Department of Applied Mathematics
Faculty of EEMCS

University of Twente
The Netherlands

P.O. Box 217
7500 AE Enschede
The Netherlands
Phone: +31-53-4893400
Fax: +31-53-4893114
Email: memo@math.utwente.nl
www.math.utwente.nl/publications

Memorandum No. 1773

Matching based exponential neighborhoods for parallel machine scheduling

T. Brueggemann and J.L. Hurink

August, 2005

ISSN 0169-2690

Matching Based Exponential Neighborhoods For Parallel Machine Scheduling

Tobias Brueggemann [1,*], Johann L. Hurink [2]

Department of Applied Mathematics, University of Twente, P.O. Box 217, 7500 AE Enschede, The Netherlands

Abstract

We study the minimum total weighted completion time problem on parallel machines, which is known to be strongly $\mathcal{NP}$-hard. We develop general ideas leading to exponential neighborhoods in which the best improving neighbor can be determined by calculating a maximum weighted matching of an improvement graph. In a second step we introduce neighborhoods that form the base for the exponential neighborhoods.

Key words: exponential neighborhoods, local search, parallel machines.
MSC2000: 90B35, 68M20.

1 Introduction

Many optimization problems in the practical world are computational intractable. It would simply cost too much time to solve them to optimality. Hence, there is need for a practical approach to solve such problems. A way too achieve this is the development of heuristic (approximation) algorithms that are able to find satisfying solutions within a reasonable amount of computational time. In the literature concerning heuristic algorithms two different classes can be distinguished. The first class of heuristic algorithms consists of constructive algorithms. These algorithms build solutions by assigning values

* Corresponding author.
Email addresses: t.brueggemann@math.utwente.nl (Tobias Brueggemann), j.l.hurink@math.utwente.nl (Johann L. Hurink).

[1] supported by the Netherlands Organization for Scientific Research (NWO) grant 613.000.225 (Local Search with Exponential Neighborhoods).
[2] supported by BSIK grant 03018 (BRICKS: Basic Research in Informatics for Creating the Knowledge Society).

to one or more decision variables at a time. The second class are the improvement algorithms, that start with a feasible solution and iteratively try to advance to a better solution. In this class, local search resp. neighborhood search algorithms play a big role.

A local search heuristic starts, roughly spoken, with some 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 of a given solution and a method to select a solution from the neighborhood of a given solution is needed.

The neighborhood structure of a given solution has an important influence on the efficency of the local search heuristic. The structure determines the navigation through the solution space during the iterations of the local search method and the computation time of one iteration is affected by the choice of the neighborhood structure as well. Therefore, one expects that the size of the neighborhood has influence on the quality of the final solution of a local search approach, because a larger neighborhood covers a bigger amount of solutions and of course, affects the running time. So, there has to be found a compromise between size, quality and running time.

A possible way to do this, is to restrict the neighborhood of a solution to promising solutions, i.e. to solutions which may have a good objective value. Another possibility is to develop efficient methods to find the best solution in a given neighborhood, which is often an interesting optimization problem itself.

In the last years, large-scale neighborhoods came into the picture. These large-scale neighborhoods mostly contain an exponential number of solutions but allow a polynomial exploration. A nice survey about large-scale neighborhood techniques is given by Ahuja et al. [1]. They categorize 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 received by subclasses or restrictions that can be solved in polynomial time.

In this work different approaches are presented for receiving efficiently searchable large-scale neighborhoods for the problem of scheduling independent jobs on parallel machines minimizing the weighted average completion time (using the notation given by Graham et al. [5] this problem is denoted by $P||\sum w_j C_j$). The neighborhoods consist mainly of matchings in a certain improvement graph. Until now, neighborhoods based on matchings are mostly used for ap-

proximating the traveling salesman problem and some vehicle routing problems, see Ahuja et al. [1]. According to the work of Ahuja et al. [1], the large-scale neighborhoods presented in this work belong to the second category of large-scale neighborhoods.

The outline of the paper is as follows. In Section 2 a brief description of the problems is given and some important notations are introduced. In Section 3 several neighborhoods are introduced and examined for the problem $P||\sum w_jC_j$.

2 Problem description

The problem of scheduling n jobs with processing times $p_j > 0$ and weights $w_j > 0$ on m identical parallel machines without preemption is considered. The goal is to find a solution minimizing the sum of weighted completion times.

A solution of the problem consists of an *assignment* $A : \{1, \ldots, n\} \rightarrow \{1, \ldots, m\}$ 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.

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

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

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

is minimized. This problem is strongly $\mathcal{NP}$-hard, as described in Lenstra et al. [7] and problem *[SS13]* in Garey and Johnson [4], and thus computational intractable. The proof can be done by a reduction from 3-Partition to instances with $p_j = w_j$, showing that already $P|p_j = w_j|\sum w_jC_j$ is strongly $\mathcal{NP}$-hard. Furthermore, for two machines the problem is also $\mathcal{NP}$-hard, but only in the weak sense.

