Parallel Local Search

M.G.A. Verhoeven [1] E.H.L. Aarts [2,1]

[1]Eindhoven University of Technology, Department of Mathematics and Computing Science, P.O. Box 513, 5600 MB Eindhoven, Netherlands, marcov@win.tue.nl

[2]Philips Research Laboratories, P.O. Box 80000, 5600 JA Eindhoven, Netherlands, aarts@prl.philips.nl

Abstract

We present a survey of parallel local search algorithms in which we review the concepts that can be used to incorporate parallelism into local search. For this purpose we distinguish between single-walk and multiple-walk parallel local search, and between asynchronous and synchronous parallelism. Within the class of single-walk algorithms we differentiate between multiple-step and single-step parallelism. To describe parallel local search we introduce the concepts of hyper neighborhood structures and distributed neighborhood structures. Furthermore, we present templates that capture most of the parallel local search algorithms proposed in the literature. Finally, we discuss some complexity issues related to parallel local search.

Keywords. Local search, parallel algorithms, survey, single-walk and multiple-walk parallelism.

1 Introduction

In a combinatorial optimization problem one is asked to find an optimal solution among a finite, possibly large, number of alternative solutions [Papadimitriou & Steiglitz, 1982]. An important achievement in combinatorial optimization is the existence of two classes of problems known as $\mathcal{P}$ and $\mathcal{NP}$, and the fact that, unless $\mathcal{P} = \mathcal{NP}$, there exists a class of problems for which no algorithm can exist that solves each instance of the problem to optimality within a time that is polynomially bounded in the size of the instance. These problems are called *NP-hard* [Garey & Johnson, 1979]. As a consequence of this, the larger instances of a problem can only be solved to proximity in acceptable running times.

Local search algorithms constitute a class of approximation algorithms that are based on the exploration of neighborhoods of solutions. They have shown to be successfully applicable to a wide range of problems, giving good quality solutions [Aarts & Lenstra, 1995]. However, local search algorithms often require substantial amounts of running time for large instances.

One way to reduce the running time of local search algorithms is by using parallelism. In the literature various approaches have been suggested for designing parallel local search algorithms. We have observed that, disregarding implementation details, most algorithms that have been presented are based on a few ideas only. In this article we present a survey of the concepts used to design parallel local search algorithms. Furthermore, we present two templates of parallel local search algorithms that abstract from implementation details and capture the basic ideas underlying most parallel local search algorithms.

2 Local search

A combinatorial optimization problem is either a minimization problem or a maximization problem consisting of a set of problem instances. Without loss of generality, we only consider minimization problems, since a maximization problem can be transformed into a minimization problem by reversing the sign of the cost function.

Definition 2.1. An *instance* of a combinatorial optimization problem is given by a compact representation of a pair $(\mathcal{S}, f)$, where $\mathcal{S}$ is the set of solutions, and $f : \mathcal{S} \to \mathbb{R}$ is a function that gives the cost of a solution. The objective is to find a solution with minimal cost. □

Local search algorithms constitute a class of approximation algorithms that are based on repeatedly replacing a solution by a neighboring solution. The basic local search algorithm is the so-called *iterative improvement* algorithm. An iterative improvement algorithm starts off with an initial solution constructed by some heuristic. Next, the algorithm repeatedly tries to improve the current solution by replacing it with a neighbor with lower cost. If a solution has been reached that has no neighbors with lower cost, a local minimum has been found. An essential concept in local search algorithms is the notion of a neighborhood structure.

Definition 2.2. Let $(\mathcal{S}, f)$ be an instance of a combinatorial optimization problem. Then a *neighborhood structure* $\mathcal{N} : \mathcal{S} \to \mathcal{P}(\mathcal{S})$ assigns to each solution in $\mathcal{S}$ a set of solutions, called a *neighborhood*. A solution $s \in \mathcal{S}$ is a *local minimum* of $\mathcal{N}$, if $f(s') \geq f(s)$ for each $s' \in \mathcal{N}(s)$. □

proc *Iterative_Improvement* $(s : \mathcal{S})$
var $w : \mathcal{P}(\mathcal{S})$;
begin
$w := \emptyset$;
while $\mathcal{N}(s) \setminus w \neq \emptyset$ **do**
$s' :\in \mathcal{N}(s) \setminus w$;
if $f(s') \geq f(s)$ **then** $w := w \cup \{s'\}$
else $s := s'$; $w := \emptyset$ **fi**
od $\{s$ is a local minimum w.r.t. $\mathcal{N}\}$
end

Figure 1: An iterative improvement algorithm.

The neighbors of a solution are not given explicitly, but can be constructed by a construction function. Most neighborhoods are based on the replacement of a few elements that constitute a solution. For this we need the following definition in which we assume that solutions are represented by sets.

Definition 2.3. Let the set of elements constituting solutions in $\mathcal{S}$ be given by $\mathcal{E} = \{e \mid e \in s \wedge s \in \mathcal{S}\}$, and let the *exchange function* $\tau : \mathcal{S} \times \mathcal{E}^l \to \mathcal{S}$ be a partial function with $l \in \mathbb{N}$. Then, an l-exchange neighborhood structure $\mathcal{N}_l$ is defined by $\mathcal{N}_l(s) = \{\tau(s, e_1, \ldots, e_l) \mid e_1, \ldots, e_l \in s\}$ for $s \in \mathcal{S}$. □

According to Definition 2.3 neighbors can be constructed from a solution s by applying the exchange function to s and a subset $e_1, \ldots, e_l$ of the elements of s. Below, we give an example of an exchange function and a neighborhood.

Example 2.1 (Traveling Salesman Problem). Let V be a set of N cities and $d_{ij} \in \mathbb{R}$ the distance between each $i, j \in V$. A tour t is a set of N edges $\{e_1, \ldots, e_N\}$ that constitutes a Hamilton cycle in the complete graph $(V, V \times V)$. The problem is to find a tour with minimal length. The exchange function 2-*exchange*$(t, r, s) : \mathcal{S} \times \mathbb{N} \times \mathbb{N} \to \mathcal{S}$ gives for each tour a new tour obtained by replacing two edges e_r, e_s with two other edges. The well-known 2-exchange neighborhood of Lin [1965] is now defined by $\mathcal{N}_2(t) = \{2\text{-}exchange(t, r, s) \mid 1 \leq r < s \leq N\}$. □

Figure 1 presents a schematic description of an iterative improvement algorithm. A single iteration of an iterative improvement algorithm, which we call a *step*, consists of the following actions. First, a neighbor is constructed by applying the exchange function to the current solution. Subsequently, the cost of the neighbor is evaluated. If the neighbor has lower cost, it becomes the new current solution, i.e., the proposed exchange is effectuated. Otherwise the search is continued with the old current solution. A pivoting rule [Yannakakis, 1990] determines which neighbor will be the new current solution. Well-known pivoting rules are *first improvement*, which replaces the current solution with the first lower cost neighbor that is found, and *best improvement* which uses the solution that has lowest cost among all neighbors.

The algorithm of Figure 1 terminates since in each step the cost of the current solution is decreased. However, examples are known of problem instances for which the number of steps needed to find a local optimum cannot be bounded by a polynomial in the size of the instance [Lueker, 1975]. Johnson, Papadimitriou & Yannakakis [1988] have formalized the question of the worst-case behavior of local search algorithms by introducing a new complexity class $\mathcal{PLS}$ consisting of local search problems for which local optimality can be verified in polynomial time. Also, a class of PLS-complete problems has been introduced that contains the hardest problems in $\mathcal{PLS}$, and if one of these problems can be solved in polynomial time, then all others can. It is, however, conjectured that $\mathcal{P}_s \subset \mathcal{PLS}$, where $\mathcal{P}_s$ is the class of search problems which can be solved in polynomial time [Yannakakis, 1995]. Consequently, if this conjecture holds, super polynomial running times might be required by any algorithm to find a local optimum for the problems in $\mathcal{PLS}$.

A neighborhood structure imposes a graph on the solution space. The vertices of this *solution space graph* are the solutions, and two vertices are connected by an edge if they are neighbors. The course of a local search algorithm is characterized by a walk in this graph. Starting in some vertex, the costs of neighbors are evaluated, and a step to some neighbor is made. This process is repeated until some stop criterion is met. The effectiveness of local search depends on both the structure of the solution space graph, which is determined by the neighborhood structure, and the way the solution space graph is traversed, which is determined by the type of local search that is used. To improve the effectiveness of local search one can either modify the neighborhood structure, or change the way the solution space graph is traversed. The first approach is problem dependent, whereas the latter one is problem independent.

