J Heuristics (2011) 17:637–658
DOI 10.1007/s10732-010-9149-8

Matching based very large-scale neighborhoods for parallel machine scheduling

Tobias Brueggemann · Johann L. Hurink

Received: 6 December 2007 / Revised: 14 September 2010 / Accepted: 1 November 2010 /
Published online: 11 November 2010

Abstract In this paper we study very large-scale neighborhoods for the minimum total weighted completion time problem on parallel machines, which is known to be strongly $\mathcal{NP}$-hard. We develop two different ideas leading to very large-scale neighborhoods in which the best improving neighbor can be determined by calculating a weighted matching. The first neighborhood is introduced in a general fashion using combined operations of a basic neighborhood. Several examples for basic neighborhoods are given. The second approach is based on a partitioning of the job sets on the machines and a reassignment of them. In a computational study we evaluate the possibilities and the limitations of the presented very large-scale neighborhoods.

Keywords Scheduling · Parallel machines · Total weighted completion time · Very large-scale neighborhoods · Local search

1 Introduction

Many optimization problems from practice are computationally intractable and it simply cost too much time to solve them to optimality. Hence, there is need for practical approaches to solve such problems. One class of such practical approaches contains heuristic (approximation) algorithms that are able to find satisfying solutions within a reasonable amount of computational time. In the literature concerning

T. Brueggemann was supported by the Netherlands Organization for Scientific Research (NWO) grant 613.000.225 (Local Search with Exponential Neighborhoods).
J.L. Hurink was supported by BSIK grant 03018 (BRICKS: Basic Research in Informatics for Creating the Knowledge Society).

T. Brueggemann · J.L. Hurink (✉)
Department of Applied Mathematics, University of Twente, P.O. Box 217, 7500 AE Enschede, The Netherlands
e-mail: j.l.hurink@utwente.nl

heuristic algorithms two important subclasses can be distinguished. The first subclass of heuristic algorithms consists of constructive algorithms. These algorithms build solutions by assigning values to one or more decision variables at a time. The second subclass contains improvement algorithms, that start with a feasible solution or a set of feasible solutions and iteratively try to advance to better solutions (e.g. local search, population based or hybrid approaches). In the latter class, local search algorithms (also called neighborhood search algorithms) play an important role.

The basis concept of a local search heuristic is to start with some initial solution and iteratively replaces the current solution by some solution in a neighborhood of this solution. Thus, for a local search approach, a method for calculating an initial solution, a neighborhood structure on the set of solutions and a method to select a solution from the neighborhood of a given solution are needed.

The neighborhood structure has an important influence on the efficiency of local search. It determines the navigation through the solution space and it's size affects the time it takes to find a best neighbor within the neighborhood. Therefore, one may expect that the size of the neighborhood has an influence on the quality of the final solution, since a larger neighborhood covers a larger number of solutions and of course, affects the running time to find a best neighbor. This indicates that a compromise between quality and running time has to be found.

In the last years, very large-scale neighborhoods get more and more attention. These very large-scale neighborhoods mostly contain an exponential number of solutions but allow a polynomial exploration. A nice survey about very large-scale neighborhood techniques is given by Ahuja et al. (2002). They categorize very large-scale neighborhoods into three not necessarily distinct classes. Their first category of neighborhood search algorithms consists of variable-depth methods. These algorithms partially exploit exponential-sized neighborhoods using heuristics. The second category consists of network flow based improvement algorithms. These methods use network flow techniques to identify improving neighbors. Finally, their third category consists of neighborhoods that are defined by restricting the set of solution of the considered problem in such a way that a subproblem is achieved which can be solved in polynomial time.

In the literature, two main directions concerning large-scale neighborhoods can be found. On the one hand, more theoretic oriented papers exist, which consider the computational time to find the best solution in a neighborhood versus the size of the neighborhood and are interested in neighborhoods of maximal size which can be explored within a certain time complexity; e.g. linear or quadratic (see e.g. Deineko and Woeginger 2000). On the other hand, more application oriented papers investigate how large-scale neighborhoods can be used to build successful heuristics for optimization problems (see e.g. Potts and van der Velde (1995) for the traveling salesman problem, Congram et al. (2002) for the single machine total weighted tardiness problem, or Agarwal et al. (2007) for the problem considered in this paper). Mostly, the success of the large-scale neighborhoods comes in combination with a sophisticated search method. This leaves open the question whether the sophisticated search method or the used large-scale neighborhood is the main reason beyond the success of the approaches.

In this work we give some more insight in the development and use of very large-scale neighborhoods. It is not the aim to design an overall best heuristic for the con-

sidered problem, but to compare two different approaches for receiving efficiently searchable very large-scale neighborhoods. We do this for the problem of scheduling independent jobs on parallel machines minimizing the weighted total completion time (using the notation given by Graham et al. (1979) this problem is denoted by $P \parallel \sum w_j C_j$). The developed neighborhoods consist mainly of matchings in an improvement graph. Until now, neighborhoods based on matchings are mostly used for approximating the traveling salesman problem and some vehicle routing problems, see Ahuja et al. (2002). According to the work of Ahuja et al. (2002), the matching based very large-scale neighborhoods presented in this work belong to the second category of very large-scale neighborhoods. The second approach presented in this paper can be put into the third category, since it is based on solving a restricted version of the considered scheduling problem. In a computational study, we investigate the efficiency of these two approaches and try to identify what makes a very large-scale neighborhood efficient.

For the problem $P \parallel \sum w_j C_j$ Belouadah and Potts (1994) develop a branch and bound algorithm. For receiving lower bounds, they use Lagrangian relaxation on the constraint, that at most m jobs may be processed during any unit time interval. Additionally, for the branching steps they use dominance rules from Elmaghraby and Park (1974). The presented algorithm is capable of solving instances of up to 30 jobs and 8 machines on a CDC 7600 computer in about 60 seconds. Due to the nature of the algorithm, computational time heavily increases with the number of the machines.

A different approach is used by van den Akker et al. (1999). They formulate the problem as a set-covering problem with an exponential number of binary variables. The idea is to assign to every machine exactly one set of jobs. There are 2^n different sets of jobs for the machines and hence, this gives a huge amount of columns for the integer linear program and its relaxation of assigning at most one set of jobs to a machine. Van den Akker et al. solve the relaxed linear program by column generation where the pricing algorithm for determining entering columns runs in pseudo-polynomial time, i.e. in $\mathcal{O}(n \sum_{j=1}^{n} p_j)$ time and space. Computational testing indicates, that the algorithm is capable of solving instances from Belouadah and Potts (1994) in one fourth of the time that their method needs. Moreover, because the algorithm performs better with increasing m, the column generation approach is able to solve instances of size up to $n = 100$ jobs and $m = n/10$ machines in at most an hour on a HP 9000/710 computer, which is about twice as fast as the CDC 7600. Chen and Powell (1999) give a further approach based on column generation.

Skutella and Woeginger (2000) present a polynomial time approximation scheme (PTAS). They use transformations that simplify an instance without dramatically increasing the objective value in order to show that there exists for every $\varepsilon > 0$ a polynomial time algorithm that computes a solution with at most a factor of $1 + \varepsilon$ away from the optimal solution. Sahni (1976) develop an FPTAS (fully PTAS) for the problem $Pm \parallel \sum w_j C_j$ with fixed m, which runs in time bounded by a polynomial in the input size and $1/\varepsilon$.

Barnes and Laguna (1993) introduce a tabu search algorithm for this problem. They use a combined neighborhood of job-insertions (we call such job-insertion a *move*) and swaps, i.e. they work on assignments of jobs to machines and change the assignment of one job or exchange the assignments of two jobs. Their computational

experiments point out that this method is rather successful in delivering near-optimal solutions. Agarwal et al. (2007) develop a very large-scale neighborhood based on moving jobs from one machine to another. In principle, they allow sequences of moves with decreasing job priorities. If in a first part of such a sequence a job j has been moved to a different machine and is inserted there directly before a job k, in the remaining part of the sequence only moves of jobs with lower priority than k are allowed. Agarwal et al. (2007) introduce an improvement graph for this neighborhood, which is searched heuristically and hence, forms a variable depth search algorithm. They run computational tests using this neighborhood in local search frameworks as iterative improvement, tabu search and iterated local search. They conclude that an iterated local search heuristic with several runs using randomly generated initial solutions delivers the best results regarding running time and solution quality.

The outline of the paper is as follows. In Sect. 2 we give a brief description of the problem and introduce important notations. In Sect. 3 we present a first approach to obtain very large-scale neighborhoods. This approach uses the idea of combining independent moves. Next, in Sect. 4 we introduce a second method to derive very large-scale neighborhoods. Here, we use a restricted version of the considered problem, which enables us to determine a best improving neighbor in polynomial time. Afterwards, in Sect. 5 we give computational results for these neighborhoods and discuss the possibilities and limitations of the concept. Finally, we give some concluding remarks.