In the one machine case ($m = 1$), the resulting problem $1||\sum w_jC_j$ is solvable in polynomial time by sorting the jobs in order of non-increasing weight to

processing time ratios $\frac{w_j}{p_j}$ (Smith's ratio, cf. Smith [8]). In the case of more than one machine ($m > 1$), jobs scheduled on different machines are not influencing each other. Moreover, the objective function $\sum w_j C_j$ 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 by non-increasing Smith's ratio. 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 ratio is given.

From now on we assume w.l.o.g. that the jobs are ordered according to Smith's ratio, i.e. $\frac{w_1}{p_1} \geq \frac{w_2}{p_2} \geq \ldots \geq \frac{w_n}{p_n}$. 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 by M_{ik} the job j which is processed as k-th job on machine i. Using the processing orders M, the schedule S corresponding to a given assignment A can be calculated by

$$\begin{aligned} S_{M_{i1}} &:= 0 \text{ and} \\ S_{M_{i,j+1}} &:= S_{M_{ij}} + p_{M_{ij}} \text{ for } j = 1, \ldots, n_i - 1, \end{aligned} \tag{2}$$

where n_i denotes the number of jobs assigned to machine i. Based on the above considerations, we may describe solutions of problem $P||\sum w_j C_j$ by assignments A. We denote by $f(A)$ the objective value of the schedule belonging to assignment A.

3 Neighborhoods for $P||\sum w_j C_j$

In the following we introduce a general idea leading to neighborhoods of up to exponential size for problem $P||\sum w_j C_j$. The basic principles presented in the following are not only restricted to neighborhood search for problem $P||\sum w_j C_j$ but are also adoptable for similar problems, which have the property, that the machines are independent (parallel); e.g. $Q||\sum w_j C_j$, $R||\sum w_j C_j$.

The main idea behind the presented approach to build large neighborhoods is to start with a rather simple basic neighborhood $\mathcal{N}_1$ and to build a new large 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. The presented approach to build the large neighborhood 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 neighborhoods $\mathcal{N}_1$ has to consist of operators $op(i,j)$ which operate on pairs (i,j) of different machines (i.e. $i \neq j$). 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 an operation $op(i,j)$ for a fixed pair $i \neq j$ is only dependent on the given schedules on the two machines i and j and has only effects on the machines i and j.
- the operator is symmetric, i.e. the assignments $A' := op(i,j)(A)$ and $A'' := op(j,i)(A)$ are equal

We now define two operators $op(i_1,i_2)$ and $op(i_3,i_4)$ to be independent if $i_j \neq i_k$ for $j \neq k$; i.e. the two machine pairs are disjoint. The first property guarantees that for a set of pairwise independent operators it does not matter in which sequence these operations are applied to a given solution. Furthermore, since the objective function $\sum w_j C_j$ splits up in m separate parts for the m machines, the first property also guarantees that for a set of pairwise independent operators the change in the objective value is additive. More precisely, if we apply a set of pairwise independent operators $op(i_1,j_1),\ldots,op(i_k,j_k)$ to a solution A the resulting change in the objective value is given by

$$f(op(i_k,j_k)(\ldots(op(i_1,j_1)(A))\ldots) - f(A) = \sum_{l=1}^{k} \delta_{i_l,j_l}(A)$$

whereby $\delta_{ij}(A) := f(op(i,j)(A)) - f(A)$ denotes the objective change resulting from the application of $op(i,j)$ to assignment A.

All basic neighborhoods $\mathcal{N}_1$ fulfilling the above properties form 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 received by applying a set of pairwise independent operators to the assignment A. In the following we treat the neighborhood $\mathcal{N}_1$ in more detail. We describe how a best neighbor in this neighborhood can be obtained and give some bound on the size of the neighborhood.

The operators of the neighborhood $\mathcal{N}_2$ can be represented by matchings. To achieve this, we consider the following weighted graph $G(A) = (V,E,c(A))$, where

- the vertex set V contains a vertex for each machine (i.e. there are m vertices).
- an edge $e = (i,j) \in E$ $(i \neq j)$ represents the operator $op(i,j)$ (and due to the above symmetry property also the operator $op(j,i)$)

- an edge $e = (i, j)$ gets a weight $c(A)_{ij} = \delta_{i,j}(A)$ representing the change in the objective value resulting from applying the operator $op(i, j)$ to assignment A.

The weights of the graph can be calculated by applying the operator $op(i, j)$ for every pair of machines i, j with $i < j$ to assignment A; i.e. 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 $op(i, j)$.

A matching M in this graph is given by a subset of edges $M \subseteq E$, such that no two edges have a vertex in common. Therefore, each matching 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 $w(M)$ of a matching M is given by the sum of the weights of all edges present in the matching, i.e.

$$w(M) := \sum_{ij \in M} c(A)_{ij} = \sum_{ij \in M} \delta_{ij}(A).$$

Again, this weight $w(M)$ is equal to the change in the objective value resulting from applying the operator of $\mathcal{N}_2$ belonging to the matching M. Hence, we can determine the best neighbor of a solution A in neighborhood $\mathcal{N}_2$ by calculating a maximum weighted matching M in the graph $G(A)$. Observe, that the structure of the graph $G(A)$ is independent of the solution A but the weights heavily depend on it.