Most neighborhood structures are based on exchange functions that modify only a few elements resulting in small neighborhoods. In order to avoid the risk of getting stuck in a poor-quality local optimum, one may try to increase the size of the neighborhood by exchanging more elements. Another option is to change the nature of the neighborhood, i.e., exchanging elements in a different way. Examples of this approach are *variable-depth neighborhood structures* [Kernighan & Lin, 1970; Lin & Kernighan, 1973], also called *ejection chains* [Reeves, 1993]. Variable-depth neighborhoods are based on a simple exchange function. A neighbor is obtained by constructing a sequence of exchanges, instead of a single exchange. An algorithm that uses a variable-depth neighborhood is often no longer regarded as a basic iterative improvement algorithm since it actually generates a sequence of solutions, which may contain solutions with higher cost than a preceding solution. The solution that eventually replaces the current solution is the solution in this sequence with minimal cost. The algorithm stops if this neighbor has higher cost than the current solution.

Another approach to escape from local minima is to accept neighbors with cost higher than the current solution. We mention the following local search variants which all have different mechanisms to escape from local minima.

Iterated local search [Johnson, 1990; Martin, Otto & Felten, 1991] uses two neighborhoods. The search starts with the first neighborhood until a local minimum is found. Then a neighbor is chosen from a second neighborhood. Subsequently, this neighbor is used as starting solution for the next run of a local search algorithm with the first neighborhood. In this way local search is performed at two levels.

Tabu search [Glover, 1989; Glover, Taillard & De Werra, 1993] tries to direct the search into unexplored areas of the solution space once a local minimum is found, by memorizing the course of the search. To this end, a finite list of exchanges that have lead the search to a local minimum is constructed. The algorithm escapes from a local minimum by performing an exchange not in this list. *Intensification* of the search around good local minima is used to search promising areas of the solution space more thoroughly. Furthermore, *diversification* is used to ensure that other regions of the solution space are not neglected.

Simulated annealing [Kirkpatrick, Gelatt & Vecchi, 1983; Aarts & Korst, 1989] probabilistically accepts neighbors with higher costs, in addition to cost improving neighbors which are always accepted. To this end, an *acceptance probability* is used that is reversely related to the size of the cost increase. In the course of the search the acceptance probability is lowered. It has been proven that simulated annealing converges to a global minimum under mild conditions if the acceptance probability is lowered slowly enough.

Genetic local search [Goldberg, 1989; Mühlenbein, Gorges-Schleuter & Krämer, 1988] incorporates local search into genetic algorithms. In genetic local search, a *population* of solutions is explored, instead of only a single solution. Using two *parent* solutions from this population, a new solution is constructed by using *cross-over* and *mutation*. This solution is then improved by a local search algorithm and added to the population. A new population is selected from the old population using concepts from biological evolution strategies.

A widely recognized advantage of local search is its general applicability. Only a specification of solutions, a cost function, and a neighborhood structure are required, which can easily be defined for many problems. Furthermore, practice has revealed that the average-case performance of local search is quite good in the sense that high quality solutions can be found within acceptable running times. For overviews on local search in which the above variants are discussed in more detail we refer to [Yannakakis, 1990; Reeves, 1993; Aarts & Lenstra, 1995].

3 Concepts of parallel local search algorithms

There exist problems for which local search algorithms require substantial amounts of running time for the larger instances; see for instance [Sechen, 1988], who reports running times up to 24 hours for his simulated annealing based cell-placement algorithm when applied to real-life problem instances. To cope with this drawback several approaches have been proposed to implement these algorithms on parallel machines. Much effort in this field has been concentrated on the design of parallel simulated annealing algorithms [Aarts & Korst, 1989; Azencott, 1992]. For an overview of parallel simulated annealing we refer to [Greening, 1990]. More recently, also parallel approaches to other local search algorithms, such as tabu search and genetic local search, have been reported. Voss [1993] presents a classification of parallel tabu search algorithms based on the use of different search strategies and starting solutions. Crainic, Toulouse & Gendreau [1993b] extended this classification with dimensions based on communication organization and information handling.

The objective of our paper is to present an overview of a more general class of parallel local search algorithms. In our discussion we concentrate on the underlying concepts used in the

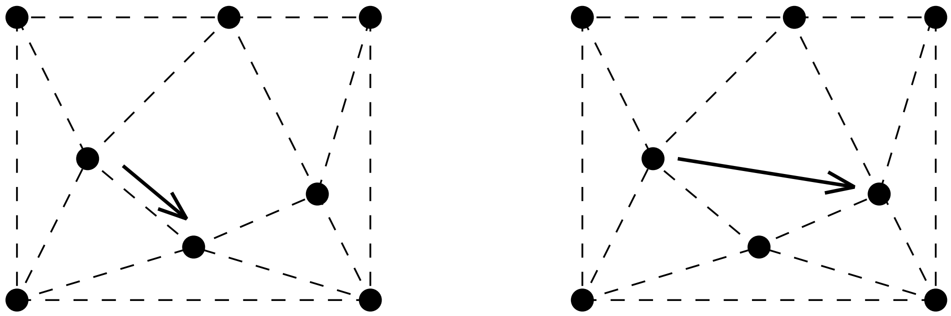

Figure 2: Local search with single-step parallelism (left) and multiple-step parallelism (right)

various approaches to introduce parallelism in local search, and we try to abstract from implementation issues that depend on the machine model at hand.

First, we distinguish between *tailored* and *general* approaches. A tailored approach can only be applied to a specific problem, whereas a general approach can be applied to a wide variety of problems. Furthermore, we distinguish between single-walk and multiple-walk parallelism and between asynchronous and synchronous algorithms. In the class of single-walk algorithms, we distinguish between multiple-step and single-step parallelism.

Tailored approaches. A tailored approach to employ parallelism in local search requires parallel execution of a problem specific part of the algorithm. Cost computation is a typical problem dependent part of a local search algorithm. Also operations on the data structures used to represent a solution are problem specific. If one of these problem dependent parts is very time consuming, the running time of the algorithm can be decreased substantially by efficient parallel execution of this part.

Kravitz & Rutenbar [1987] apply a tailored approach to their local search algorithm for the standard cell placement problem. They compute the cost of a neighbor in parallel. Another example of parallel cost computation is given by Taillard [1994], who uses a parallel longest path algorithm to compute the length of a schedule in the job shop scheduling problem. Yeong & Kim [1991] use a parallel algorithm for the replacement of a solution by a neighbor in the traveling salesman problem.

The applicability of a tailored approach is limited, because it depends strongly on the problem at hand. Therefore, we will discard these approaches here and concentrate on techniques that can be applied to a wide variety of local search problems.

General approaches. A general approach can be applied to a broad class of local search problems. We can distinguish between two approaches: *single-walk* and *multiple-walk* parallel local search. In a single-walk algorithm only a single walk in the solution space graph is carried out, whereas in a multiple-walk algorithm several walks are performed simultaneously.

Within the class of multiple-walk algorithms we can distinguish between algorithms performing interacting walks and algorithms which perform multiple independent walks. The latter can be considered as the most straightforward approach to introduce parallelism in local search.

In single-walk parallel local search a single walk is carried out in parallel. Typically, this requires some distribution of a solution or elements of a solution over a number of processors. Furthermore, we distinguish between *single-step* and *multiple-step* parallelism.

The idea of single-step parallelism is to evaluate neighbors simultaneously and subsequently make a single step. To this end the neighborhood of a solution is partitioned into subsets that are searched in parallel. In an algorithm with multiple-step parallelism several consecutive steps through the solutions space graph are made simultaneously. Figure 2 shows solution space graphs that illustrate the idea of single-step and multiple-step parallelism. Here dotted edges connect neighbors and a solid arc represents a step made by a local search algorithm.

Finally, both in single-walk and multiple-walk parallel local search we distinguish between *synchronous* and *asynchronous* algorithms. In a synchronously parallel algorithm one or more steps of the algorithm are performed simultaneously by all processors. Synchronous parallelism requires a global clocking scheme which guarantees that communication occurs at fixed points in time. In asynchronous parallelism no such central clocking scheme exists.

A multiple-walk algorithm performs multiple single walks. Hence, it is possible to combine multiple-walk and single-walk parallelism in the design of parallel local search algorithms. Also tailored approaches can be incorporated into general approaches resulting in all kinds of hybrid approaches, but the items in our classification are the basic concepts of such algorithms.

The remainder of this paper is organized as follows. In Section 4 we discuss multiple-walk parallelism. Section 5.1 discusses single-step parallelism, and Section 5.2 discusses multiple-step parallelism in more detail. Finally, Section 6 discusses some complexity issues of parallel local search.