2 Description of the problem

We consider the problem of scheduling a set $J = \{1, \ldots, n\}$ of jobs with processing times p_j and weights w_j for $j \in J$ on m identical parallel machines without preemption. The goal is to find a solution minimizing the sum of weighted completion times.

A solution of the problem consists of an *assignment* $A : \{1, \ldots, n\} \to \{1, \ldots, m\}$ of the jobs to the machines and a vector S of starting times $(S_1, \ldots, S_n)$ of the jobs. The starting times S are called a *feasible schedule* for the given assignment A, if and only if for all jobs that are processed on the same machine no two jobs overlap, i.e. if

$$\text{either} \quad S_j \geq S_i + p_i \quad \text{or} \quad S_i \geq S_j + p_j$$

for all pairs $i, j = 1, \ldots, n$ with $A(i) = A(j)$ and $i \neq j$. We denote the vector of completion times for the corresponding feasible schedule S by C, i.e. $C_j := S_j + p_j$ for $j = 1, \ldots, n$.

We are now interested in an assignment A of jobs to machines and a feasible schedule S, such that the objective function

$$f(S) := \sum_{j=1}^{n} w_j C_j \tag{1}$$

is minimized. This problem is strongly $\mathcal{NP}$-hard, as described in Lenstra et al. (1977) and Garey and Johnson (1979). The proof can be done by a reduction from

3-Partition to instances with $w_j = p_j$ for all $j = 1, \ldots, n$, showing that already $P|w_j = p_j|\sum w_j C_j$ is strongly $\mathcal{NP}$-hard. Furthermore, for two machines (and also for any fixed number of machines), the problem $P2 \parallel \sum w_j C_j$ is also $\mathcal{NP}$-hard, but only in the ordinary sense.

In the case of having one machine ($m = 1$), the resulting problem $1 \parallel \sum w_j C_j$ is solvable in polynomial time by sorting the jobs according to Smith's rule, i.e. $\frac{w_1}{p_1} \geq \frac{w_2}{p_2} \geq \cdots \geq \frac{w_n}{p_n}$ (Smith 1956). In the case of more than one machine ($m > 1$), jobs scheduled on different machines are not influencing each other. Moreover, the objective function Eq. 1 splits up in m separate parts for the m machines. Thus, if an assignment A of jobs to machines is given, the problem decomposes into m independent single machine problems, and an optimal schedule S respecting this assignment can easily be determined: order all jobs processed on the same machine according to Smith's rule. Summarizing, the calculation of the optimal schedule belonging to a given assignment A and the corresponding objective value $f(A)$ can be done in $\mathcal{O}(n \log n)$ and in $\mathcal{O}(n)$ if already an ordering of the jobs according to Smith's rule is given.

From now on we assume w.l.o.g. that the jobs are ordered according to Smith's rule. Additionally, for two jobs j and k we say that job j has a *higher priority* than job k, if $j < k$. Hence, jobs scheduled on the same machine are always sorted by their priority. Furthermore, for a given assignment A, we denote with $M_i := A^{-1}(i)$ the set of jobs processed by machine i. Let $n_i := |M_i|$ be the number of jobs assigned to machine i. Using the machine sets M_i, the schedule S corresponding to a given assignment A can be determined by

$$S_j := \sum_{\substack{k \in M_i \\ k < j}} p_k \quad \text{for all } j \in M_i \text{ and } i = 1, \ldots, m.$$

Based on the above considerations, we represent solutions of the problem $P \parallel \sum w_j C_j$ by assignments A and we denote by $f(A)$ the objective value of the schedule belonging to assignment A.

In the following we use partial sums of processing times and weights for the machines. For a given assignment A, machine i and job j we denote with L_{ij} the sum of processing times of all jobs assigned to machine i with higher or equal priority compared to job j. Additionally, with W_{ij} we denote the sum of weights of all jobs assigned to machine i with lower priority compared to job j. More formally:

$$\begin{aligned} L_{ij} &:= \sum_{\substack{k \in M_i \\ k \leq j}} p_k \quad \text{for all } j = 1, \ldots, n \text{ and } i = 1, \ldots, m, \\ W_{ij} &:= \sum_{\substack{k \in M_i \\ k > j}} w_k \quad \text{for all } j = 1, \ldots, n \text{ and } i = 1, \ldots, m. \end{aligned} \tag{2}$$

Observe, that $L_{A(j),j} = C_j$ for all jobs j. For a fixed machine i the values L_{ij} and W_{ij} for all $j = 1, \ldots, n$ can be determined in $\mathcal{O}(n)$ and hence, all values can be calculated in $\mathcal{O}(nm)$. If we only need the values for a fixed pair of machines i_1 and i_2, we can calculate L_{ij} and W_{ij} for $i = i_1, i_2$ and $j \in M_{i_1} \cup M_{i_2}$ in $\mathcal{O}(n_{i_1} + n_{i_2})$.

3 Neighborhoods by combining independent moves

In the following we introduce a first approach leading to very large-scale neighborhoods of up to exponential size for the problem $P \parallel \sum w_j C_j$. The basic principles presented are not only restricted to neighborhood search for the problem $P \parallel \sum w_j C_j$ but are also adoptable for similar problems, which have the property, that the schedules on the different machines are independent; e.g. $Q \parallel \sum w_j C_j$, $R \parallel \sum w_j C_j$.

The main idea behind the presented approach to build very large-scale neighborhoods is to start with a rather simple basic neighborhood $\mathcal{N}_1$ and to build a new very large-scale neighborhood $\mathcal{N}_2$ by allowing combinations of operators from the neighborhood $\mathcal{N}_1$. To be able to evaluate the neighborhood $\mathcal{N}_2$, only combinations of operators which are somehow independent are allowed. This concept of combining independent moves has already been used for other optimization problems. Potts and van der Velde (1995) used the concept for the traveling salesman problem, Congram et al. (2002) applied it to the single machine total weighted tardiness problem, and Hurink (1999) to a single machine batching problem. Furthermore, in Ergun et al. (2006) a quite general concept of the use of compound independent moves for routing problems with side constraints is presented. In all these approaches searching the large-scale neighborhoods reduced to a shortest path problem or could be solved using dynamic programming. As we will see, in our case searching the large-scale neighborhood leads to a matching problem.

The presented approach to get a very large-scale neighborhood for the parallel machine scheduling problem is general in the sense that it can be applied to all basic neighborhoods $\mathcal{N}_1$ which fulfill some stated properties. These properties and the proposed construction of the neighborhood $\mathcal{N}_2$ are given in the following.

The basic neighborhood $\mathcal{N}_1$ has to consist of operators $\mathrm{op}(i_1, i_2)$ which operate on pairs (i_1, i_2) of different machines (i.e. $i_1 \neq i_2$). Hence, $\mathcal{N}_1$ contains up to $\frac{1}{2}m(m-1)$ neighbors. The operators have to fulfill the following properties:

- the change resulting from the application of an operation $\mathrm{op}(i_1, i_2)$ for a fixed pair $i_1 \neq i_2$ is only dependent on the given schedules of the two machines i_1 and i_2, and has only effects on the machines i_1 and i_2,
- the operator is symmetric, i.e. the assignments $A' := \mathrm{op}(i_1, i_2)(A)$ and $A'' := \mathrm{op}(i_2, i_1)(A)$ are equal.

Next, two operators $\mathrm{op}(i_1, i_2)$ and $\mathrm{op}(i_3, i_4)$ are defined to be *independent* if $i_j \neq i_k$ for $j, k \in \{1, 2, 3, 4\}$; $j \neq k$. Thus, if we apply a set of pairwise independent operators $\mathrm{op}(s_1, t_1), \ldots, \mathrm{op}(s_k, t_k)$ to a solution A the resulting change in the objective value is given by

$$f(A) - f(\mathrm{op}(s_k, t_k)(\ldots(\mathrm{op}(s_1, t_1)(A))\ldots)) = \sum_{l=1}^{k} \delta_{s_l, t_l}(A)$$

whereby $\delta_{i_1, i_2}(A) := f(A) - f(\mathrm{op}(i_1, i_2)(A))$ denotes the improvement of the objective value resulting from the application of $\mathrm{op}(i_1, i_2)$ to assignment A.

All basic neighborhoods $\mathcal{N}_1$ fulfilling the above properties can be used as the base to design an (in m) exponential neighborhood $\mathcal{N}_2$. The neighborhood $\mathcal{N}_2$ of an

assignment A consists of all assignments that result from applying a set of pairwise independent operators to the assignment A. In the following we describe how a best neighbor in this neighborhood can be obtained.