Determining a maximum weighted 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 [3] and Lawler [6] extending the work of Edmonds [2]. Using this algorithm, the best neighbor in $\mathcal{N}_2$ can be determined in $\mathcal{O}(m^3)$.

In order to give bounds on the number of neighbors of a solution A in the neighborhood $\mathcal{N}_2$, we have to give a bound on the number M_m of different matchings in a complete graph K_m with m vertices:

- the number M_m^* of maximal cardinality matchings in K_m is given by

$$M_m^* = \frac{m!}{l!2^l} \geq \left(\frac{\sqrt{m}}{2}\right)^m,$$

  where $l := \lfloor \frac{m}{2} \rfloor$
- a matching of size $k \leq \lfloor \frac{m}{2} \rfloor$ in K_m exists of $m - 2k$ isolated vertices and a complete subgraph of size $2k$ with a maximal cardinality matching. Therefore, the number M_m^k of matchings of size k in K_m is given by

$$M_m^k = \binom{m}{m-2k} \cdot M_{2k}^* = \binom{m}{2k} \cdot M_{2k}^*.$$

- summing this up for every k yields

$$M_m = \sum_{k=0}^{l} \binom{m}{2k} \cdot \frac{(2k)!}{k!2^k} = \sum_{k=0}^{l} \frac{m!}{k!(m-2k)! \cdot 2^k}$$
$$\geq \sum_{k=0}^{l} \binom{m}{k} \left(\frac{m-2k+1}{2}\right)^k \geq \frac{1}{2}\sqrt{2^m}$$

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 basis neighborhood $\mathcal{N}_1$, which fulfill the stated properties for this neighborhood.

3.1 Move Neighborhood

A first example of a basis neighborhood $\mathcal{N}_1$ is built up using move operators $op^{move}(i,j)$. Hereby, $op^{move}(i,j)$ considers the machines i and j of the given assignment A and moves 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 $op^{move}(i,j)$ can be realized efficiently.

In principle, an operator $op^{move}(i,j)$ represents a best possible move in a neighborhood consisting of operators $move(k,l)$, that reassign job k to machine l, where

- for k only jobs currently assigned to machine i or j are allowed,
- for l only the machines i and j are allowed and
- for l only the machine where k currently is not assigned to is allowed.

In the following we study the effects of a single operator $move(k,l)$ and show how on base of these results the effect of an operator $op^{move}(i,j)$ can be calculated.

In order to evaluate the effects of a move of job k from its machine $A(k)$ to the target machine l, denote by τ_1 the position at which job k is scheduled on machine $A(k)$ w.r.t. assignment A and by τ_2 the position at which job k has to be inserted on machine l.

If job k is deleted from machine $A(k)$, the completion times of the jobs $M_{A(k),\tau_1+1}, \ldots, M_{A(k),n_{A(k)}}$ decrease by p_k units. This lowers the objective value

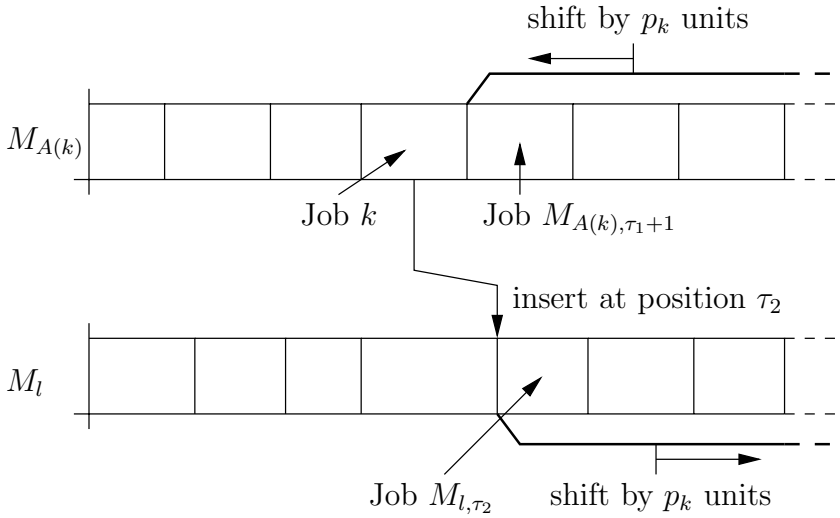

Fig. 1. $move(k,l)(A)$

by

$$\delta_1 = \left(W_{A(k),n_{A(k)}} - W_{A(k),\tau_1}\right) p_k,$$

where $W_{ik} := \sum_{l=1}^{k} w_{M_{il}}$. Inserting job k on the target machine l on position τ_2 increases the completion times of the jobs $M_{l,\tau_2}, \ldots, M_{l,n_l}$ by p_k units and raises the objective value by

$$\delta_2 = \left(W_{l,n_l} - W_{l,\tau_2-1}\right) p_k$$

(see Figure 1 for an illustration).