4 Multiple-walk parallelism

Multiple-walk algorithms have inherent parallelism, which can be exploited by distributing the walks over a number of processors, and performing them simultaneously.

We distinguish between algorithms that perform multiple independent walks and algorithms that perform multiple interacting walks. The first class of algorithms is the simplest one, since no complex parallelization scheme is required. In order to describe algorithms with multiple interacting walks more precisely, we extend the notion of neighborhood structures to hyper neighborhood structures, in a similar way as was done by Vaessens, Aarts & Lenstra [1992] for sequential algorithms. Neighborhood structures define neighbors for a single solution; hyper neighborhood structures define neighbors for a set of solutions.

Definition 4.1. Let $M \geq 1$. Then, a hyper neighborhood structure $\mathcal{H} : \mathcal{S}^M \to \mathcal{P}(\mathcal{S})$ assigns to a set of M solutions in $\mathcal{S}$, a subset of solutions in $\mathcal{S}$. $\square$

Let $\Omega = \{1, \dots, P\}$ denote a set of P processors, and let $h_p : \{1, \dots, M\} \to \Omega$ be a function for each $p \in \Omega$, where $h_p(i)$ gives the i-th processor with which p communicates. h_p gives for each processor p, M other processors with which interaction takes place in order to construct a hyper neighbor.

Using the concept of hyper neighborhood structures, we can formulate the template presented in Figure 3 which describes algorithms with interacting multiple-walk parallelism. The set of current solutions $s_1, \dots, s_P$ is often called the *population*. Each current solution s_p is assigned to processor p. All processors perform simultaneously a walk in the solution space graph by executing an iterative improvement algorithm, or some other local search algorithm. After a number

```
proc Multiple_Walks (s_1, ..., s_P ∈ S)
  begin
    stop_criterion := false;
    while ¬ stop_criterion do
      par p ∈ Ω do
        Iterative_Improvement(s_p)
        s_p :∈ H(s_{h_p(1)}, ..., s_{h_p(M)});
      rap
    od
  end
```

Figure 3: A multiple-walk parallel local search algorithm.

of local search steps, a hyper neighbor is constructed, which requires interaction with M other processors $h_p(1), \ldots, h_p(M)$. This process is continued until some stop criterion is met.

4.1 Multiple independent walks

The most straightforward approach to parallel local search is to simply perform a number of independent walks in parallel, since this requires no communication between different runs. Therefore, we first discuss a simple but important result for the speed-up that can be obtained with multiple independent walks.

Theorem 4.1. *Let $Q_P(t)$ be the probability of not having found a solution with a given quality in t time units with P independent walks. If $Q_1(t) = e^{-\lambda t}$ with $\lambda \in \mathbb{R}^+$, i.e., Q_1 corresponds to an exponential distribution, then $Q_P(t) = Q_1(Pt)$.* □

Hence, it is possible to achieve linear speed-up by multiple independent walks, if the probability to find an optimal (or sub-optimal) solution within a given amount of time units is distributed exponentially. An appropriate time measure for local search algorithms is the number of solutions of which the cost has been evaluated. Theorem 4.1 states that the probability to find a (sub)optimal solution in t time units and P processors is equal to the probability to find a (sub)optimal solution in Pt time with a single processor, when this probability is distributed exponentially. So, in that case a linear speed-up is reached with multiple independent walks.

Several authors have studied the property of Theorem 4.1 by investigating the probabilistic behavior of local search algorithms for various problems. Battiti & Tecchiolli [1992] empirically investigate the behavior of tabu search for randomly generated instances of the quadratic assignment problem. They observe that the probability of finding a (sub)optimal solution with a tabu search algorithm is indeed distributed exponentially, provided the search is started with a local minimum. This indicates that good efficiencies can be obtained with multiple independent parallel walks in tabu search, if the time needed to find the first local minimum is relatively small compared to the time spent in the remainder of the search process. Taillard [1991] also shows that the probability of finding an optimal solution for quadratic assignment problems with a tabu search algorithm fits well with an exponential distribution. Dodd [1990] empirically shows that a similar conclusion also holds for simulated annealing. He observes that a linear speed-up can be obtained with a maximum of 16 processors. Osborne & Gillett [1991] investigate the empirical behavior of simulated annealing for the Steiner problem in graphs. They also observe that the probability of finding a near-optimal solution is distributed exponentially. Ten Eikelder, Verhoeven, Vossen & Aarts [1995] present some results for the average-case behavior of iterated local search algorithms for the traveling salesman problem. They show that the probability of finding a solution with a small deviation over the global minimum with an iterated local search algorithm is given by a shifted exponential distribution in order to compensate for the time needed to find the first local minimum. Good efficiencies can be obtained with multiple-walk parallelism, but the amount of speed-up depends on the time needed to find the first local minimum, the time needed to find subsequent local minima, and the desired solution quality.

Another result which shows that multiple independent walks can be profitable is given by Ferreira & Zerovnik [1993]. They show that, after a certain amount of time, the probability to find a (sub)optimal solution with multiple runs of an iterative improvement algorithm is larger than the probability that it is found by a single simulated annealing run in the same amount of time.

The inherent parallelism of genetic algorithms is also based on Theorem 4.1, although in genetic algorithms the walks interact to combine good parts of individual solutions. The population size is therefore an important parameter of a parallel genetic algorithm, because it determines the speed-up and efficiency that can be obtained ultimately.

4.2 Multiple interacting walks

Interaction of walks can be used to faster direct the search into promising areas of the solution space. To model the interaction between parallel walks we use the concept of hyper neighborhoods. Hyper neighborhoods can be based on selection of a single solution from a set of solutions. Such hyper neighborhoods assign the same solution to different processors and are only useful if the walks performed by processors follow different paths from this solution. This can be achieved by either introducing randomization in walks or by enforcing different directions for walks. Other hyper neighborhoods are based on combining parts of different solutions to new solutions.

Processors exchange information when walks interact. This information, such as the occurrence of low cost solutions in a population of solutions or the availability of good parts of individual solutions, can be used to speed up the search. An example to use the additional information obtained by interaction of walks, is indicated by Lin & Kernighan [1973] and Kirkpatrick & Toulouse [1985], who observe that local minima often have many elements in common. These elements can be identified as the good parts of solutions. So, the availability of multiple local minima allows fast identification of good parts of individual solutions which should not be changed in the course of the search. This then leads to a reduction of the sizes of neighborhoods.

Implementation issues. First, we discuss asynchronous and synchronous multiple walks. In synchronous multiple-walk parallelism, walks are synchronized at certain points in time, i.e., the construction of a hyper neighbor can only be done in a state which depends on the state of all processors. In asynchronous multiple-walk parallelism construction or selection of a hyper neighbor is not synchronized at a point in time. Here, the number of processors with which a processor interacts to construct a hyper neighbor is typically a subset of all processors. In that case a processor can start computing as soon as communication with that subset of processors has occurred, and it does not need to wait for other processors.

The amount of communication overhead involved with multiple walk parallelism depends on the hyper neighborhood structure at hand, on the machine architecture, and on the mapping of walks on to processors. It is preferable to map interacting walks on to the individual processors of a given machine such that the resulting communication overhead is minimal. Load balancing is done by defining an equal amount of work to be performed between two interactions of walks. This can be done by fixing the number of steps between two communications, which typically means that hyper neighbors are not always constructed from local minima, since the number of steps to find a local minimum is not equal in each walk. An advantage of asynchronous walks over synchronous walks is their reduced load imbalance.

Examples. Parallel genetic local search algorithms are examples of local search algorithms with multiple-walk parallelism. The following hyper neighborhood is used in genetic algorithms. Two parent solutions, selected from —a subset of— the current population, are used in a cross-over operation to construct a new solution. In a cross-over operation good parts of individual solutions are combined. Both synchronous and asynchronous multiple-walk parallelism can be applied in genetic algorithms. Asynchronous parallel genetic algorithms are mostly based on an *island model* in which parents are chosen from a subset of the population instead of from the entire population. Here, we do not further elaborate on parallel genetic algorithms; for more details we refer the reader to [Jog & Van Gucht, 1987; Goldberg, 1989; Michalewicz, 1992; Mühlenbein, 1992].

Several authors have applied multiple-walk parallelism to simulated annealing and tabu search. A hyper neighborhood of a population of solutions which is often used here, is the selection of the best solution in the population. In a tabu search algorithm different paths from this solution should be followed by different processors. This can by done by blocking certain exchanges or by using different search parameters for different processors. Note that in this way an intensification of the search around good solutions is easily achieved. Additional diversification can be achieved by combining long-term memory of different walks.