The operators of the neighborhood $\mathcal{N}_2$ correspond to matchings in a weighted graph $G(A) = (V, E, c(A))$, where

- the vertex set V contains a vertex for each machine,
- an edge $e = \{i_1, i_2\} \in E$ with $i_1 \neq i_2$ represents the operator $\text{op}(i_1, i_2)$,
- an edge $e = \{i_1, i_2\}$ gets a weight $c(A)_{i_1,i_2} = \delta_{i_1,i_2}(A)$.

The weights of the graph can be calculated by applying the operator $\text{op}(i_1, i_2)$ for every pair of machines i_1, i_2 with $i_1 < i_2$ to assignment A. Thus, the complexity of building up the graph is $\frac{1}{2}m(m-1)$ times the complexity of evaluating the effects of a single operator $\text{op}(i_1, i_2)$. Each matching in this graph corresponds to a set of pairwise independent operators of $\mathcal{N}_1$ and, thus, to an operator of $\mathcal{N}_2$ and vice versa. Furthermore, the weight $w(\mathcal{M})$ of a matching $\mathcal{M}$ is given by the sum of the weights of all edges present in the matching, i.e.

$$w(\mathcal{M}) := \sum_{\{i_1,i_2\}\in\mathcal{M}} c(A)_{i_1,i_2} = \sum_{\{i_1,i_2\}\in\mathcal{M}} \delta_{i_1,i_2}(A).$$

We can determine the best neighbor of an assignment A in neighborhood $\mathcal{N}_2$ by calculating a maximum weight matching $\mathcal{M}$ in the graph $G(A)$. Observe, that the structure of the graph $G(A)$ is independent of the considered assignment A but the weights heavily depend on it. Determining a maximum weight matching in a general graph with $|V|$ vertices and $|E|$ edges can be done in different ways. There exists a $\mathcal{O}(|V|^3)$-algorithm from Gabow (1973) and Lawler (1976) extending the work of Edmonds (1965). Using this algorithm, the best neighbor in $\mathcal{N}_2$ can be determined in $\mathcal{O}(m^3)$.

Summarizing, the neighborhood $\mathcal{N}_2$ consists of an (in m) exponential number of neighbors, whereby each neighbor represents the results achieved from applying a set of independent operators of the basic neighborhood $\mathcal{N}_1$, and is efficiently searchable. In the following we introduce some examples for the basic neighborhood $\mathcal{N}_1$, which fulfill the stated properties for the neighborhood $\mathcal{N}_2$.

3.1 Move neighborhood

A first example of a basic neighborhood $\mathcal{N}_1$ is built up using move operators $\text{op}^{\text{move}}(i_1, i_2)$, which consider machines i_1 and i_2 of the given assignment A and move exactly one job between these two machines in such a way that the change in the objective value is best possible. Obviously, these operators fulfill the stated property for the basic neighborhood $\mathcal{N}_1$. It remains to describe how these operators $\text{op}^{\text{move}}(i_1, i_2)$ can be evaluated efficiently.

If $\text{move}(j, i)$ denotes an operator that reassigns job j to machine i, an operator $\text{op}^{\text{move}}(i_1, i_2)$ represents a best possible move in a neighborhood given by the operators $\text{move}(j, i)$, $i \in \{i_1, i_2\}$; $j \in (M_{i_1} \cup M_{i_2}) \setminus M_i$, i.e. the best possible move in a neighborhood of size $n_{i_1} + n_{i_2}$. The overall change in the objective value

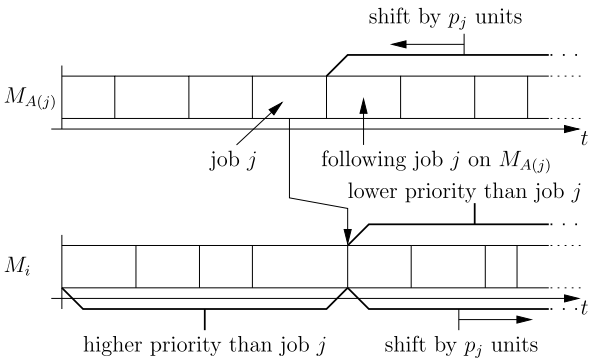

Fig. 1 Illustration of move$(j, i)(A)$

$\delta^A_{\text{move}(j,i)} := f(A) - f(\text{move}(j,i)(A))$ resulting from an application of move(j, i) to assignment A is given by

$$\delta^A_{\text{move}(j,i)} = p_j(W_{A(j),j} - W_{ij}) + w_j(L_{A(j),j} - L_{ij} - p_j), \tag{3}$$

(see Fig. 1 for an illustration). Thus, if the corresponding values for W and L from Eq. 2 are known, $\delta^A_{\text{move}(j,i)}$ can be calculated in $\mathcal{O}(1)$.

For the operator $\text{op}^{\text{move}}(i_1, i_2)$ we now have to find the best move of a job between the two machines i_1 and i_2. This can be achieved by evaluating all moves move(j, i_2) for jobs j with $j \in M_{i_1}$ and all moves move(j, i_1) for jobs j with $j \in M_{i_2}$. Since in a preprocessing, the relevant values for W and L can be calculated in $\mathcal{O}(n_{i_1} + n_{i_2})$, the overall complexity to evaluate the operator $\text{op}^{\text{move}}(i_1, i_2)$ is $\mathcal{O}(n_{i_1} + n_{i_2})$. For the neighborhood $\mathcal{N}_2$ we have to evaluate all operators $\text{op}^{\text{move}}(i_1, i_2)$ with $i_1 < i_2$ to build up the graph $G(A)$. This can be realized in $\mathcal{O}(nm)$.

The neighborhood $\mathcal{N}_2$ contains an exponential number of neighbors (exponential in m), where each neighbor again dominates a certain number of neighbors w.r.t. the neighborhood defined by move(j, i) operators. More precisely, a matching $\mathcal{M}$ containing the edges $\{s_1, t_1\}, \ldots, \{s_k, t_k\}$ in $G(A)$ leads to a solution which is the best in a neighborhood of size $\prod_{l=1}^{k}(n_{s_l} + n_{t_l})$. In contrast to this, neighborhood $\mathcal{N}_1$ (i.e. move a job between a pair of machines i_1 and i_2) contains only $\frac{1}{2}m(m-1)$ neighbors, where each neighbor represents the best solution in a neighborhood of size $n_{i_1} + n_{i_2}$.

3.2 Combining several moves

Another basic neighborhood $\mathcal{N}_1$ can be defined by not only moving one job from a machine to another, but swapping two jobs between a pair of machines, or as a generalization, by allowing a fixed amount of moves between a pair of machines in one step. In the next part we introduce the foundations to evaluate such operators efficiently. A general method is introduced to calculate the change of the objective value resulting from operators exchanging several jobs between machines i_1 and i_2, where i_1 and i_2 are fixed.

Let A_1 be an assignment that results from assignment A by applying several move operations. We want to use the change $\delta^A_{\text{move}(j,i)}$ of the objective value resulting from applying move(j, i) to assignment A, to calculate the objective change resulting from

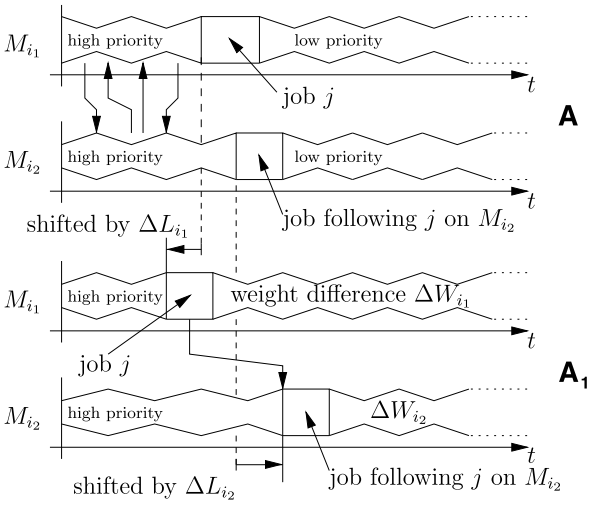

Fig. 2 Emanating from assignment A to A_1

applying move(j, i) to assignment A_1. Hereby we focus on the case that job j has not changed its machine; i.e. we consider the case that job j is processed by machine $i_1 := A(j)$ in assignment A and A_1. Simple calculations yield that the effect $\delta^{A_1}_{\text{move}(j,i_2)}$ of applying the operator move(j, i_2) to A_1 is given by

$$\delta^{A_1}_{\text{move}(j,i_2)} = \delta^{A}_{\text{move}(j,i_2)} + w_j(\Delta L_{i_1} - \Delta L_{i_2}) + p_j(\Delta W_{i_1} - \Delta W_{i_2}), \tag{4}$$

where $\Delta L_{i_1} := L^{A_1}_{i_1,j} - L^{A}_{i_1,j}$, $\Delta W_{i_1} := W^{A_1}_{i_1,j} - W^{A}_{i_1,j}$, $\Delta L_{i_2} := L^{A_1}_{i_2,j} - L^{A}_{i_2,j}$, and $\Delta W_{i_2} := W^{A_1}_{i_2,j} - W^{A}_{i_2,j}$. For an illustration of the situation see Fig. 2, where the assignments A and A_1 are shown as well as job j and the effects influencing the change in the objective value. By "high priority" we denote all jobs with higher priority than job j and by "low priority" we denote all jobs with lower priority.