Taking into account the change in completion time of job k, the overall change in the objective value $\delta^A_{move(k,l)} := f(A) - f(move(k,l)(A))$ resulting from an application of $move(k,l)$ to solution A is given by

$$\delta^A_{move(k,l)} = \delta_1 - \delta_2 + w_k \left(C_k - p_k - C_{M_{l,\tau_2-1}}\right) \tag{3}$$

Thus, if the values W_{ik} and the positions τ_1 and τ_2 are known, $\delta^A_{move(k,l)}$ can be calculated in $\mathcal{O}(1)$.

For the operator $op^{move}(i,j)$ we now have to find the best move of a job between the two machines i and j. This can be achieved by evaluating all moves $move(k,j)$ for jobs k with $A(k) = i$ and all moves $move(k,i)$ for jobs k with $A(k) = j$. Since in a preprocessing, the relevant W_{ik}-values and the relevant positions τ_1 and τ_2 can be calculated in $\mathcal{O}(n_i + n_j)$, the overall complexity to evaluate the operator $op^{move}(i,j)$ is $\mathcal{O}(n_i + n_j)$. For the neighborhood $\mathcal{N}_2$ we have to evaluate all operators $op^{move}(i,j)$ with $i < j$ to build up the graph $G(A)$. This can be realized in $\mathcal{O}(nm)$.

The neighborhood $\mathcal{N}_2$ contains an in m exponential number of neighbors. Each of these neighbors again dominates a certain number of neighbors w.r.t. the

neighborhood defined by $move(k, l)$ operators (we call this neighborhood $\mathcal{N}_0$). More precisely, consider a matching M containing the edges $(i_1, j_1), \ldots, (i_k, j_k)$ in $G(A)$. This matching corresponds to an assignment A' with

$$A' = op^{move}(i_1, j_1) \circ op^{move}(i_2, j_2) \circ \cdots \circ op^{move}(i_k, j_k)(A).$$

Each edge (i_l, j_l) can be understood as an operator $op^{move}(i_l, j_l)$ applied to assignment A. Observe, that $op^{move}(i_l, j_l)$ dominates $n_{i_l} + n_{j_l}$ operators in the neighborhood $\mathcal{N}_0$. Therefore, the considered matching M using operator $op^{move}(i_l, j_l)$ for each edge (i_l, j_l) represents the best solution in a neighborhood of size $\prod_{l=1}^{k}(n_{i_l} + n_{j_l})$. In contrast to this, neighborhood $\mathcal{N}_1$ (i.e. move a job between a pair of machines) contains only $\frac{1}{2}m(m-1)$ neighbors, where each neighbor represents the best solution in a neighborhood of size $n_i + n_j$.

3.2 Combining several moves

Another example for a basis neighborhood $\mathcal{N}_1$ can be received by not only moving one job from a machine to another, but swapping two jobs between a pair of machines. Of course, this can be generalized by allowing a fixed amount of moves between a pair of machines in one step. In the next part we introduce the foundations for this idea. We develop an important technique to combine several move-operators to exchange several jobs between machines l_s and l_t, where l_s and l_t are fixed. A general method is introduced to calculate the effects to the objective value of such combined operators.

We consider an assignment A_1 that is received from an assignment A by applying several move operations. We are interested in which situations the values $\delta^A_{move(k,l_t)}$ can still be used to calculate the objective change resulting from applying $move(k, l_t$ to assignment A_1. Recall, that the values $\delta^A_{move(k,l_t)}$ denote the change in the objective value by moving job k to machine l_t in assignment A. Clearly, it makes only sense to consider the case that job k is still assigned in A_1 to the same machines as in A, i.e. we restrict to assignments A_1 in which job k is processed by the same machine as in assignment A.

Let job k be a job that is processed by machine $l_s := A(k)$ in assignment A and in assignment A_1. Denote with τ_1 the position of job k on machine l_s in assignment A, i.e. $k = M^A_{l_s,\tau_1}$. By emanating from assignment A to assignment A_1, the position of job k on it's machine may have changed due to insertions and deletions of jobs scheduled on machine l_s prior to job k. Therefore, we have a possibly different value τ_1' describing the position of job k on machine l_s in assignment A_1. The change in completion time of job k by going from assignment A to A_1 is given by