Malek, Guruswamy & Pandya [1989] present tabu search and simulated annealing algorithms with synchronous multiple-walk parallelism in which the best solution found in the population is communicated to all processors after a fixed number of steps of the simulated annealing or tabu search algorithm. In their algorithm the hyper neighborhood of a solution consists of a single element, viz., the best solution encountered in the entire population. Different parameter settings are used to guarantee that different paths are followed by processors. They obtain a good speed-up for seven processors.

Crainic, Toulouse & Gendreau [1995] present various synchronous multiple-walk tabu search algorithms. They propose two hyper neighborhoods: one in which the best solution in the entire population is selected by all processors, and one in which each of the P processors selects a different solution among the P best solutions encountered since the previous interaction. Different paths are followed by choosing a different parameter setting for processors. Surprisingly, the best quality solutions are found by multiple independent walks. In [Crainic, Toulouse & Gendreau, 1993a] the same authors present asynchronous multiple-walk parallel tabu search algorithms. They propose two hyper neighborhoods in which processors only interact asynchronously with a central processor which keeps track of either the best solution or the P best solutions found in the search process, where P is the number of processors. In the first hyper neighborhood this overall best solution is selected as hyper neighbor and in the latter hyper neighborhood a hyper neighbor is randomly chosen from the P best solutions found. Compared to their synchronous algorithms they obtain slightly better final solutions. Although actual running times of the various synchronous and asynchronous algorithms are not presented, we can infer from the number of iterations at which the best solution is found, that the algorithms display little speed-up.

Aarts, De Bont, Habers & Van Laarhoven [1986] present parallel simulated annealing algorithms based on a combination of multiple interacting walks and single-step parallelism in which the size of the population is gradually decreased in the course of the search. They propose the following two hyper neighborhoods. In the first one the best solution from the entire population is chosen, i.e. $M = P$, and in the second one a solution is randomly chosen from two solutions, i.e. $M = 2$. The first hyper neighborhood results in synchronous multiple-walk parallelism, the second one in asynchronous multiple-walk parallelism. Similar algorithms using these hyper neighborhoods are implemented by Diekmann, Lüling & Simon [1993] who obtain a speed-up of 85 on a network of 120 transputers for the traveling salesman problem.

Moscato [1993] and Fox [1993] present parallel algorithms that combine genetic algorithms, simulated annealing, and tabu search. They claim a linear speed-up, but no empirical evidence is given. Mahfoud & Goldberg [1995] propose a combination of genetic algorithms and simulated annealing, but so far they do not present results for combinatorial optimization problems.

5 Single-walk parallelism

In this section we discuss a template that captures local search algorithms with single-walk parallelism. In an algorithm with single-walk parallelism one or more consecutive steps in the solution space graph are made in parallel. To this end applications of the exchange function, which we call *exchanges*, are proposed that all act on the same solution. Since effectuation of a proposed exchange can prohibit effectuation of other exchanges, the proposed exchanges are combined such that a feasible solutions is constructed, which can infer that not all proposed exchanges can be effectuated. Let τ be an exchange function as defined in Definition 2.3. Then a single-walk parallel local search algorithm consists of the following steps.

1. Partition the domain $\{(s, e_1, \ldots, e_l) \mid e_1, \ldots, e_l \in s\}$ of the exchange function τ for a given current solution $s \in \mathcal{S}$.
2. Propose exchanges for each subset simultaneously.
3. Effectuate —a subset of— the proposed exchanges in Step 2, which results in a new solution s'.
4. Replace s by s', and continue Steps 1 to 3 until some stopping criterion is met.

Single-step parallel local search which is based on parallel neighborhood exploration, is a special case of the above concept. In single-step parallelism each processor examines a part of the neighborhood of the current solution s, and only a single exchange is effectuated in Step 3. So, speed-up is only achieved in Step 2. Single-step parallelism implies that, in terms of a walk in the solution space graph, only a single step is made, as solution s' is obtained from s with a single exchange. In an algorithm with multiple-step parallelism several exchanges are effectuated in Step

3, which implies that a solution s' is constructed which can only be reached from s by performing multiple steps in the solution space graph, i.e., multiple exchanges have to be performed to obtain s' from s; see Figure 2.

Below, we present a machine-independent template for single-walk parallel local search algorithms. For this we introduce the concept of distributed neighborhood structures which have to deal with the following issues, also come across in the above algorithmic concept. A distributed neighborhood has to specify how domains of the exchange function and solutions are partitioned. Furthermore, it has to be specified how proposed exchanges are combined. These aspects are captured in the following definitions.

Definition 5.1. Let $\mathcal{S}$ be a set of solutions, $\mathcal{U} = \bigcup_{s\in\mathcal{S}} \mathcal{P}(s)$ a set of *partial* solutions, and let $\Omega = \{1, \ldots, P\}$ be a set of processors. A *solution distribution* δ gives a partial solution δ_p for each processor $p \in \Omega$. A *local neighborhood structure* $\mathcal{L} : \mathcal{U} \to \mathcal{P}(\mathcal{U})$ gives for each partial solution a set of partial solutions. □

Similar to traditional neighborhoods, local neighborhoods are not given explicitly, but are constructed using an exchange function. The following definition states how this is done.

Definition 5.2. Let τ be an exchange function that can be applied to partial solutions. A *domain distribution* $\lambda_p : \mathcal{U} \to \mathcal{P}(\mathcal{E}^l)$ gives for each processor $p \in \Omega$ and partial solution $u \in U$ a set of arguments of the exchange function τ. A *local neighborhood* $\mathcal{L}(u) = \{\tau(u, e_1, \ldots, e_l) \mid (e_1, \ldots, e_l) \in \lambda_p(u)\}$ for $u \in \mathcal{U}$. □

The domain distribution describes which elements of a partial solution are the arguments used by the exchange function to construct local neighbors. If solutions are partitioned, exchanges involving elements of different partial solutions cannot occur, and therefore several partitions need to be examined. This issue is also described in the following definition of a distributed neighborhood structure.

Definition 5.3. Let $\mathcal{S}$ be a set of solutions, $\mathcal{U}$ a set of partial solutions, Ω a set of P processors, and τ an exchange function. Then, a *distributed neighborhood structure* $\mathcal{D}$ is a triple (Δ, λ, ϕ) defined as follows. A *distribution structure* Δ gives for each solution s and set of processors Ω a set of solution distributions, with $\Delta(s, \Omega) \subseteq \{\delta \in \Omega \to \mathcal{U} \mid \bigcup_{p\in\Omega} \delta_p = s\}$. λ gives for each $p \in \Omega$ a domain distribution $\lambda_p : \mathcal{U} \to \mathcal{P}(\mathcal{E}^l)$ with $l \in \mathbb{N}^+$, which gives the arguments of the exchange function τ used to construct the local neighborhood $\mathcal{L}(\delta_p)$ of a partial solution δ_p. A *combination function* $\phi : \mathcal{U}^P \to \mathcal{S}$ combines P partial solutions to a feasible solution. □

A distributed neighborhood structure consists of a distribution structure that specifies how solutions are distributed over processors, a domain distribution that defines local neighborhoods, and a combination function that specifies how proposed exchanges have to be combined. Note that only a subset of the proposed exchanges might be effectuated by the combination function, i.e., the combination function does not always effectuate all proposed exchanges since this might lead to infeasible solutions. Next we formalize the notion of local optimality for a distributed neighborhood structure.

Definition 5.4. Given are a problem instance $(\mathcal{S}, f)$ and a distributed neighborhood structure $\mathcal{D} = (\Delta, \lambda, \phi)$. Then a solution $s \in \mathcal{S}$ is a local minimum of $\mathcal{D}$, if for each solution distribution

---

proc *Single_Walk* $(s : \mathcal{S})$
  var $\delta : \Omega \to \mathcal{U}$; $D : \mathcal{P}(\Omega \to \mathcal{U})$;
  begin
    $D := \emptyset$;
    while $\Delta(s, \Omega) \setminus D \neq \emptyset$ **do**
      $\delta :\in \Delta(s, \Omega) \setminus D$;
      par $p \in \Omega$ **do** *Iterative_Improvement* (δ_p) **rap**;
      if gain found **then** $D := \emptyset$; $s := \phi(\delta_1, \ldots, \delta_P)$
      else $D := D \cup \{\delta\}$ **fi**
    od$\{s$ is a local minimum w.r.t. $\mathcal{D}\}$
  end

Figure 4: A single-walk parallel local search algorithm.