Equation 4 is the base of combining several moves. In the next two subsections, we give examples of possible combinations of moves. The combined moves then again can be used to build a very large-scale neighborhood that can be explored via matchings.

3.2.1 Swap neighborhood

The second example of a basic neighborhood $\mathcal{N}_1$ uses swap operators $\text{op}^{\text{swap}}(i_1, i_2)$. Hereby, $\text{op}^{\text{swap}}(i_1, i_2)$ is a combination of two move operators, which consider the machines i_1 and i_2 of the given assignment A and move exactly one job from machine i_1 to i_2 and one job from machine i_2 to i_1. This is done in such a way that the change in the objective value is best possible. Obviously, these operators again fulfill the stated property for the basic neighborhood $\mathcal{N}_1$. In the following we describe how the operators $\text{op}^{\text{swap}}(i_1, i_2)$ can be evaluated efficiently by first studying the effects of a single swap of jobs swap(j, k) and, then, showing how on base of these results the effect of an operator $\text{op}^{\text{swap}}(i_1, i_2)$ can be calculated.

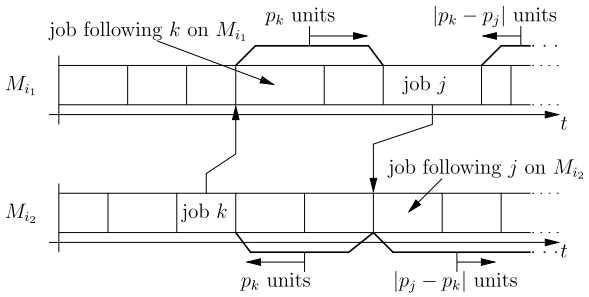

Fig. 3 Illustration of $\text{swap}(j,k)(A)$ for $k < j$

Let i_1 and i_2 be a pair of different machines, job j be a job assigned to machine i_1, job k be a job assigned to machine i_2, and assume w.l.o.g. that job k is of higher priority than job j. Then, a swap of jobs $\text{swap}(j,k)$ consists of two moves $\text{move}(j,i_2)$ and $\text{move}(k,i_1)$, i.e. $\text{swap}(j,k)(A) = \text{move}(j,i_2)(\text{move}(k,i_1)(A))$ (see Fig. 3 for an illustration of a swap of jobs).

Applying Eq. 4 with $A_1 := \text{move}(k,i_1)(A)$ leads to the following overall change in the objective value $\delta^A_{\text{swap}(j,k)} := f(A) - f(\text{swap}(j,k)(A))$ resulting from an application of $\text{swap}(j,k)$ to assignment A:

$$\delta^A_{\text{swap}(j,k)} := \delta^A_{\text{move}(j,i_2)} + \delta^A_{\text{move}(k,i_1)} + 2w_j p_k. \tag{5}$$

Thus, if the values $\delta^A_{\text{move}(j,i_2)}$ and $\delta^A_{\text{move}(k,i_1)}$ are known in advance, $\delta^A_{\text{swap}(j,k)}$ can be calculated in $\mathcal{O}(1)$.

For the operator $\text{op}^{\text{swap}}(i_1,i_2)$ we now have to calculate the best exchange of jobs from machine i_1 and i_2. Since in a preprocessing the values $\delta^A_{\text{move}(j,i_2)}$ for $j \in M_{i_1}$ and $\delta^A_{\text{move}(k,i_1)}$ for $k \in M_{i_2}$ can be obtained in $\mathcal{O}(n)$, the overall complexity of $\text{op}^{\text{swap}}(i_1,i_2)$ is $\mathcal{O}(n^2)$. For the neighborhood $\mathcal{N}_2$ we have to evaluate all operators $\text{op}^{\text{swap}}(i_1,i_2)$ with $i_1 < i_2$ to build up the graph $G(A)$. This can be realized in an overall complexity of $\mathcal{O}(n^2)$.

Again, the neighborhood $\mathcal{N}_2$ contains an exponential number of neighbors, where a neighbor corresponding to a matching $\mathcal{M}$ containing the edges $(s_1,t_1),\ldots,(s_k,t_k)$ in $G(A)$ leads to a solution which is the best in a neighborhood of size $\prod_{l=1}^{k}(n_{s_l}n_{t_l})$. In contrast to this, neighborhood $\mathcal{N}_1$ (i.e. swap a pair of jobs between a pair of machines i_1 and i_2) contains only $\frac{1}{2}m(m-1)$ neighbors, where each neighbor represents a best solution in a neighborhood of size $n_{i_1}n_{i_2}$.

3.2.2 $\overline{k}$-move neighborhood

The ideas of move and swap can be generalized to a basic neighborhood $\mathcal{N}_1$ resulting from $\overline{k}$-move operators $\text{op}^{\overline{k}\text{-move}}(i_1,i_2)$. Hereby, $\text{op}^{\overline{k}\text{-move}}(i_1,i_2)$ moves up to k jobs between machines i_1 and i_2. The moves are chosen such that the change in the objective value is best possible. These operators fulfill the stated properties for the basic neighborhood $\mathcal{N}_1$. It remains to describe how these operators $\text{op}^{\overline{k}\text{-move}}(i_1,i_2)$ can be realized efficiently.

An operator $\text{op}^{\overline{k}\text{-move}}(i_1,i_2)$ represents a best possible move of up to k jobs between machines i_1 and i_2. In the following we examine a single operator that moves

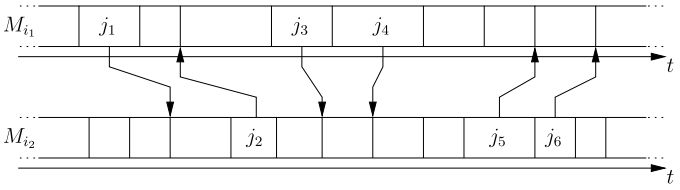

Fig. 4 Illustration of 6-move$(j_1, \ldots, j_6)$

exactly k jobs between machines i_1 and i_2. To simplify notation, we denote by $\overline{A}(j)$ the machine that job j is not assigned to, i.e.

$$\overline{A}(j) = \begin{cases} i_1 & \text{if} A(j) = i_2, \\ i_2 & \text{if} A(j) = i_1. \end{cases}$$

This makes it possible to write move$(j)(A)$ instead of move$(j, \overline{A}(j))(A)$. For a fixed k we now examine a combination of moves

$$k\text{-move}(j_1, \ldots, j_k)(A) := \text{move}(j_k) \circ \text{move}(j_{k-1}) \circ \cdots \circ \text{move}(j_1)(A),$$

where $j_1 < j_2 < \cdots < j_k$, i.e. job j_l has a higher priority than job j_{l+1} for $l = 1, \ldots, k-1$ (see Fig. 4 for an illustration).

At first we consider the assignment $A_2 := \text{move}(j_2) \circ \text{move}(j_1)(A)$. Due to Eq. 4 and a similar argumentation as used in Sect. 3.2.1 for evaluating a swap, the difference of the objective values of assignments A and A_2 can be calculated by

$$f(A) - f(A_2) = \delta^A_{\text{move}(j_1, \overline{A}(j_1))} + \delta^A_{\text{move}(j_2, \overline{A}(j_2))} + 2w_{j_2} p_{j_1} \Delta_{j_2, j_1}, \tag{6}$$

where $\Delta_{j_s, j_t} = 1$ if job j_t and j_s are assigned to different machines in A and $\Delta_{j_s, j_t} = -1$ in the other case. Applying this in an iterative manner, leads to the following expression for the change of the objective value resulting from k-move$(j_1, \ldots, j_k)(A)$:

$$\delta^A_{k\text{-move}(j_1, \ldots, j_k)} = \sum_{s=1}^{k} \delta^A_{\text{move}(j_s, \overline{A}(j_s))} + \sum_{s=1}^{k} 2w_{j_s} \sum_{t=1}^{s-1} \Delta_{j_s, j_t} p_{j_t}. \tag{7}$$

The calculation of value $\delta^A_{k\text{-move}(j_1, \ldots, j_k)}$ with Eq. 7 needs in the worst-case a running time of $\mathcal{O}(k^2)$, if the values $\delta^A_{\text{move}(j_s, \overline{A}(j_s))}$ are known in advance.