$$\Delta C_{l_s} := C_k^{A_1} - C_k^A. \tag{4}$$

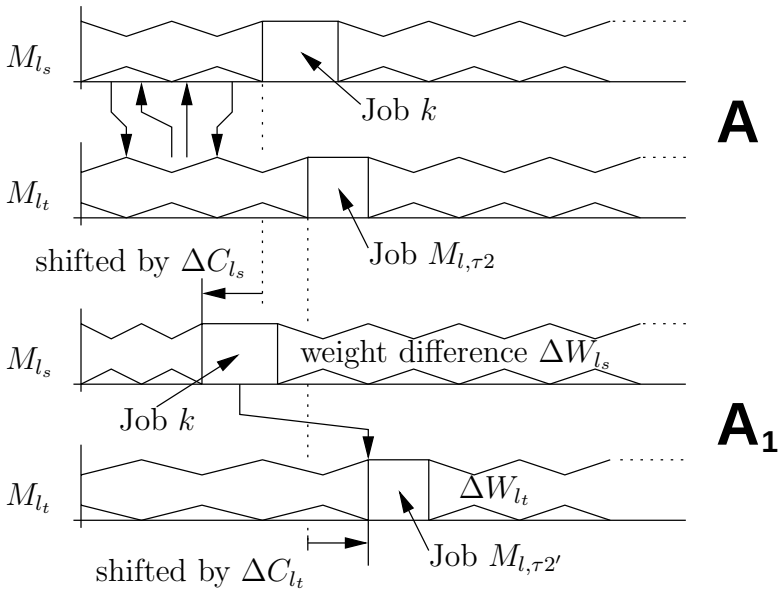

Fig. 2. combine several moves

Furthermore, by going from assignment A to A_1, insertions and deletions may have happened on machine l_s after job k's processing. This also influences a move of job k to machine l_t in assignment A_1. In order to take this into account, we define the value ΔW_{l_s} to denote the change of weight on machine l_s after job k. The value ΔW_{l_s} calculates as

$$\Delta W_{l_s} := W^{A_1}_{l_s,n_{l_s}} - W^{A_1}_{l_s,\tau_1'} - W^{A}_{l_s,n_{l_s}} + W^{A}_{l_s,\tau_1}. \tag{5}$$

Next, we concentrate on the target machine l_t. To calculate the effect of moving job k to machine l_t we need to know the starting time of the job scheduled at the insert position. We denote with τ_2 and τ_2' the insert positions of job k on machine l_t in assignment A resp. A_1. Again, the starting time of the job scheduled at the insert position may have changed in assignment A_1 as well as the job scheduled at this position. Hence, we define the value ΔC_{l_t} describing this change as

$$\Delta C_{l_t} := S^{A_1}_{M^{A_1}_{l_t,\tau_2'}} - S^{A}_{M^{A}_{l_t,\tau_2}}. \tag{6}$$

Of course, changes to the weights of the jobs following $M^A_{l_t,\tau_2}$ and $M^{A_1}_{l_t,\tau_2'}$ on machine l_t have also to be taken into account. For this we define the value ΔW_{l_t} for the target machine similar to ΔW_{l_s} for the source machine to be

$$\Delta W_{l_t} := W^{A_1}_{l_t,n_{l_t}} - W^{A_1}_{l_t,\tau_2'-1} - W^{A}_{l_t,n_{l_t}} + W^{A}_{l_t,\tau_2-1}. \tag{7}$$

For an illustration of the situation see Figure 2. The assignments A and A_1 are shown as well as job k and the prior described effects influencing the change

in the objective value. With the help of the values defined in Equations (4), (5), (6), (7) we can state the following lemma.

Lemma 1 *Let assignments A and A_1 be given with the above mentioned properties, let k be a job scheduled on machine l_s in assignment A and A_1, and let l_t be the designated target machine. Then*

$$\delta^{A_1}_{move(k,l_t)} = \delta^{A}_{move(k,l_t)} + w_k(\Delta C_{l_t} - \Delta C_{l_s}) + p_k(\Delta W_{l_s} - \Delta W_{l_t}).$$

PROOF.

$$\begin{aligned}
\delta^{A_1}_{move(k,l_t)} = &(W^{A_1}_{l_s,n_{l_s}} - W^{A_1}_{l_s,\tau_1'})p_k - \\
&(W^{A_1}_{l_t,n_{l_t}} - W^{A_1}_{l_t,\tau_2'-1})p_k + \\
&(C^{A_1}_k - p_k - S^{A_1}_{M^{A_1}_{l_t,\tau_2'}})w_k \\
= &(\Delta W_{l_s} + W^{A}_{l_s,n_{l_s}} - W^{A}_{l_s,\tau_1})p_k - \\
&(\Delta W_{l_t} + W^{A}_{l_t,n_{l_t}} - W^{A}_{l_t,\tau_2-1})p_k + \\
&(\Delta C_{l_s} + C^{A}_k - p_k - \Delta C_{l_t} + S^{A}_{M^{A}_{l_t,\tau_2}})w_k \\
= &\delta^{A}_{move(k,l_t)} + w_k(\Delta C_{l_s} - \Delta C_{l_t}) + p_k(\Delta W_{l_s} - \Delta W_{l_t}). \qquad \square
\end{aligned}$$

Lemma 1 is the base of combining several move operators by using the additive term of the lemma to adapt the values $\delta_{move(k,l_t)}$. In the next two subsections, we give examples of possible combinations of move operators. The combined move operators then again can be used to build an exponential neighborhood that can be explored by the matching method.