$\delta \in \Delta(s, P)$, processor $p \in \Omega$, and partial neighbor $\delta'_p \in \mathcal{L}(\delta_p)$ holds $f(\delta'_p) \geq f(\delta_p)$. □

Using the concept of distributed neighborhood structures we can formulate the template for single-walk parallel local search given in Figure 4. In this algorithm first a distribution of the current solution is chosen. Then, all processor simultaneously propose exchanges. Subsequently, a subset of the proposed exchanges is effectuated to obtain a new feasible solution. This process is repeated until no solution distribution of the current solution can be improved upon, at which point a local optimum has been found.

To guarantee termination of the algorithm, ϕ should combine a set of neighbors of s in such a way that, if a neighbor has lower cost than s, their combination also has lower cost than s. We define this termination condition as follows.

Definition 5.5. Given are a problem instance $(\mathcal{S}, f)$ and a distributed neighborhood structure $\mathcal{D} = (\Delta, \lambda, \phi)$. Then, the combination function ϕ is *progressive*, if for all $\delta \in \Delta(s, P)$, $p \in \Omega$, and $s \in \mathcal{S}$ holds $\exists_{\delta'_p \in \mathcal{L}(\delta_p)} f(\delta'_p) < f(\delta_p) \Rightarrow f(\phi(\delta'_1, \ldots, \delta'_P)) < f(s)$. □

An important issue is raised by the question how we can compare a distributed neighborhood structure $\mathcal{D}$ with a traditional neighborhood structure $\mathcal{N}$. This enables a comparison of the solutions found by a parallel local search algorithm with those found by a sequential local search algorithm. The following definition is useful for such a comparison.

Definition 5.6. Given are a problem instance $(\mathcal{S}, f)$, a neighborhood structure $\mathcal{N}$, and a distributed neighborhood structure $\mathcal{D}$. Then, $\mathcal{D}$ is called *isomorphic* with $\mathcal{N}$, if each local minimum of $\mathcal{N}$ is a local minimum of $\mathcal{D}$, and vice versa. □

If a distributed neighborhood structure $\mathcal{D}$ is isomorphic with neighborhood structure $\mathcal{N}$, then the expected average cost of final solutions found by a sequential and parallel algorithm, are equal, provided that a parallel local search algorithm finds a given local minimum with the same probability as a sequential local search algorithm.

Implementation issues. First, we discuss synchronous and asynchronous single-walk parallel algorithms. In an algorithm with synchronous single-walk parallelism some step of the algorithm can only be performed in state dependent on the state of all processors. This can occur for example when global communication has to take place before a next step of the algorithm can be performed. Synchronization is typically required in a decentralized algorithm in which each processor has a copy of the current solution. In order to update all local copies of the current solution after some exchanges have been proposed, global communication is required. If there is a central control mechanism which manages access to the current solution, no inconsistencies in the current solution can occur. So, processors can then proceed independently from each other, although it may occur that some of the proposed exchanges cannot be effectuated, because the current solution has been altered by another processor. The choice between synchronous or asynchronous parallelism is often determined by characteristics of the parallel machine at hand, such as the availability of common memory or the configurability of a machine in a master/slave model. As we want to abstract from machine-dependent issues, we do not further elaborate on this subject.

The speed-up obtained with asynchronous and synchronous single-walk parallelism depends on problem characteristic and on the architecture of the machine on which the algorithm is implemented. Two important issues to obtain a good speed-up are load balancing and communication overhead, which briefly address below.

In many single-walk parallel local search algorithms global communication is necessary, because effectuation of an exchange has a global impact on the characteristics of solution. If a machine allows direct communication with a central processor, or if it has common memory from which all processors can read efficiently, then the overhead of asynchronous global communication is limited. In many other cases, however, global communication between all processors causes a substantial communication overhead.

Load balancing is needed to reduce the amount of time processors are idle. In synchronous parallelism load balancing must be done by assigning equal amounts of work to all processors in between synchronization points. In asynchronous single-walk parallelism load balancing is only needed for processors that interact.

Finally, the efficiency of single-walk parallelism is also determined by the ratio between the time needed to evaluate the cost of a neighbor and the time needed for combination and effectuation of proposed exchanges. For this reason the data structures used to represent a solution are important, since these data structures determine the efficiency of operations that have to be performed to combine and effectuate proposed exchanges.

5.1 Single-step parallelism

In the previous section, we have introduced a template to capture single-walk parallel local search algorithms. In order to apply single-step parallelism to a given problem, the distributed neighborhood of Definition 5.3 is instantiated as follows. The distribution structure is chosen as $\Delta(s, \Omega) = \{\delta : \Omega \to \mathcal{S} \mid \delta_p = s \text{ for all } p \in \Omega\}$ and the combination function is given by $\phi(s_1, \ldots, s_P) = s_i$ with $1 \leq i \leq P$. Moreover, it is not difficult to define a domain distribution that partitions a traditional neighborhood, which guarantees that the distributed neighborhood is isomorphic with it. The distributed neighborhood of Definition 5.3 is now straightforward, since a complete solution is assigned to each processor, and the combination function simply effectuates one of the proposed exchanges. This means that the combination function only has to select a solution from the proposed solutions.

An advantage of single-step parallelism is its general applicability. Moreover, it results trivially in a distributed neighborhood structure that is isomorphic with a traditional neighborhood structure that uses the same exchange function. So, it is easily guaranteed that the parallel algorithm finds the same quality solutions as a sequential algorithm.

A disadvantage of single-step parallelism is the amount of speed-up that can be obtained. A local search algorithm with single-step parallelism explores neighbors of a solution simultaneously. Subsequently, one of the neighbors replaces the current solution resulting in a single step in the solution space graph. The amount of speed-up that can be obtained by single-step parallelism strongly depends on the number of exchanges that has to be examined before a proposed exchange is accepted. If best-improvement is used as pivoting rule, then single-step parallelism gives good speed-up, because to find the neighbor with lowest cost the entire neighborhood of a solution has to be examined. Best-improvement is often used in tabu search algorithms, so single-step parallelism can be effectively applied in tabu search algorithms. It is, however, questionable whether best-improvement is always the most effective pivoting rule, since for many problems the same quality solutions can be found with first-improvement in smaller running times.

For algorithms with first-improvement the amount of speed-up that can be obtained by single-step parallelism depends on the stage of the search process that has been reached by the algorithm. If the current solution has few cost improving neighbors, a situation arises which is almost similar to best-improvement, since then it is more likely that a large proportion of neighbors has to be examined before a solution with lower cost is found. If the fraction of neighbors that would be accepted by a local search algorithm at a certain stage of the search is given by α, then the maximum speed-up with single-step parallelism is $\frac{1}{\alpha}$, regardless of the number of processors; see also [Roussel-Ragot & Dreyfus, 1990; Azencott, 1992] for a theoretical analysis of the speed-up that can be obtained by single-step parallelism in simulated annealing. Generally, α is large in the beginning of the search process, and consequently the resulting speed-up is small. As the cost of the current solution decreases, α increases. So, for a local search algorithm with first improvement, single-step parallelism can often only be applied efficiently in a later stage of the search process.

Examples. One the first examples of single-step parallelism in the literature is the algorithm of Kravitz & Rutenbar [1987], who apply asynchronous single-step parallelism to simulated annealing for the cell placement problem. They report a speed-up of 2.5 on a shared memory machine with four processors. Diekmann, Lüling & Simon [1993] present asynchronous and synchronous single-walk parallel simulated annealing algorithms for the traveling salesman problem and the link assignment problem. Their algorithms are implemented on a network of 120 transputers. They obtain a speed-up of 35 for the traveling salesman problem, and a speed-up of 70 for the link assignment problem. Kindervater, Lenstra & Savelsbergh [1993] present a local search algorithm with synchronous single-step parallelism for the time-constrained traveling salesman problem. Verification of local optimality can be done in $\mathcal{O}(\log N)$ time with $\mathcal{O}(N^2/\log N)$ processors on a PRAM machine, but no empirical results on a parallel machine are given. Synchronous single-step parallelism is used by Ravikumar [1992] in an iterative best improvement algorithm for the traveling salesman problem.

As already mentioned, many tabu search algorithms also use best improvement as pivoting rule. Synchronous single step parallelism can then be applied effectively, as is shown by Taillard [1990, 1991] for the flow shop sequencing problem and the quadratic assignment problem. Chakrapani & Skorin-Kapov [1993a, 1993b], use synchronous single-step parallelism in combination with best improvement for the traveling salesman problem and the quadratic assignment problem, respectively. Their algorithm is implemented on a massively parallel Connection Machine with 16,384 processors, but running times are only compared to an implementation on a work station. They also report that 55% of the time is spent in communication. Li & Pardalos [1992] use synchronous single-step parallelism in a parallel variable depth algorithm for the quadratic assignment problem.