For the operator $\text{op}^{\overline{k}\text{-move}}(i_1, i_2)$ we now have to calculate the best move of up to k jobs between machines i_1 and i_2. To do so, denote with $M_{i_1} \cup M_{i_2}$ the set of jobs processed by machine i_1 or i_2. For every l with $1 \leq l \leq k$ we have to calculate for every possible subset $\{j_1, \ldots, j_l\} \subseteq M_{i_1} \cup M_{i_2}$ of cardinality l the value $\delta^A_{l\text{-move}(j_1, \ldots, j_l)}$ to determine the best move of up to k jobs between machines i_1 and i_2. Since in a preprocessing the values $\delta^A_{\text{move}(j_s, \overline{A}(j_s))}$ can be obtained in $\mathcal{O}(n)$, the overall complexity of $\text{op}^{\overline{k}\text{-move}}(i_1, i_2)$ is $\mathcal{O}(k^2 n^k)$. For the neighborhood $\mathcal{N}_2$ we have to evaluate all operators $\text{op}^{\overline{k}\text{-move}}(i_1, i_2)$ with $i_1 < i_2$ to build up the graph $G(A)$. This can be realized in

$\mathcal{O}(k^2m^2n^k)$. Thus, for constant values of k the neighborhood $\mathcal{N}_2$ can be evaluated in polynomial time. Furthermore, the operator $\text{op}^{\overline{k}\text{-move}}(i_1, i_2)$ retrieves the best solution out of $\sum_{l=1}^{k} \binom{n_{i_1}+n_{i_2}}{l}$ solutions contained in the neighborhood consisting of l-move operators with $1 \leq l \leq k$.

As before, the neighborhood $\mathcal{N}_2$ contains an exponential number of neighbors, and a neighbored solution achieved by a matching $\mathcal{M}$ containing the edges $(s_1, t_1), \ldots, (s_\mu, t_\mu)$ in $G(A)$ leads to a solution which is the best in a neighborhood of size

$$\prod_{\nu=1}^{\mu}\left(\sum_{l=1}^{k}\binom{n_{s_\nu}+n_{t_\nu}}{l}\right).$$

In contrast to this, the neighborhood $\mathcal{N}_1$ consisting of $\text{op}^{\overline{k}\text{-move}}(i_1, i_2)$ for $1 \leq i_1, i_2 \leq m$ and $i_1 \neq i_2$ contains only $\frac{1}{2}m(m-1)$ neighbors, where each neighbor represents the best solution in a neighborhood of size $\sum_{l=1}^{k} \binom{n_{i_1}+n_{i_2}}{l}$.

4 Split neighborhood

In this section we introduce the so-called split neighborhood. The best improving neighbor in this neighborhood can be determined by solving an assignment problem and due to the special structure this assignment can be determined via some sorting routine. Since in this section we need to consider machine sets M_i resulting from different assignments, we add a subscript to indicate the corresponding assignment; i.e. M_i^A is the machine set of machine i under assignment A.

Let q be a fixed, but arbitrarily chosen job. A *split* of jobs with *anchor* q for a given assignment A is a partition of all sets M_i^A into a so-called *left-part* M_{i1}^A and a *right-part* M_{i2}^A, such that

$$M_{i1}^A := \{j \in M_i^A : j \leq q\},$$
$$M_{i2}^A := \{j \in M_i^A : j > q\}.$$

A second assignment A' is a neighbor of A with respect to the anchor q, if the resulting sets $M_{i1}^{A'}$ and $M_{i2}^{A'}$ are equal to the sets M_{i1}^A and M_{i2}^A for $i = 1, \ldots, m$, i.e. if there exist two permutations π_l, π_r, such that $M_{\pi_l(i),1}^A = M_{i1}^{A'}$ and $M_{\pi_r(i),2}^A = M_{i2}^{A'}$. The neighborhood $\mathcal{N}_{\text{split}(q)}(A)$ consists of all neighboring assignments A' with respect to the anchor q. This means, that the neighborhood $\mathcal{N}_{\text{split}(q)}(A)$ contains all reassignments of the right-parts M_{i2} to the left-parts M_{i1}, and, thus, consist of up to $m!$ assignments.

In the following, we describe how the neighborhood $\mathcal{N}_{\text{split}(q)}(A)$ can be efficiently explored and a best neighboring assignment can be determined. For this, consider M_{i1} and M_{i2}, $i = 1, \ldots, m$, to be a split of jobs with anchor q for a given assignment A. We define the contribution $f(M_{il})$ of a part M_{il} for $l \in \{1, 2\}$ and $i \in \{1, \ldots, m\}$ to be

$$f(M_{il}) := \sum_{j \in M_{il}} w_j L_{ij}.$$

Using these values, the objective value $f(A)$ calculates as

$$f(A) = \sum_{i=1}^{m} \big(f(M_{i1}) + f(M_{i2})\big) + \sum_{i=1}^{m} L_{iq} W_{iq},$$

where $L_{iq} = \sum_{j \in M_{i1}} p_j$ and $W_{iq} = \sum_{j \in M_{i2}} w_j$.

For a neighboring assignment $A' \in \mathcal{N}_{\text{split}(q)}(A)$ given by two permutations $\pi_l, \pi_r \in \mathcal{S}_m$, such that $M^A_{\pi_l(i),1} = M^{A'}_{i1}$ and $M^A_{\pi_r(i),2} = M^{A'}_{i2}$; $i = 1, \ldots, m$, the objective value calculates as

$$\begin{aligned} f(A') &= \sum_{i=1}^{m} \big(f(M^{A'}_{i1}) + f(M^{A'}_{i2}) + L^{A'}_{iq} W^{A'}_{iq}\big) \\ &= \sum_{i=1}^{m} \big(f(M^{A}_{\pi_l(i),1}) + f(M^{A}_{\pi_r(i),2}) + L^{A}_{\pi_l(i),q} W^{A}_{\pi_r(i),q}\big) \\ &= \sum_{i=1}^{m} \big(f(M_{i1}) + f(M_{i2})\big) + \sum_{i=1}^{m} L^{A}_{\pi_l(i),q} W^{A}_{\pi_r(i),q}. \end{aligned}$$

Hence, the objective value of a neighboring assignment A' differs from A in the value of $\sum_{i=1}^{m} L^A_{\pi_l(i),q} W^A_{\pi_r(i),q}$. This value is determined by the choice, which right parts and left parts are assigned to the same machine in A'. As a consequence, the search for a best neighbor in $\mathcal{N}_{\text{split}(q)}(A)$ reduces to finding an assignment between the values $L_{1q}, \ldots, L_{mq}$ and the values $W_{1q}, \ldots, W_{mq}$ which minimizes the sum of the products of the assigned values. This assignment problem in a bipartite graph has a special product property of the costs; i.e. each vertex has assigned a non-negative value and the costs of assigning two vertices is equal to the product of the two values of the vertices. Based on this property, the assignment which results if we put the ith largest L-value and the $m + i$th largest W-value to machine i for $i = 1, \ldots, m$ turns out to be an optimal assignment (see for example the chapter on the assignment problem in Brucker (2004) for a proof). Thus, the best neighbor can be calculated in time $\mathcal{O}(m \log m)$ by sorting the L and W values.

5 Computational results

In this section, we report on computational experiments conducted to analyze the efficiency of the introduced neighborhoods. The efficiency of neighborhoods is determined by the achieved solution quality and the used computation time. Our aim is to evaluate the potential of the neighborhood and not on that of specific local search method. Therefore, we have chosen to use Iterative Improvement as local search method (i.e. in each iteration we move to an improving neighbor and stop if no such neighbor exists). For the same reason we start the search with different initial solutions. This shows which quality of local optimal solution is achieved dependent on the quality of the initial solution.

The problem instances were generated as described by van den Akker et al. (1999). They introduce three different types of instances, where the integer processing times and weights are uniformly drawn out of the following intervals:

- type (1) with $p_j \in [1, 10]$ and $w_j \in [10, 100]$,
- type (2) with $p_j \in [1, 100]$ and $w_j \in [1, 100]$,
- type (3) with $p_j \in [10, 20]$ and $w_j \in [10, 20]$.

Additionally, we consider another type Eq. 4, where the processing times are taken uniformly from the interval $[5, 15]$ or $[35, 45]$ and the weights are determined by choosing the weight to processing time ratio uniformly from the interval $[0.8, 1.2]$. Since we were not able to receive any data-sets or codes of the existing exact solution methods of Belouadah and Potts (1994) and van den Akker et al. (1999), we calculated for a set of smaller instances optimal solutions via some straight forward enumeration method. Additionally, we use a lower bounding method described by Eastman et al. (1964) (abbreviated with *LB*). The computational tests were conducted on a PC with Intel Pentium IV processor running at 2.4 GHz and the used methods were coded in ANSI-C.