3.2.1 Swap Neighborhood

The second example of a basis neighborhood $\mathcal{N}_1$ is built up using swap operators $op^{swap}(i,j)$. Hereby, $op^{swap}(i,j)$ is a combination of two move operators, which considers the machines i and j of the given assignment A and moves exactly one job from machine i to j and one job from machine j to i. 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 these operators $op^{swap}(i,j)$ can be realized efficiently. Hereby, amongst other things, Lemma 1 will be used.

The operator $op^{swap}(i,j)$ represents a best possible neighbor in a neighborhood consisting of operators $swap(l,k)$ on two jobs l and k, that reassign job l to machine $A(k)$ and job k to machine $A(l)$, where $A(l) \neq A(k)$. We study the

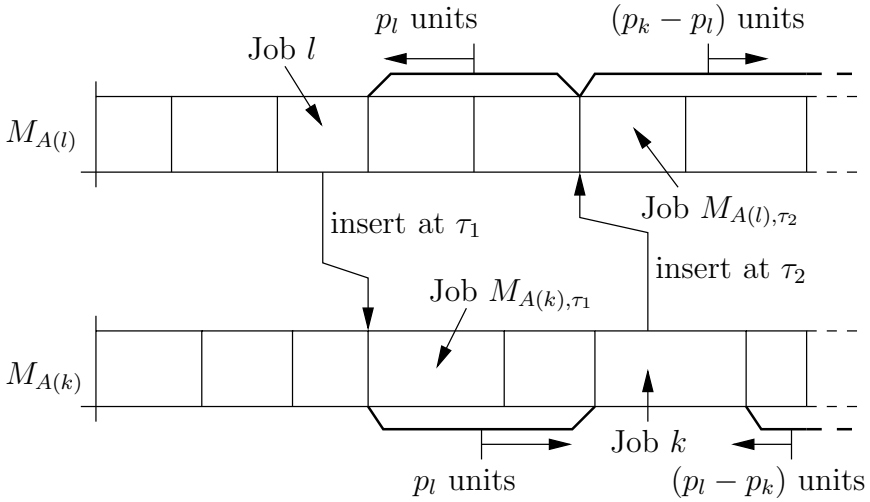

Fig. 3. $swap(l,k)(A)$

effects of a single operator $swap(l,k)$ and show how on base of these results the effect of an operator $op^{swap}(i,j)$ can be calculated.

The operator $swap(l,k)$ consists of two move operators $move(l,A(k))$ and $move(k,A(l))$, i.e. $swap(l,k)(A) = move(k,A(l))(move(l,A(k))(A))$, see Figure 3 for an illustration of the swap operator. For sake of simplicity we assume that job l is of higher priority than job k, i.e. Smith's ratio for job l and k satisfies

$$\frac{w_l}{p_l} \geq \frac{w_k}{p_k}.$$

In oder to examine the swap operator we make use of Lemma 1. We define assignment A_1 as the assignment that arises from A by moving job l from machine $A(l)$ to $A(k)$, i.e. $A_1 := move(l,A(k))(A)$. Then $swap(l,k)(A) = move(k,A(l))(A_1)$.

It remains to express the value $\delta^{A_1}_{move(k,A(l))}$ in relation to $\delta^{A}_{move(k,A(l))}$. The assignments A and A_1 differ only by job l, which in assignment A_1 is processed by machine $A(k)$. Hence, the starting times of all jobs following job l on machine $A(l)$ have decreased by p_l time units. Because of this, the starting time of the job that is scheduled at job k's insert position on machine $A(l)$ has also decreased by p_l time units. Thus, for Equation (6) we receive $\Delta C_{l_t} = -p_l$.

Additionally, job l has been inserted on machine $A(k)$. Therefore, all jobs following job l on machine $A(k)$ start p_l time units later, resulting in an increase of job k's starting time by p_l time units. This causes for Equation (4) that $\Delta C_{l_s} = p_l$.

Job l has a higher priority than job k. Hence, there are no other influences in A_1 on the jobs following job k on machine $A(k)$ and the jobs following job k's insert position on machine $A(l)$. From this follows, that with Equation (5)

and (7) we receive $\Delta W_{l_s} = 0$ and $\Delta W_{l_t} = 0$. With Lemma 1 we can conclude, that

$$\delta^{A_1}_{move(k,A(l))} = \delta^{A}_{move(k,A(l))} + 2w_k p_l.$$

Summarizing, the change in the objective value by applying the swap operator $swap(l,k)$ with $l < k$ on assignment A calculates as

$$\delta^{A}_{swap(l,k)} := \delta^{A}_{move(l,A(k))} + \delta^{A}_{move(k,A(l))} + 2w_k p_l. \tag{8}$$

Thus, if the values $\delta^{A}_{move(l,A(k))}$ and $\delta^{A}_{move(k,A(l))}$ are known in advance, $\delta^{A}_{swap(l,k)}$ can be calculated in $\mathcal{O}(1)$.

