\mathcal{N}_l$ can be applied indirectly to the given population P.

- The Boolean function CONTINUE ITERATION has a level l as input; it returns the value TRUE as long as iterations have to go on in the iteration loop and FALSE otherwise.

- The procedure GENERATE NEIGHBORS has a level l, a multiset C of size c_l, and the hyper-neighborhood function $\mathcal{N}_l$ as input, and a multiset Q of size p_{l+1} as output. It generates a multiset Q of neighbors from C using $\mathcal{N}_l$. The basic part of the procedure

```
program LOCAL SEARCH TEMPLATE;
  begin
    INITIALIZE(P);
    LOCAL SEARCH (1, P)
  end;
```

Figure 2. The LOCAL SEARCH TEMPLATE.

```
procedure LOCAL SEARCH (l: integer; P ∈ S^{p_l});
  { input:  l ∈ ℕ, P ∈ S^{p_l}
    output: P ∈ S^{p_l}          }
  begin
    while CONTINUE POPULATION GENERATION (l) do
      begin
        GENERATE CLUSTERS (l, P, C);
        for all C ∈ C do
          begin
            while CONTINUE ITERATION (l) do
              begin
                GENERATE NEIGHBORS (l, C, N_l, Q);
                LOCAL SEARCH (l + 1, Q);
                REDUCE NEIGHBORS (l, C, Q)
              end
          end;
        CREATE (l, P̂);
        REDUCE POPULATION (l, P, P̂)
      end
  end;
```

Figure 3. The procedure LOCAL SEARCH.

prescribes how a neighbor of C is to be determined (randomly, in a specified order, or otherwise) and what the size of Q has to be.

- The procedure REDUCE NEIGHBORS has a level l, a c_l-tuple C, and a p_{l+1}-tuple Q as input, and a modified version of C as output. It determines how to merge the old cluster C and the collection Q of (modified) neighbors into a new cluster C.

- The procedure CREATE has a level l as input and a population $\hat{P}$ as output. It puts a population $\hat{P}$ together from solutions found in (usually) the final iteration for each $C \in \mathcal{C}$.

- The procedure REDUCE POPULATION merges P and $\hat{P}$ into a new population P.

To make the recursive procedure finite, we need to define a bottom level l^*. At this level l^*, the Boolean function CONTINUE POPULATION GENERATION assumes the value FALSE. The levels $l < l^*$ are called *active* levels. Obviously, for the description of a local search algorithm

only a specification of the active levels is needed. We know of no algorithms that use more than two active levels.

The next section shows how most local search algorithms proposed in the literature fit into our template.

4 Instantiations of the local search template

The local search template captures most types of local search algorithms proposed in the literature. This is shown by the specification of the bottom level l^* and, for each active level, by an instantiation of the procedures GENERATE CLUSTERS, REDUCE NEIGHBORS, CREATE and REDUCE POPULATION. The other procedures are usually less characteristic of an algorithm; they are instantiated only if further restrictions are required.

In handling the various local search algorithms we make a distinction between *point-based* and *population-based* local seach and between local search with exactly one and more than one active level.

4.1 Single-level point-based local search

Among point-based local search algorithms with one active level, first the classes of *threshold* and *tabu search* algorithms are discussed. Both are characterized by the fact that only one generation is created. Hence, they are determined by the iteration loop of the procedure LOCAL SEARCH. Next, *variable-depth search* is discussed.

4.1.1 Threshold algorithms and tabu search

The following instantiations hold for both threshold and tabu search algorithms:

- CONTINUE POPULATION GENERATION returns the value TRUE for the first generation and FALSE for the subsequent generation. In this way only one generation is created.
- GENERATE CLUSTERS sets $\mathcal{C} = (C)$ equal to (P). Both C and P contain one solution only.
- CREATE sets $\hat{P}$ equal to the current C.
- REDUCE POPULATION sets the new population P equal to $\hat{P}$.

We now consider threshold and tabu search algorithms separately.