For receiving initial solutions we use two different constructive heuristics. A first method is introduced by Eastman et al. (1964) and is called *LRF-heuristic* (*Largest Ratio First*). Kawaguchi and Kyan (1986) prove that schedules constructed by the LRF-heuristic do not exceed $(\sqrt{2}+1)/2 < 1.208$ times the optimal value. Computational testing from Baker and Merten (1973) and Barnes and Laguna (1993) indicate that the LRF-heuristic is rather successful in delivering near-optimal solutions. Additionally, a second heuristic goes back to an idea of Hoede (2006). Here, iteratively a non-scheduled job j is chosen for which

$$w_j \sum_{k \in N \setminus \{j\}, k<j} p_k + p_j \sum_{k \in N \setminus \{j\}, k>j} w_k$$

is maximal. This job j is inserted on the machine giving the least increase in the objective value. We denote this algorithm as *FF-heuristic*.

We denote with *k-MOVE* the iterative improvement procedure that advances in every iteration to the best solution in the neighborhood $\mathcal{N}_{k\text{-move}}$. With *MATCHING k-MOVE* we denote the corresponding iterative improvement algorithm using the matching based neighborhood $\mathcal{N}^M_{k\text{-move}}$. To determine the best improving neighbor in the neighborhood $\mathcal{N}^M_{k\text{-move}}$ we have to solve a maximum weighted matching problem for the improvement graph corresponding to a given assignment. We realized this by solving an integer linear program, which turned out to be fast enough for our purposes. Furthermore, by *SPLIT* we denote the iterative improvement procedure that calculates for every job $q = 1, \ldots, n$ the best improving neighbor of the current solution in $\mathcal{N}_{\text{split}(q)}$ and then advance to the best solution found until there can be made no improvement for any job q. At last, we use iterative improvement on the combined neighborhoods $\mathcal{N}_{\text{split}(q)}$ and $\mathcal{N}_{2\text{-move}}$. More precisely, we run SPLIT and afterwards advance to the best improving neighbor in $\mathcal{N}_{2\text{-move}}$. This process is repeated until no further improvement is possible. We denote this algorithm with *SPLIT-2MOVE*.

If it is not mentioned, we always start with an initial solution calculated by the LRF-heuristic. If we speak of objective values of a method we always mean the local

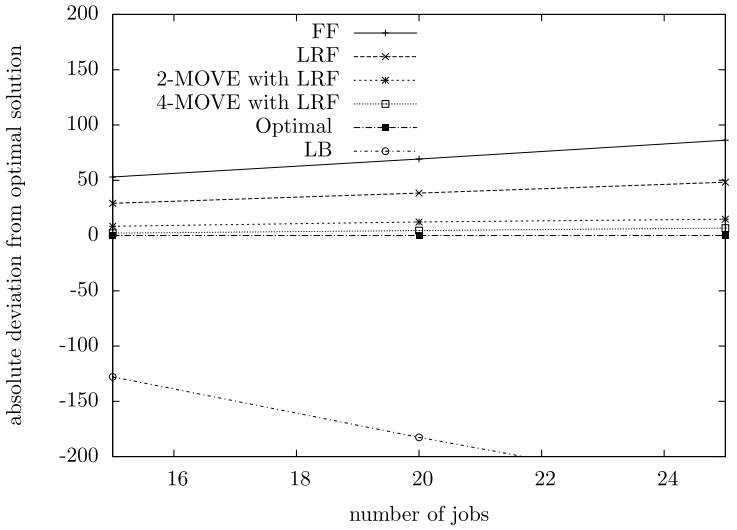

Fig. 5 Deviation from optimal solutions (type Eq. 1 instances and $\frac{n}{m} = 5$)

optimum obtained at the end of the algorithm. We tested the given neighborhoods with different initial solutions and instances with a job to machine ratio of $\frac{n}{m} = 10$ and $\frac{n}{m} = 25$. In order to receive significant results we average over 100 randomly generated instances for each measurement.

First, we give an indication of the overall quality of the tested neighborhoods and the lower bound by comparing them to optimal solutions. In Fig. 5 we present the results of the two heuristics FF and LRF, the values of k-MOVE using LRF as initial solution for $k \in \{2, 4\}$, the optimal value and the lower bound LB for instances with a job to machines ratio of $\frac{n}{m} = 5$ and $n \in \{15, 20, 25\}$. The values in this figure show the average absolute deviation from the optimal objective value (the relative deviations are $\leq$0.5% times the optimal value). One can see that especially the solutions resulting from k-MOVE with $k \in \{2, 4\}$ are close to the optimal solution, whereas the lower bounding scheme *LB* does not get close to the optimal values. In Fig. 5 results for instances of type Eq. 1 are given, but these results are representative for the other types.

The results of Fig. 5 indicate that the results of 4-MOVE using LRF as initial solution form a good indication of the optimal objective value. Therefore, in the following we always present deviations of achieved solutions from the 4-MOVE value using the LRF as initial solution. Furthermore, initial tests have shown that, as already indicated in Fig. 5, the LRF heuristic mostly performs better than the FF-heuristic (only for type Eq. 4 and $\frac{n}{m} = 10$ the FF-heuristic performs better than LRF).

In a next series of test, we investigate the performance of the different basic neighborhoods. In Fig. 6 the outcome for a job to machine ratio of $\frac{n}{m} = 25$ of k-MOVE for $k = 1, \ldots, 4$ and SPLIT are given. Again, type Eq. 1 is taken as a representative.

- With increasing k the values of k-MOVE get better for all types of instances except for type Eq. 3, where the situation looks a bit different (see Fig. 7). Here, the LRF-

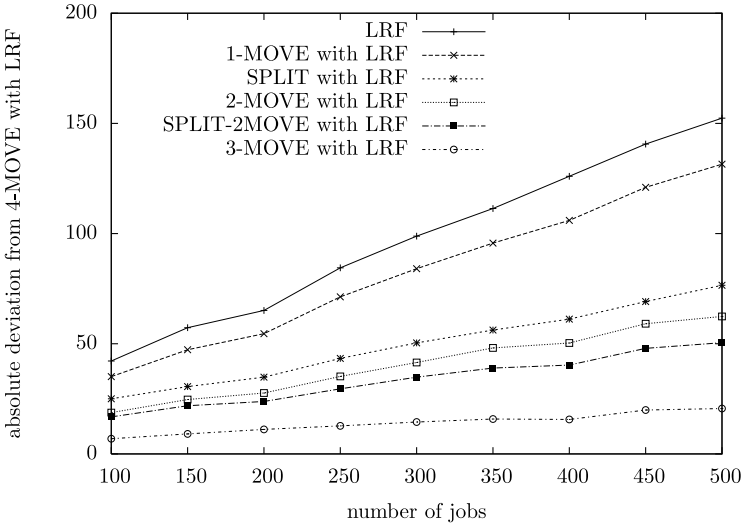

Fig. 6 Overall performance (type Eq. 1 instances and $\frac{n}{m} = 25$)

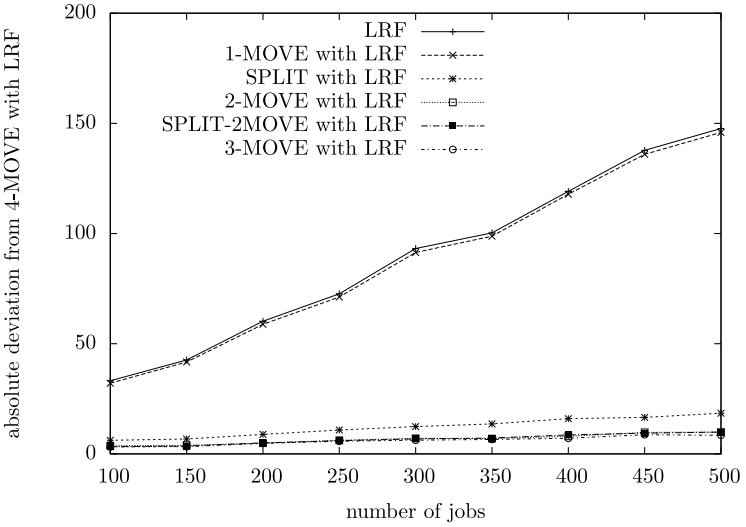

Fig. 7 Overall performance (type Eq. 3 instances and $\frac{n}{m} = 25$)

heuristic performs nearly the same compared to 1-MOVE. Additionally, 2-MOVE and 3-MOVE render comparable. Indeed, for instances of type Eq. 3 the jobs do not differ a lot and thus, an improvement mainly can be achieved by exchanging jobs rather than by moving. As a consequence, k-move with an odd k behaves almost the same as $(k-1)$-move.