For the operator $op^{swap}(i,j)$ we now have to find the best exchange of jobs from machine i and j. If by a preprocessing the values $\delta^{A}_{move(k,j))}$ and $\delta^{A}_{move(l,i))}$ are known, the overall complexity of $op^{swap}(i,j)$ is $\mathcal{O}(n_i n_j)$. For the neighborhood $\mathcal{N}_2$ we have to evaluate all operators $op^{swap}(i,j)$ with $i < j$ to build up the graph $G(A)$. This can be realized in $\mathcal{O}(n^2)$.

The neighborhood $\mathcal{N}_2$ contains an in m exponential number of neighbors. Each of these neighbors again dominates a certain number of neighbors w.r.t. the neighborhood defined by the *swap* operators (we again call this neighborhood $\mathcal{N}_0$). More precisely, consider a matching M containing the edges $(i_1,j_1),\ldots,(i_k,j_k)$ in $G(A)$. This matching corresponds to an assignment A' with

$$A' = op^{move}(i_1,j_1) \circ op^{move}(i_2,j_2) \circ \cdots \circ op^{move}(i_k,j_k)(A).$$

Each edge (i_l,j_l) can be understood as an operator $op^{swap}(i_l,j_l)$ applied to assignment A. Observe, that $op^{swap}(i_l,j_l)$ dominates $n_{i_l} n_{j_l}$ operators in the neighborhood $\mathcal{N}_0$. Therefore, the considered matching M using operator $op^{swap}(i_l,j_l)$ for each edge (i_l,j_l) represents the best solution in a neighborhood of size $\prod_{l=1}^{k}(n_{i_l} n_{j_l})$. In contrast to this, neighborhood $\mathcal{N}_1$ (i.e. swap a pair of jobs between a pair of machines) contains only $\frac{1}{2}m(m-1)$ neighbors, where each neighbor represents a best solution in a neighborhood of size $n_i n_j$.

3.2.2 $\overline{\alpha}-move$ Neighborhood

The ideas of the *move* and *swap* operators now is generalized with the help of Lemma 1. This generalization leads to another basis neighborhood $\mathcal{N}_1$ built up using $\overline{\alpha}-move$ operators $op^{\overline{\alpha}-move}(i,j)$. Hereby, $op^{\overline{\alpha}-move}(i,j)$ considers the machines i and j of the given assignment A and moves up to α jobs between machines i and j. This is done in such a way that the change in the objective value is best possible. These operators are defined such that the stated properties for the basic neighborhood $\mathcal{N}_1$ are fulfilled. It remains to describe how these operators $op^{\overline{\alpha}-move}(i,j)$ can be realized efficiently.

An operator $op^{\overline{\alpha}-move}(i,j)$ represents a best possible move of up to α jobs between machines i and j. In the following we define a single operator that

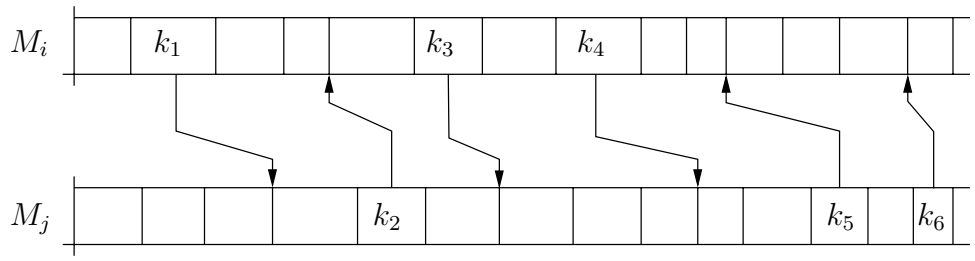

Fig. 4. $\alpha - move(\ldots)$

moves exactly α jobs between machines i and j and study the effects. For this we consider the fixed pair of machines i and j and only jobs k that are processed by one of these machines, i.e. $A(k) = i$ or $A(k) = j$. Moreover, by $\overline{A}(k)$ we denote the machine that job k is not assigned to, i.e.

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

This allows to simplify the definition of the operator *move* such that $move(k)(A) := move(k, \overline{A}(k))(A)$. For a fixed α we examine now a combination of moves

$$\alpha - move(k_1, \ldots, k_\alpha)(A) := move(k_\alpha) \circ move(k_{\alpha-1}) \circ \ldots \circ move(k_1)(A),$$

where $k_1 < k_2 < \ldots < k_\alpha$, i.e. job k_r has a higher priority than job k_{r+1} for $r = 1, \ldots, \alpha - 1$.