Threshold algorithms constitute a class of algorithms that contains iterative improvement, simulated annealing [Kirkpatrick, Gelatt & Vecchi, 1983] and threshold accepting [Dück & Scheuer, 1990]. They are characterized by the following instantiations:

- GENERATE NEIGHBORS generates only one neighbor in Q using the neighborhood function $\mathcal{N}_1$. In most cases a neighbor is generated randomly; sometimes this is done deterministically.
- REDUCE NEIGHBORS determines whether the solution Q_1 (that is, the unique component of Q) satisfies $f(Q_1) - f(C_1) < t$ for a certain threshold value t, where C_1 denotes the

first (and only) component of C. If this is the case, Q_1 replaces the current solution C_1; otherwise, C_1 remains unchanged. Depending on the nature of the thresholds, one distinguishes between several types of threshold algorithms. Iterative improvement and threshold accepting both use deterministic thresholds, in contrast to simulated annealing, which uses probabilistic thresholds. The class of iterative improvement algorithms contains the deterministic iterative improvement algorithm introduced in Section 2 as a special case.

Tabu search [Glover, 1989] combines the deterministic iterative improvement algorithm with a possibility to accept cost increasing solutions. In this way the search is directed away from local minima, such that other parts of the solution space can be inspected. This is done by maintaining a finite list of solutions that are not acceptable in the next few iterations. This list is called the *tabu list.* However, a solution on the tabu list may be accepted if its quality is in some sense good enough, in which case it is said to attain a certain *aspiration level.* Tabu search algorithms are characterized by the following instantiations:

- GENERATE NEIGHBORS selects deterministically all neighbors of the current solution C_1 with respect to $\mathcal{N}_1$ by inspecting these in a prespecified order.
- REDUCE NEIGHBORS determines among all solutions in Q that are not on the tabu list, and all solutions in Q that are on the tabu list but attain a certain aspiration level, a solution $Q_j \neq C_1$ of minimum cost. C_1 is then replaced by Q_j.
- CONTINUE ITERATION returns the value TRUE as long as the best solution found so far is not yet of a prescribed quality, or as long as a prescribed number of iterations has not yet been reached. Furthermore, when the tabu list contains all neighbors of C_1 and none of these attains the aspiration level, the function CONTINUE ITERATION returns the value FALSE.

It is also possible that the procedure GENERATE NEIGHBORS selects only one neighbor per iteration and that the function REDUCE NEIGHBORS accepts this neighbor when it is not on the tabu list or attains the aspiration level, and rejects it otherwise. But in that case the tabu list has to be significantly larger, so as to avoid that the procedure accepts a solution with a cost larger than the current solution too often, which would require unacceptably large amounts of memory space and computation time.

4.1.2 Variable-depth search algorithms

In contrast to the above algorithms, variable-depth search algorithms creates several generations. In each generation a finite sequence of neighboring solutions is generated, such that the computation of each next solution in the sequence takes one iteration. Each solution in this sequence, except the first one, is in principle a minimum cost neighbor of the previous one. However, in this approach the risk of cycling is large. To avoid cycling, a sort of tabu list is introduced, which prevents the search from generating a solution that has occurred in the sequence before. At the beginning of each iteration loop the tabu list is emptied. The instantiations for variable-depth search are as follows:

- GENERATE CLUSTERS sets $\mathcal{C} = (C)$ equal to (P). Both C and P contain one solution only.

- REDUCE NEIGHBORS determines among all solutions in Q not on the tabu list a solution $Q_j \neq C_1$ of minimum cost. C_1 is then replaced by Q_j.

- CREATE sets $\hat{P}$ equal to the current $C = (C_1)$, where C_1 is a solution found in the last iteration loop that is different from the solution with which the iteration loop started. There are two possibilities for choosing C_1:

  1. Choose a solution with smallest cost among those obtained in the last iteration loop.
  2. Choose the first solution obtained in the last iteration loop that has smaller cost than the solution with which the iteration loop started, provided that such a solution has been found; otherwise, choose an arbitrary solution obtained in the last iteration loop. Note that, as soon as a solution is found that has smaller cost than the solution with which the iteration loop started, the loop can be stopped immediately by letting CONTINUE ITERATION return the value FALSE.

- REDUCE POPULATION simply selects the best of the two solutions in P and $\hat{P}$. Ties are broken arbitrarily.

- GENERATE NEIGHBORS selects all neighbors of the current solution deterministically by inspecting these in a prespecified order.

- CONTINUE POPULATION GENERATION returns the value TRUE as long as the sequence of the costs of solutions in P for the subsequent generations is strictly decreasing.