- Mostly, the quality of local optima resulting from SPLIT is between 1-MOVE and 2-MOVE. Only for instances of type Eq. 2, the quality of SPLIT is comparable to 3-

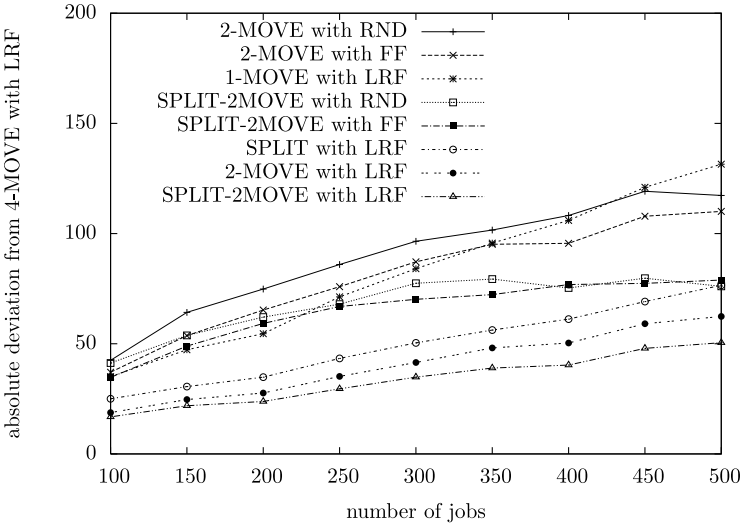

Fig. 8 k-MOVE and SPLIT using different initial solutions (type Eq. 1 instances and $\frac{n}{m} = 25$)

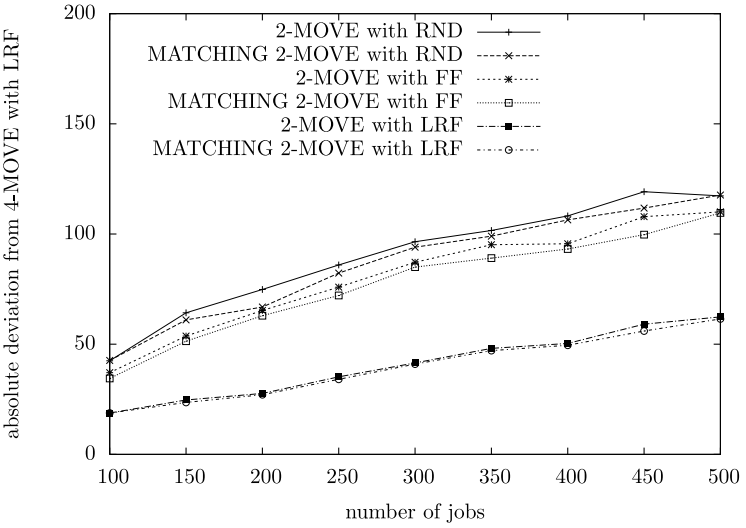

Fig. 9 Comparing k-MOVE and MATCHING k-MOVE (type Eq. 1 instances and $\frac{n}{m} = 25$)

MOVE (for smaller instances) and 4-MOVE (for larger instances). The combined approach SPLIT-2MOVE performs slightly better than 2-MOVE for all types.

To investigate how sensitive our methods are to the choice of initial solutions (see Fig. 8) we apply the basic neighborhoods using the two heuristics FF and LRF, but also use randomly generated initial solution (RND). First, note that using an initial solution of bad quality (RND), only seldom leads to solutions of a similar quality

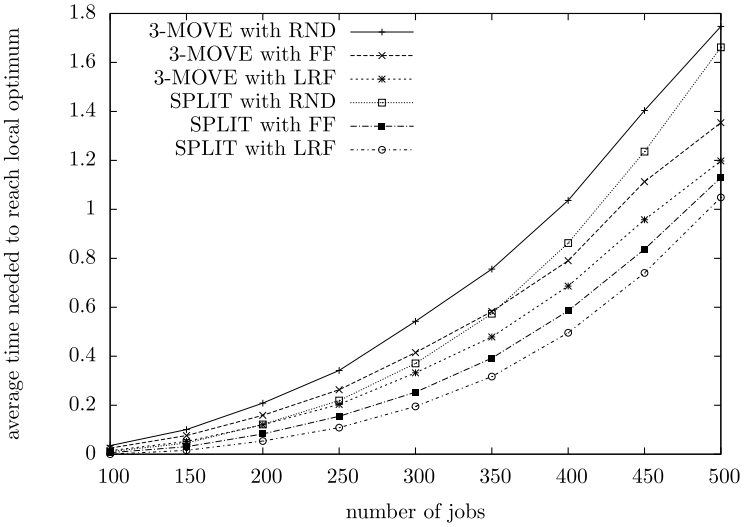

Fig. 10 Average running time in seconds to reach local optimum (type Eq. 2 instances with $\frac{n}{m} = 10$)

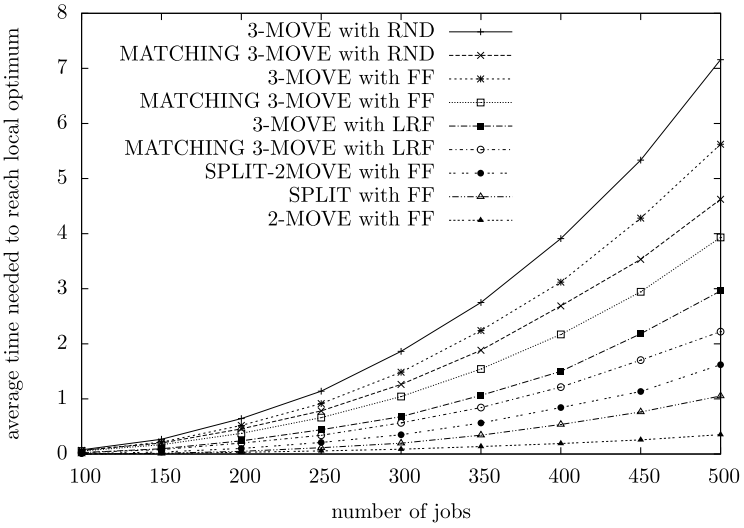

Fig. 11 Average running time in seconds to reach local optimum (type Eq. 2 instances with $\frac{n}{m} = 25$)

than the solutions resulting from an initial solution of good quality. Nevertheless, the differences are not that big. Indeed, k-MOVE with $k = 2, 3, 4$ is able to bring initial solutions of bad quality in the range of using initial solutions obtained by the LRF-heuristic. This holds for all types of instances. To the contrary, additional tests have shown that 1-MOVE is very sensitive to the choice of initial solutions in all types of instances. SPLIT is also sensitive to the choice of initial solutions, but performs better than 1-MOVE using bad initial solutions.

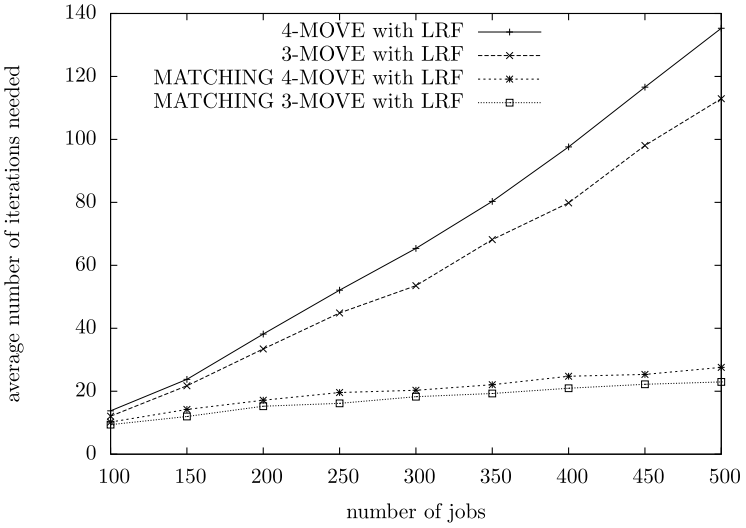

Fig. 12 Average number of iterations to reach local optimum (type Eq. 2 instances with $\frac{n}{m} = 25$)

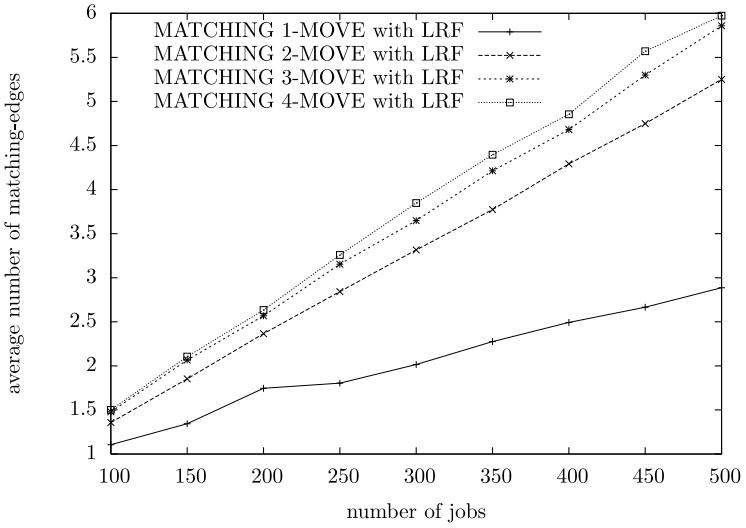

Fig. 13 Average number of edges in maximum weighted matching (type Eq. 2 instances with $\frac{n}{m} = 25$)

In a last series of tests we compare k-MOVE to MATCHING k-MOVE. The experiments indicate, that k-MOVE and MATCHING k-MOVE are performing nearly the same regarding solution quality for all types of instances, although the matching based approaches are always slightly better. The results for $k = 2$, $\frac{n}{m} = 25$ and instances of type Eq. 1 can be seen in Fig. 9.