5.2 Multiple-step parallelism

Essential to achieve a good speed-up with multiple-step parallelism is that a large proportion of the proposed exchanges can be effectuated in the combination step. The applicability of multiple-step parallelism therefore depends strongly on the problem at hand, as the problem must allow effectuation of several proposed exchanges for a given solution.

Two approaches for designing distributed neighborhood structures can be distinguished based on whether or not solutions are partitioned into partial solutions. If the cost difference between two neighboring solutions can be computed using only information on the removed and inserted parts of a solution, then it might be profitable to partition a solution and assign each partial solution to a processor. This enables an efficient implementation of the combination function ϕ, because a new solution can then be constructed without global communication. A disadvantage, however, is that exchanges which involve elements of different partial solutions, cannot be proposed. In order to find the same quality solutions as a sequential local search algorithm, different distributions have to be examined, which might lead to a complex distribution structure.

Another approach in the design of distributed neighborhood structures is to provide each processor with a copy of the current solution and to define an appropriate domain distribution. This often leads to a distributed neighborhood that is trivially isomorphic with a traditional neighborhood that uses the same exchange function. To achieve speed-up, however, an effective combination function is essential, because effectuation of an exchange can prohibit effectuation of other exchanges to guarantee feasibility of solutions. Furthermore, global communication is often required in the combination function, which can lead to a large communication overhead depending on the machine architecture.

Examples. The TSP of Example 2.1 is a typical example of a problem for which partitioning of a solution is often used. Here a solution is a Hamilton cycle, and the cost difference between neighbors can be computed using only the lengths of removed and inserted edges. Moreover, it is possible to combine proposed exchanges in a straightforward way without extensive communication. Most of the examples we discuss below are implemented on a distributed-memory MIMD machine and use synchronous parallelism.

One of the first applications of multiple-step parallelism to the TSP is presented by Felten, Karlin & Otto [1985] who divide a tour in consecutive paths. Each path is assigned to a different processor. A new distribution is obtained by assigning adjacent edges that are assigned to different processors to the same processor. Fiechter [1994] presents a parallel tabu search algorithm for the TSP which uses a similar solution distribution. A partial solution assigned to a processor consists of one path in the tour. New distributions of a tour are obtained by assigning different paths to processors.

Allwright & Carpenter [1989] present a parallel simulated annealing algorithm for the TSP, based on a distribution of a tour over a linear array of processors. A partial solution in this distribution consists of two non-adjacent paths in the tour. Edges are randomly reassigned to processors to obtain a new distribution. In [Verhoeven, Aarts, Van de Sluis & Vaessens, 1992; Verhoeven, Aarts & Swinkels, 1995] we present distributed neighborhood structures for the TSP, based on a similar solution distribution. We prove that our distributed neighborhoods are isomorphic with the well-known 2-exchange and 3-exchange neighborhoods. Here, a new distribution is obtained by assigning a single edge to a new processor, which leads to synchronization between processors. Verhoeven, Swinkels & Aarts [1995] present a parallel Lin-Kernighan algorithm based on domain distribution. Each processor works on a copy of the entire tour and the proposed exchanges are subsequently combined to a new tour, where as many as possible exchanges are effectuated in the combination function.

Bachem & Wottawa [1992] propose a solution distribution based on a geometric partition of the cities in a TSP instance. The partial solutions, which consists of Hamilton paths, are combined by a tour construction heuristic. A new distribution is obtained by constructing a new geometric partition.

Another problem for which many examples of multiple-step parallelism can be found is the placement of cells in circuit layouting. Shahookar & Mazunder [1991] present an overview of this field. In most of the examples simulated annealing is used. For cell placement problems a local change might have a global impact on a solution, since the cost of a solution is determined by the relative position of pairs of cells. Without restrictions, effectuation of proposed exchanges can lead to erroneous cost computation. Erroneous cost computation changes the convergence behavior of simulated annealing, but various empirical results in the following papers show that simulated annealing for cell placement is rather robust to errors in the cost evaluation. Romeo & Sangiovanni-Vincentelli [1991] present some theoretical results on erroneous cost evaluation in simulated annealing.

Darema, Kirkpatrick & Norton [1987] use a solution distribution in which they assign the cells to processors, so each partial solution consists of a subset of the set of cells, and all proposed exchanges are accepted by the combination function. A new distribution is obtained by randomly reassigning different cells to each processor. They compare two solution distributions: one that guarantees correct cost computations, and one that allows errors in the cost computation. A drawback of the first approach is that only a limited number of processors can be used which prohibits scaling of the algorithm. The second approach does not suffer from this drawback. Their empirical results show that both approaches find the same quality final solutions. Jones & Banerjee [1987] present a similar algorithm in which they also partition a solution. A new distribution is obtained by exchanging cells between neighboring processors, and the combination function accepts all proposed exchanges, which may lead to erroneous cost computations. The algorithm is implemented on a distributed MIMD machine. Casotto, Romeo & Sangiovanni-Vincentelli [1987] use a local search heuristic for finding a new partition of the solution, in order to minimize the probability that an error in the cost computation occurs. The erroneous cost computations are caused by accepting all proposed exchanges without checking the position of cells in the newly constructed solution. The algorithm is implemented on a shared-memory parallel machine.

A hybrid algorithm that uses both multiple-walk parallelism and multiple-step parallelism is presented by Rose, Snelgrove & Vranesic [1988]. They apply multiple-walk parallelism at high temperatures of the simulated annealing algorithm and multiple-step parallelism during low temperature annealing, because at low temperatures erroneous cost computation is less likely to occur due to the low acceptance rate.

Barbosa & Gafni [1989] present a simulated annealing algorithm with multiple-step parallelism that can be applied to problems for which only a single element is replaced in the neighborhood structure. Moreover, such an exchange must have limited interaction with other elements of a solution. They propose a solution partitioning approach that guarantees that no erroneous cost computation can occur when all proposed exchanges are effectuated, at the cost of a limited number of employable processors. As a consequence, the maximum speed-up that can be obtained with their approach depends on characteristics of the problem instance at hand. An example of such a problem is the vertex cover problem [Barbosa & Boeres, 1990].

Another example of multiple-step parallelism is the algorithm of Savage & Wloka [1991] for the graph partitioning problem who propose a solution distribution based on partitioning a solution. In order to deal with erroneous cost computation, the combination function either accepts all proposed exchanges if the resulting global solution has lower cost, or rejects all proposed exchanges if the resulting global solution has higher cost.

Taillard [1993] and Garcia, Potvin & Rousseau [1994] propose parallel tabu search algorithms for the vehicle routing problem. Taillard proposes a solution distribution in which complete, geographically close routes are preferably assigned to processors that are neighbors in the processor network on which the algorithm is implemented. A new distribution is obtained by assigning elements to neighboring processors. Garcia, Potvin & Rousseau propose a domain distribution for the vehicle routing problem with time windows. Their combination function consecutively effectuates the proposed exchanges in order of descending gain. Abramson [1991] presents a simulated annealing algorithm with multiple-step parallelism for the time tabling problem.

6 Complexity issues of parallel local search

In this section we discuss some issues related to the parallel complexity of finding local optima. For an overview of the complexity theory of parallel computations the reader is referred to [Parberry, 1987].

If PLS-complete problems require super-polynomial running times, as is conjectured by Yannakakis [1995], then parallel algorithms that find local minima using a polynomially bounded number of processors, which is the case in any realistic machine model, will also run in super-polynomial time. Therefore, we restrict ourselves to the parallel complexity of the *verification problem*, i.e., the problem of deciding whether a solution is a local optimum. Consider a problem instance $(\mathcal{S}, f)$ and a neighborhood structure $\mathcal{N}$, then we have the following result.

Theorem 6.1. *Let* $|\mathcal{N}(s)| = k > 1$ *for all* $s \in \mathcal{S}$, *and let* $c(n)$ *be the complexity of deciding whether* $f(s') \geq f(s)$ *for a given solution* $s' \in \mathcal{N}(s)$, *where* n *denotes the size of a solution. Then, a parallel algorithm for the verification problem exists that uses* $k/\log k$ *processors and which has a time complexity of* $\mathcal{O}(c(n)\log k)$ *and an efficiency of* $\mathcal{O}(1)$.