- CONTINUE ITERATION returns the value TRUE as long as the number of iterations has not yet reached a specified upper bound.

4.2 Multi-level point-based local search

We now discuss point-based local search algorithms with more than one active level. Very few algorithms of this type have been proposed in the literature. Furthermore, the existing ones are often tailored to a specific problem type. Algorithms of this type can usually be composed from single-level point-based local search algorithms. Hence, it is not necessary to specify the corresponding procedures and functions in detail here.

Nevertheless, since algorithms of this kind seem to give good results, an example due to Martin, Otto & Felten [1989] is given. Their algorithm for the traveling salesman problem uses, in our terminology, two active levels.

At level 1 they use simulated annealing. Their neighborhood is a subset of the 4-exchange neighborhood. After selecting a single neighbor at level 1, at level 2 they determine a local minimum with respect to a special 3-exchange neighborhood, using any single-level point-based local search algorithm that is able to do so. Then this local minimum is compared with the current solution at level 1 and is accepted using simulated annealing. The authors attribute the power of their algorithm to the fact that, after making a single 4-exchange and then applying 3-exchanges until a local optimum is reached, typically many links in the tour have been changed. Algorithms of this type, which use more levels in the local search template, seem to be powerful and deserve wider attention.

4.3 Single-level population-based local search

We now discuss a class of single-level population-based local search algorithms, called *genetic* algorithms. These were first introduced by Holland [1975] and have been well described in a textbook by Goldberg [1989].

In each generation, first some clusters C of the current population P are created. To each C, the hyper-neighborhood function $\mathcal{N}_1$ is applied to produce a set of new solutions. From these new solutions and the solutions of the current population, the low cost solutions are selected to form a new population, which then starts up a next generation. The generation loop terminates as soon as some stopping criterion, which usually is chosen heuristically, is satisfied. The instantiations for the class of genetic algorithms are as follows:

- GENERATE CLUSTERS generates from the population P of size p_1 a multiset $\mathcal{C}$ of clusters C of size c_1. In most cases the clusters are formed heuristically and in such a way that solutions with lower cost are contained in a cluster with higher probability. Note that a solution in P can occur in more than one cluster and even several times in the same cluster.
- REDUCE NEIGHBORS takes from the current cluster C and from Q the c_1 best solutions to form a new cluster C.
- CREATE sets $\hat{P}$ equal to the union of all current clusters $C \in \mathcal{C}$.
- REDUCE POPULATION merges P and $\hat{P}$ into a new population P. In most variants, this is done by choosing from P and $\hat{P}$ exactly p_1 elements, with a preference for low-cost solutions.

The remaining procedures can be chosen as follows.

- GENERATE NEIGHBORS selects randomly a number of neighbors of the current cluster C using the hyper-neighborhood function $\mathcal{N}_1$. In many implementations, this number of neighbors also equals c_1.
- CONTINUE ITERATION usually returns the value TRUE for the first iteration and FALSE otherwise. In this way, only one set of neighbors is generated for each chosen cluster, after which the iteration loop is left. In this case, the function REDUCE NEIGHBORS can be skipped, since there is no reason to create a new current cluster C when there is one iteration only.
- CONTINUE POPULATION GENERATION gives the value TRUE for instance as long as a certain upper bound on the number of generations has not yet been reached, or as long as the population contains different solutions.

4.4 Multi-level population-based local search

Few examples of population-based local search algorithms with more than one active level are known. Here we discuss the so-called *genetic local search* approach [Ulder, Aarts, Bandelt, Van Laarhoven & Pesch, 1990], which is a variant of the class of genetic algorithms. The only difference is that there is now a second active level, in which a point-based hyper-neighborhood function $\mathcal{N}_2 : \mathcal{S} \rightarrow \mathcal{P}(\mathcal{S})$ is used.

After the computation of a tuple Q of neighbors at the first level, at the second level a local minimum is computed for each solution Q_j, using the neighborhood function $\mathcal{N}_2$. After that, back at level 1 the function REDUCE NEIGHBORS is applied to the current Q, which now contains local minima with respect to the neighborhood function $\mathcal{N}_2$. The instantiations for the second level are as follows:

- CONTINUE POPULATION GENERATION gives the value TRUE for the first generation and FALSE for the subsequent generations. In this way, only one generation is created.
- GENERATE CLUSTERS creates for each solution $P_i \in P$ a cluster $C = (P_i)$.
- CREATE sets $\hat{P}$ equal to the set that, for each cluster, contains a local minimum obtained in the iteration loop for the corresponding cluster.
- REDUCE POPULATION sets the new population P equal to $\hat{P}$.