Summarizing the test on the quality of the solutions, we can state that not the size of the neighborhood but the structure of the underlying basic neighborhoods are

mainly responsible for the achieved quality. The basic neighborhoods have to allow enough variation; independent combinations of these basic neighborhood operators do not give a real improvement of the navigation behavior.

The results presented till now, concentrated on the quality of the achieved solutions. In the next part we investigate the running time. We use instances of type Eq. 2 to present the results on the running time, since for this type of instances all our methods need the most time to reach a local optimum. The first observation is, that regardless which method we use, starting with initial solutions of bad quality leads to a higher average running time of the methods (see Fig. 10 and Fig. 11). Additionally, we observe, that the average time needed to reach a local optimum by using MATCHING k-MOVE is smaller than using the basic k-MOVE for all k (see Fig. 11). For $\frac{n}{m} = 25$, the running time of SPLIT is comparable to 2-MOVE (see Fig. 11) and the average time needed to reach a local optima by using the combined approach SPLIT-2MOVE is slightly higher than for SPLIT alone.

The decrease in the computational times using MATCHING k-MOVE instead of k-move is a result from the fact that MATCHING k-MOVE needs a smaller number of iterations to reach the local optimum (see Fig. 12). The lower running time and less number of iterations for MATCHING k-MOVE compared to k-MOVE is caused by the ability to apply several operations in a single iteration. In Fig. 13 we present the average number of edges contained in a maximum weighted matching for instances of type Eq. 2 with $\frac{n}{m} = 25$.

Summarizing, for the considered parallel machine scheduling problem combining independent moves in one operator has a positive effect on the running time but does not help too much to improve the quality of achieved local optima.

6 Concluding remarks

We presented a general approach to build up a neighborhood $\mathcal{N}^M_{k\text{-move}}$ of exponential size out of a smaller basic neighborhood $\mathcal{N}_{k\text{-move}}$. Furthermore, we presented a neighborhood $\mathcal{N}_{\text{split}(q)}$ that is based on a splitting of the job sets on the machines. Computational tests show that using the neighborhood $\mathcal{N}^M_{k\text{-move}}$ instead of $\mathcal{N}_{k\text{-move}}$ is on average a better choice. Doing so, has an impact on the time needed to reach a local optimum. Moreover, the solution quality is only slightly better by using the matching based methods instead of the basic ones.

The neighborhood $\mathcal{N}_{\text{split}(q)}$ delivers better results compared to $\mathcal{N}^M_{1\text{-move}}$ and $\mathcal{N}_{1\text{-move}}$ but is not as good as $\mathcal{N}^M_{2\text{-move}}$ and $\mathcal{N}_{2\text{-move}}$. The average time needed to reach a local optimum is comparable to $\mathcal{N}_{2\text{-move}}$. By using the two neighborhoods in a combined approach, the quality of local optima increase but with an expense in running time.

The results of this paper somehow confirm the conclusions drawn on the practical use of neighborhoods of exponential size (see e.g. Hurink 1999, or Brueggemann and Hurink 2007). Very large-scale neighborhoods are not per definition good for making local search efficient. Based on our experiences, we may conclude that the size of the neighborhood does not guarantee a better quality. And only if structural properties of the considered problem make it possible to combine several neighborhood operators,

very large-scale neighborhoods derived by combining operators may be successful in speeding up the computational time (which was for our problem the case).

All in all, we suggest to develop and use very large-scale neighborhoods of the considered types only if problem specific properties or computational arguments give an indication that the very large-scale neighborhoods have some potential to be a success.

Acknowledgements The authors are grateful to the anonymous referee for the helpful comments on an earlier draft of the paper.

References

Agarwal, R., Ergun, Ö., Orlin, J.B., Potts, C.N.: Solving parallel machine scheduling problems with very-large scale neighborhood search. Working paper (2007). http://www2.isye.gatech.edu/~oergun/publications/ParallelMachineJOS.pdf, J. Sched., to appear

Ahuja, R.K., Özlem, E., Orlin, J.B., Punnen, A.P.: A survey of very large-scale neighborhood search techniques. Discrete Appl. Math. **123**, 75–102 (2002)

van den Akker, J.M., Hoogeveen, J.A., van de Velde, S.L.: Parallel machine scheduling by column generation. Oper. Res. **47**, 862–872 (1999)

Baker, K.R., Merten, A.G.: Scheduling with parallel processors and linear delay costs. Nav. Res. Logist. Q. **20**, 793–804 (1973)

Barnes, J.W., Laguna, M.: Solving the multiple-machine weighted flow time problem using tabu search. IIE Trans. **25**(2), 121–128 (1993)

Belouadah, H., Potts, C.N.: Scheduling identical parallel machines to minimize total weighted completion time. Discrete Appl. Math. **48**(3), 201–218 (1994)

Brucker, P.: Scheduling Algorithms, 4th edn. Springer, Berlin (2004)

Brueggemann, T., Hurink, J.L.: Two exponential neighborhoods for single machine scheduling. OR Spectrum **29**, 513–533 (2007)

Chen, Z.L., Powell, W.B.: Solving parallel machine scheduling problems by column generation. INFORMS J. Comput. **11**, 78–94 (1999)

Congram, R.K., Potts, C.P., van de Velde, S.L.: An iterated dynasearch algorithm for the single machine total weighted tardiness problem. INFORMS J. Comput. **14**, 52–67 (2002)

Deineko, V., Woeginger, G.J.: A study of exponential neighborhoods for the traveling salesman problem and the quadratic assignment problem. INFORMS J. Comput. **14**, 52–67 (2000)

Eastman, W.L., Even, S., Isaacs, I.M.: Bounds for the optimal scheduling of n jobs on m processors. Manag. Sci. **11**, 268–279 (1964)

Edmonds, J.: Maximum matching and a polyhedron with 0,1-vertices. J. Res. Natl. Bureau Stand. B **69**, 125–130 (1965)

Elmaghraby, S.E., Park, S.H.: Scheduling jobs on a number of identical machines. Trans. Am. Inst. Ind. Eng. **6**, 1–12 (1974)

Ergun, Ö., Orlin, J.B., Steele-Feldman, A.: Creating very large-scale neighborhoods out of smaller ones by compounding moves. J. Heuristics **12**, 115–140 (2006)

Gabow, H.N.: Implementation of algorithms for maximum matching on nonbipartite graphs. Ph.D. Thesis, Department of Computer Science, Stanford University, Stanford, California (1973)

Garey, M.R., Johnson, D.S.: Computers and Intractability. A Guide to the Theory of NP-Completeness. Freeman, New York (1979)

Graham, R.L., Lawler, E.L., Lenstra, J.K., Rinnooy Kan, A.H.G.: Optimization and approximation in deterministic sequencing and scheduling: a survey. Ann. Discrete Math. **5**, 287–326 (1979)

Hoede, C.: Private communication (2006)

Hurink, J.: An exponential neighborhood for a one machine batching problem. OR Spektrum **21**, 461–476 (1999)

Kawaguchi, T., Kyan, S.: Worst case bound of an LRF schedule for the mean weighted flow-time problem. SIAM J. Comput. **15**(4), 1119–1129 (1986)
Lawler, E.L.: Combinatorial Optimization: Networks and Matroids. Holt, Rinehart and Winston, New York (1976)
Lenstra, J.K., Rinnooy Kan, A.H.G., Brucker, P.: Complexity of machine scheduling problems. Ann. Discrete Math. **1**, 343–362 (1977)
Potts, C.N., van de Velde, S.L.: Dynasearch-iterative local improvement by dynamic programming: Part 1. The traveling salesman problem. Technical Report, University of Twente, Enschede, The Netherlands (1995)
Sahni, S.K.: Algorithms for scheduling independent tasks. J. Assoc. Comput. Mach. **23**, 116–127 (1976)
Skutella, M., Woeginger, G.J.: A PTAS for minimizing the total weighted completion time on identical parallel machines. Math. Oper. Res. **25**, 63–75 (2000)
Smith, W.E.: Various optimizers for single-stage production. Nav. Res. Logist. Q. **3**, 59–66 (1956). Math. Oper. Res. **25**, 63–75