Proof. Use an algorithm with synchronous single-step parallelism, and partition the neighborhood in sets that contain $\log k$ solutions; cf. [Brent, 1974] □

Kruskal, Rudolph & Snir [1990] introduce a parallel complexity class $\mathcal{EP}$ which contains problems for which an efficient parallel algorithm on a PRAM machine exists. A problem is in $\mathcal{EP}$ if there exists a parallel algorithm with speed-up that scales with the problem size and for which the computational effort, i.e., the time complexity of the parallel algorithm multiplied by the number of processors, is equal to the time complexity of the sequential algorithm. The class $\mathcal{EP}$ contains the class $\mathcal{NC}$. Theorem 6.1 now leads to the following corollary.

Corollary 6.1. *Assume that the size k of a neighborhood is bounded by a polynomial function in the instance size. Then, the verification problem is in the class $\mathcal{EP}$. In particular, if the problem of deciding whether $f(s') \geq f(s)$ for $s \in \mathcal{S}$ and $s' \in \mathcal{N}(s)$ is in $\mathcal{NC}$, then the verification problem w.r.t. $\mathcal{N}$ is in $\mathcal{NC}$.*

Proof. Let T_p be the time required by an algorithm for the verification problem with p processors. Then for $p = k/\log k$, it is easily verified that $T_p = \mathcal{O}(T_1{}^{\epsilon})$ with $\epsilon < 1$ and $T_p \cdot p = \mathcal{O}(T_1)$. So, the verification problem is in $\mathcal{EP}$. □

Corollary 6.1 shows that verification of local optimality can be done in polylogarithmic time with a polynomially bounded number of processors, if the cost evaluation of a neighbor can be done in polylogarithmic time. This typically holds when the cost evaluation of a neighbor is $\mathcal{O}(1)$, for example in the 2-exchange neighborhood for the TSP. Corollary 6.1 shows that it is possible to design an efficient parallel algorithm for the verification problem with speed-up that is proportional to the size of the problem. The running time of this parallel algorithm can still be polynomial, but in practical situations one is satisfied with parallel algorithms with a good speed-up that scales with the instance size.

Another important topic is the complexity of finding a local optimum with respect to a distributed neighborhood structure. The notion of PLS-completeness and the definition of the class $\mathcal{PLS}$ can also be applied to the problem of finding a local optimum with respect to a distributed neighborhood structure. Using this extension, we obtain the following straightforward result.

Theorem 6.2. *Let a distributed neighborhood structure $\mathcal{D}$ be isomorphic with a neighborhood $\mathcal{N}$. If the problem of finding a local optimum with respect to $\mathcal{N}$ is PLS-complete, then the problem to find a local optimum with respect to $\mathcal{D}$ is PLS-complete.* □

Finally, we discuss upper bounds on the speed-up that can be obtained with the various approaches discussed in this paper. We assume that we have an infinite number of processors at our disposal. The number of processors that can be employed in multiple-walk parallelism is not bounded by a polynomial in the size of the instance, so here we do not have an upper bound on the maximum speed-up. In an algorithm with single-walk parallelism the maximum speed-up is at most the size of the domain of the exchange function, because at least one exchange has to be evaluated by each processor that contributes to the speed-up. For most neighborhoods the domain size is low-order polynomial. In practice the number of processors of a parallel machine is typically much smaller than the instance size, so in these cases the number of available processors is a trivial upper bound on the maximum speed-up that can be obtained.

7 Conclusions

We have presented a survey of parallel local search algorithms based on a distinction between multiple-walk and single-walk algorithms using hyper neighborhoods and distributed neighborhoods, respectively. In the class of single-walk algorithms we distinguished between single-step and multiple-step parallel local search. We have shown that most of the parallel local search algorithms existing in the literature fit well into this frame work. Furthermore, we have discussed some complexity issues of parallel local search.

An important observation is that under mild conditions good speed-ups can be obtained with multiple independent walks carried out in parallel. Multiple-walk parallelism therefore offers a promising avenue to parallel tabu search and simulated annealing algorithms for a wide range of problems. Single-step parallelism can primarily be used in local search algorithms which use a best improvement pivoting rule, as is the case for most tabu search algorithms. Multiple-step parallelism seems to be the best approach to design parallel local search algorithms tailored to the problem at hand. The design of distributed neighborhoods deserves special attention in this respect. The effectiveness of both single-step and multiple-step parallelism is machine dependent, due to load balancing and communication overhead issues. A general conclusion is that local search can be accelerated considerably by execution on parallel machines.

References

AARTS, E.H.L., F.M.J. DE BONT, J.H.A. HABERS, AND P.J.M. VAN LAARHOVEN [1986], Parallel implementations of the statistical cooling algorithm, *Integration* **4**, 209–238.

AARTS, E.H.L. AND J. KORST [1989], *Simulated Annealing and Boltzmann Machines*, Wiley & Sons.

AARTS, E.H.L. AND J.K. LENSTRA (eds.) [1995], *Local Search in Combinatorial Optimization*, Wiley.

ABRAMSON, D. [1991], Constructing school timetables using simulated annealing: sequential and parallel algorithms, *Management Science* **37**, 98–113.

ALLWRIGHT, J.R.A. AND D.B. CARPENTER [1989], A distributed implementation of simulated annealing for the traveling salesman problem, *Parallel Computing* **10**, 335–338.

AZENCOTT, R. (ed.) [1992], *Simulated Annealing: Parallelization Techniques*, Wiley, New-York.

BACHEM, A. AND M. WOTTAWA [1992], *Parallelization of heuristics for large Traveling Salesman Problems*, Report 92.119, Universität Köln, Germany.

BARBOSA, V.C. AND M.C.S. BOERES [1990], An Occam-based evaluation of a parallel version of simulated annealing, *Microprocessing and Microprogramming* **30**, 85–92.

BARBOSA, V.C. AND E. GAFNI [1989], A distributed implementation of simulated annealing, *J. of Parallel and Distributed Computing* **6**, 411–434.

BATTITI, R. AND G. TECCHIOLLI [1992], Parallel biased search for combinatorial optimization: genetic algorithms and tabu, *Microprocessors and Microsystems* **16**, 351–367.

BRENT, R.P. [1974], The parallel evaluation of general arithmetic expressions, *Journal ACM* **21**, 201–206.

CASOTTO, A., F. ROMEO, AND A. SANGIOVANNI-VINCENTELLI [1987], A parallel simulated annealing algorithm for the placement of macro-cells, *IEEE Transactions on Computer-Aided Design* **6**, 838–847.

CHAKRAPANI, J. AND J. SKORIN-KAPOV [1993a], Connection machine implementation of a tabu search algorithm for the traveling salesman problem, *J. of Computing and Information Technology* **1**, 29–36.

CHAKRAPANI, J. AND J. SKORIN-KAPOV [1993b], Massively parallel tabu search for the quadratic assignment problem, *Annals of Operations Research* **41**, 327–342.

CRAINIC, T.G., M. TOULOUSE, AND M. GENDREAU [1993a], *Parallel asynchronous tabu search for multicommodity location-allocation with balancing requirements*, Publication 935, Centre de recherche sur les transports, Université de Montréal.

CRAINIC, T.G., M. TOULOUSE, AND M. GENDREAU [1993b], *Towards a taxonomy of parallel tabu search algorithms*, Publication 933, Centre de recherche sur les transports, Université de Montréal.

CRAINIC, T.G., M. TOULOUSE, AND M. GENDREAU [1995], Synchronous tabu search parallelization strategies for multicommodity location-allocation with balancing requirements, *OR Spektrum* **17**.

DAREMA, F., S. KIRKPATRICK, AND V.A. NORTON [1987], Parallel algorithms for chip placement by simulated annealing, *IBM Journal of Research and Development* **31**, 391–402.

DIEKMANN, R., R. LÜLING, AND J. SIMON [1993], Problem independent distributed simulated annealing and its applications, in: R.V.V. Vidal (ed.), *Applied simulated annealing*, LNEMS 396, Springer-Verlag, 18–44.

DODD, N. [1990], Slow annealing versus multiple fast annealing runs - an empirical investigation, *Parallel Computing* **16**, 269–272.

EIKELDER, H.M.M. TEN, M.G.A. VERHOEVEN, T.W.M. VOSSEN, AND E.H.L. AARTS [1995], A probabilistic analysis of local search, in: I.H. Osman and J.P. Kelly (eds.), *Metaheuristics: The State of the Art 1995*, Kluwer Academic Publishers.

FELTEN, E., S. KARLIN, AND S.W. OTTO [1985], The traveling salesman problem on a hypercubic mimd machine, *Proc. Int. Conference on Parallel Processing*, 6–10.