CONTINUE ITERATION, REDUCE NEIGHBORS and GENERATE NEIGHBORS are the same as the ones specified for the deterministic iterative improvement algorithm.

5 Conclusion

We have proposed a unifying framework for the many different types of local search algorithms in combinatorial optimization. The main component of our template is a recursive procedure LOCAL SEARCH. On the basis of the depth of the recursion (one or more levels) and the size of the population (one or more solutions), local search algorithms have been classified and several variants proposed in the literature have been shown to fit into the classification. We hope that our template and the corresponding classification may stimulate the development of new types of local search algorithms. In this respect, multi-level local search seems to deserve special attention, as existing algorithms of that type have led to impressive computational results.

References

BOCK, F. [1958], An algorithm for solving 'traveling salesman' and related network optimization problems, Talk given at the 14th ORSA Meeting, St. Louis, October 24, 1958.

CROES, G.A. [1958], A method for solving traveling salesman problems, *Operations Research 6*, 791-812.

DÜCK, G., T. SCHEUER [1990], Threshold accepting; a general purpose optimization algorithm, *Journal of Computational Physics 90*, 161-175.

GLOVER, F. [1989], Tabu search - Part I, *ORSA Journal on Computing 1*, 190-206.

GOLDBERG, D.E. [1989], *Genetic Algorithms in Search, Optimization, and Machine Learning*, Addison-Wesley, Reading, Massachusetts.

HOLLAND, J.H. [1975], *Adaptation in Natural and Artificial Systems*, University of Michigan Press, Ann Arbor.

KIRKPATRICK, S., C.D. GELATT JR., M.P. VECCHI [1983], Optimization by simulated annealing, *Science 220*, 671-680.

MARTIN, O., S.W. OTTO, E.W. FELTEN [1989], Large-step Markov chains for the traveling salesman problem, *Manuscript*.

ULDER, N.L.J., E.H.L. AARTS, H.-J. BANDELT, P.J.M. VAN LAARHOVEN, E. PESCH [1990], Improving TSP exchange heuristics by population genetics, *Proceedings International Workshop on Parallel Problem Solving from Nature*, Dortmund, Germany, October 1-3, 1990, 109-116.

YANNAKAKIS, M. [1990], The analysis of local search problems and their heuristics, *Proceedings 7th Annual Symposium on Theoretical Aspects of Computer Science (STACS 90)*, Lecture Notes in Computer Science 415, Springer, Berlin, 298-311.

List of COSOR-memoranda - 1992

Number	Month	Author	Title
92-01	January	F.W. Steutel	On the addition of log-convex functions and sequences
92-02	January	P. v.d. Laan	Selection constants for Uniform populations
92-03	February	E.E.M. v. Berkum H.N. Linssen D.A. Overdijk	Data reduction in statistical inference
92-04	February	H.J.C. Huijberts H. Nijmeijer	Strong dynamic input-output decoupling: from linearity to nonlinearity
92-05	March	S.J.L. v. Eijndhoven J.M. Soethoudt	Introduction to a behavioral approach of continuous-time systems
92-06	April	P.J. Zwietering E.H.L. Aarts J. Wessels	The minimal number of layers of a perceptron that sorts
92-07	April	F.P.A. Coolen	Maximum Imprecision Related to Intervals of Measures and Bayesian Inference with Conjugate Imprecise Prior Densities
92-08	May	I.J.B.F. Adan J. Wessels W.H.M. Zijm	A Note on "The effect of varying routing probability in two parallel queues with dynamic routing under a threshold-type scheduling"
92-09	May	I.J.B.F. Adan G.J.J.A.N. v. Houtum J. v.d. Wal	Upper and lower bounds for the waiting time in the symmetric shortest queue system
92-10	May	P. v.d. Laan	Subset Selection: Robustness and Imprecise Selection
92-11	May	R.J.M. Vaessens E.H.L. Aarts J.K. Lenstra	A Local Search Template (Extended Abstract)