FERREIRA, A.G. AND J. ZEROVNIK [1993], Bounding the probability of success of stochastic methods for global optimization, *Computers & Mathematics with Applications* **25**, 1–8.

FIECHTER, C.N. [1994], A parallel tabu search algorithm for large traveling salesman problems, *Discrete Applied Mathematics* **51**, 243–267.

FOX, B.L. [1993], Integrating and accelerating tabu search, simulated annealing, and genetic algorithms, *Annals of Operations Research* **41**, 47–67.

GARCIA, B.L., J.Y. POTVIN, AND J.M. ROUSSEAU [1994], A parallel implementation of the tabu search heuristic for vehicle routing problems with time window constraints, *Computer Ops. Res.* **21**, 1025–1033.

GAREY, M.R. AND D.S. JOHNSON [1979], *Computers and Intractability: A Guide to the Theory of NP-completeness*, Freeman and Co., New York.

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

GLOVER, F., E. TAILLARD, AND D. DE WERRA [1993], A user's guide to tabu search, *Annals of Operations Research* **41**, 3–28.

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

GREENING, D.R. [1990], Parallel simulated annealing techniques, *Physica D* **42**, 293–306.

JOG, P. AND D VAN GUCHT [1987], Parallelisation of probabilistic sequential search algorithms, *Proc. 2nd International Conference on Genetic Algorithms*, 170–176.

JOHNSON, D.S. [1990], Local optimization and the traveling salesman problem, *Proc. 17th Colloquium on Automata, Languages, and Programming*, LNCS 443, 446–461.

JOHNSON, D.S., C.H. PAPADIMITRIOU, AND M. YANNAKAKIS [1988], How easy is local search?, *J. of Computer and System Sciences* **37**, 79–100.

JONES, M.H. AND P. BANERJEE [1987], An improved simulated annealing algorithm for standard cell placement, *Proc. of the International Conference on Computer Design*, 83–86.

KERNIGHAN, B.W. AND S. LIN [1970], An efficient heuristic procedure for partitioning graphs, *Bell Systems Technical Journal* **49**, 291–307.

KINDERVATER, G.A.P., J.K. LENSTRA, AND M.W.P. SAVELSBERGH [1993], Sequential and parallel local search for the time constrained traveling salesman problem, *Discrete Applied Mathematics* **42**, 211–25.

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

KIRKPATRICK, S. AND G. TOULOUSE [1985], Configuration space analysis of Travelling Salesman Problems, *Journal Physique* **46**, 1277–1292.

KRAVITZ, S.A. AND R.A. RUTENBAR [1987], Placement by simulated annealing on a multiprocessor, *IEEE Transactions on Computer Aided Design* **6**, 534–549.

KRUSKAL, C.P., L. RUDOLPH, AND M. SNIR [1990], A complexity theory of efficient parallel algorithms, *Theoretical Computer Science* **71**, 95–132.

LI, Y. AND P.M. PARDALOS [1992], Parallel algorithms for the quadratic assignment problem, in: P.M Pardalos (ed.), *Advances in Optimization and Parallel Computing*, 177–189.

LIN, S. [1965], Computer solutions of the traveling salesman problem, *Bell System Technical Journal* **44**, 2245–2269.

LIN, S. AND B.W. KERNIGHAN [1973], An effective heuristic algorithm for the traveling salesman problem, *Operations Research* **21**, 498–516.

LUEKER, G. [1975], *Two NP-complete problems in nonnegative integer programming*, manuscript, Princeton University.

MAHFOUD, S.W. AND D.E. GOLDBERG [1995], Parallel recombinative simulated annealing: a genetic algorithm, *Parallel Computing* **21**, 1–28.

MALEK, M., M. GURUSWAMY, AND M. PANDYA [1989], Serial and parallel simulated annealing and tabu search for the traveling salesman problem, *Annals of Operations Research* **21**, 59–84.

MARTIN, O., S.W. OTTO, AND E.W. FELTEN [1991], Large-steps markov chains for the travelling salesman problem, *Complex Systems* **5**, 299–326.

MICHALEWICZ, Z. [1992], *Genetic algorithms + Data structures = Evolution programs*, Springer-Verlag, Berlin.

MOSCATO, P. [1993], An introduction to population approaches for optimization and hierarchical objective functions: a discussion on the role of tabu search, *Annals of Operations Research* **41**, 85–122.

MÜHLENBEIN, H. [1992], Parallel genetic algorithms in combinatorial optimization, in: O. Balci (ed.), *Computer Science and Operations Research*, Pergamon Press.

MÜHLENBEIN, H., M. GORGES-SCHLEUTER, AND O. KRÄMER [1988], Evolution algorithms in combinatorial optimization, *Parallel Computing* **7**, 65–85.

OSBORNE, L.J. AND B.E. GILLETT [1991], A comparison of two simulated annealing algorithms applied to the directed Steiner problem on networks, *ORSA Journal on Computing* **3**, 213–225.

PAPADIMITRIOU, C.H. AND K. STEIGLITZ [1982], *Combinatorial Optimization: algorithms and complexity*, Prentice Hall.

PARBERRY, I. [1987], *Parallel Complexity Theory*, Pitman, London.

RAVIKUMAR, C.P. [1992], Parallel techniques for solving large scale travelling salesperson problems, *Microprocessors and Microsystems* **16**, 149–158.

REEVES, C.R. (ed.) [1993], *Modern Heuristic Techniques for Combinatorial Problems*, Blackwell.

ROMEO, F. AND A. SANGIOVANNI-VINCENTELLI [1991], A theoretical framework for simulated annealing, *Algorithmica* **6**, 302–345.

ROSE, J.S., W.M. SNELGROVE, AND Z.G. VRANESIC [1988], Parallel standard cell placement algorithms with quality equivalent to simulated annealing, *IEEE Transactions on Computer-Aided Design* **7**, 387–396.

ROUSSEL-RAGOT, P. AND G. DREYFUS [1990], A problem independent parallel implementation fo simulated annealing: models and experiments, *IEEE Transactions on Computer-Aided Design* **9**, 827–835.

SAVAGE, J.E. AND M.G. WLOKA [1991], Parallelism in graph-partioning, *Journal of Parallel and Distributed Computing* **13**, 257–272.

SECHEN, C. [1988], *VLSI Placement and global routing using simulated annealing*, Kluwer Academic.

SHAHOOKAR, K. AND P. MAZUNDER [1991], VLSI cell-placement techniques, *ACM Computing Surveys* **23**, 143–220.

TAILLARD, E. [1990], Some efficient heuristic methods for the flow shop sequencing problem, *European Journal of Operations Research* **47**, 65–74.

TAILLARD, E. [1991], Robust taboo search for the quadratic assignment problem, *Parallel Computing* **17**, 443–455.

TAILLARD, E. [1993], Parallel iterative search methods for vehicle routing problems, *Networks* **23**, 661–673.

TAILLARD, E. [1994], Parallel taboo search techniques for the job shop scheduling problem, *ORSA Journal on Computing* **6**, 108–117.

VAESSENS, R.J.M., E.H.L. AARTS, AND J.K. LENSTRA [1992], A local search template, *Parallel Problem Solving from Nature 2*, 65–74.

VERHOEVEN, M.G.A., E.H.L. AARTS, E. VAN DE SLUIS, AND R.J.M. VAESSENS [1992], Parallel local search and the travelling salesman problem (extended abstract), *Proc. of Parallel Problem Solving from Nature 2*, Amsterdam, North-Holland, 543–552.

VERHOEVEN, M.G.A., E.H.L. AARTS, AND P.C.J. SWINKELS [1995], A parallel 2-opt algorithm for the traveling salesman problem, *Future Generation Computer Systems* **11**, 175–182.

VERHOEVEN, M.G.A., P.C.J. SWINKELS, AND E.H.L. AARTS [1995], *Parallel Local Search and the Traveling Salesman Problem*, working document.

VOSS, S. [1993], Tabu search: applications and prospects, in: D.Z. Du and P.M. Pardalos (eds.), *Network Optimization Problems*, World Scientific Publishing Co., 333–353.

YANNAKAKIS, M. [1990], The analysis of local search problems and their heuristics, *Proc. 7^{th} Symposium on Theoretical Aspects of Computer Science*, LNCS 415, Springer-Verlag, 298–311.

YANNAKAKIS, M. [1995], Computational complexity of local search, in: [Aarts & Lenstra, 1995].

YEONG, C.S. AND M.H. KIM [1991], Fast parallel simulated annealing for traveling salesman problem on SIMD machines with linear interconnections, *Parallel Computing* **17**, 221–228.