At first we consider the assignment

$$A_2 := move(k_2) \circ move(k_1)(A).$$

Due to Lemma 1 and a similar argumentation used in Section 3.2.1 for evaluating a swap, the difference of the objective values of assignments A and A_2 can be calculated by

$$\begin{aligned} f(A_2) - f(A) = \delta^A_{move(k_1, \overline{A}(k_1))} + \\ \delta^A_{move(k_2, \overline{A}(k_2))} + 2w_{k_2} p_{k_1} \Delta_{k_2,k_1}. \end{aligned} \tag{9}$$

Hereby, $\Delta_{k_s,k_t} = 1$ if job k_t and k_s are assigned to different machines in A and conversely, $\Delta_{k_s,k_t} = -1$ if job k_t is assigned to the same machine as job k_s, i.e.:

$$\Delta_{k_s,k_t} := \begin{cases} 1 & \text{if } A(k_s) \neq A(k_t), \\ -1 & \text{if } A(k_s) = A(k_t). \end{cases}$$

This generalizes to the following lemma.

Lemma 2 *Let $A' := \alpha - move(k_1, \ldots, k_\alpha)(A)$. The change in the objective value $\delta^A_{\alpha - move(k_1,\ldots,k_\alpha)} = f(A) - f(A')$ calculates as*

$$\begin{aligned}\delta^A_{\alpha-move(k_1,\ldots,k_\alpha)} = &\sum_{u=1}^{\alpha} \delta^A_{move(k_u,\overline{A}(k_u))} + \\ &\sum_{u=1}^{\alpha} 2w_{k_u} \sum_{v=1}^{u-1} \Delta_{k_u,k_v} p_{k_v}.\end{aligned} \tag{10}$$

PROOF. Consider for a fixed l with $1 < l \leq \alpha$ the assignment

$$A_{l-1} := move(k_{l-1}) \circ \ldots \circ move(k_1)(A)$$

and $A_l := move(k_l)(A_{l-1})$. We claim, that

$$f(A) - f(A_l) = \sum_{u=1}^{l} \delta^A_{move(k_u,\overline{A}(k_u))} + \sum_{u=1}^{l} 2w_{k_u} \sum_{v=1}^{u-1} \Delta_{k_u,k_v} p_{k_v}. \tag{11}$$

For $l = 2$, Equation (11) and (9) are identical. Suppose, that

$$f(A) - f(A_{l-1}) = \sum_{u=1}^{l-1} \delta^A_{move(k_u,\overline{A}(k_u))} + \sum_{u=1}^{l-1} 2w_{k_u} \sum_{v=1}^{u-1} \Delta_{k_u,k_v} p_{k_v}$$

holds. Now we prove, that the formula also holds for l. Every job of $k_1, \ldots, k_{l-1}$ that is processed on machine $A(k_l)$ in assignment A, is moved to machine $\overline{A}(k_l)$. Every job of $k_1, \ldots, k_{l-1}$ that is processed on machine $\overline{A}(k_l)$ in assignment A is moved to the machine on which job k_l is processed.

If a job of $k_1, \ldots, k_{l-1}$ is moved onto the machine of job k_l, it is inserted before job k_l because $k_l > k_r$ for $r = 1, \ldots, l-1$. Additionally, job k_l will be moved to a position on machine $\overline{A}(k_l)$ after the insert positions of all other jobs $k_1, \ldots, k_{l-1}$ that are moved to machine $\overline{A}(k_l)$.

Hence, the processing order on machine $A(k_l)$ is not changed in A_{l-1} onwards from job k_l and the processing order on machine $\overline{A}(k_l)$ is not changed in A_{l-1} onwards from the insert position of job k_l on machine $\overline{A}(k_l)$). Additionally, the starting time of job k_l in the schedule belonging to assignment A_{l-1} differs from the starting time in the schedule belonging to assignment A by

$$S^{A_{l-1}}_{k_l} = S^A_{k_l} + \sum_{v=1}^{l-1} \Delta_{k_l,k_v} p_{k_v}.$$

By using Lemma 1 we know, that

$$f(A) - f(A_l) = f(A) - f(A_{l-1}) + \delta^A_{move(k_l,\overline{A}(k_l))} + 2w_{k_l} \sum_{v=1}^{l-1} \Delta_{k_l,j_v} p_{k_v}$$

holds, and formula (11) for A_l follows. $\square$

The calculation of value $\delta^A_{\alpha-move(k_1,\ldots,k_\alpha)}$ with (10) needs in the worst-case a running time of $\mathcal{O}(\alpha^2)$, if the values $\delta^A_{move(k_u,\overline{A}(k_u))}$ are known in advance.

For the operator $op^{\overline{\alpha}-move}(i,j)$ we now have to find the best move of up to α jobs between machines i and j. To do so, denote with $M_i \cup M_j$ the set of jobs processed by machine i or j. For every β with $1 \leq \beta \leq \alpha$ we have to calculate for every possible subset $\{k_1,\ldots,k_\beta\} \subseteq M_i \cup M_j$ of size β the value $\delta^A_{\beta-move(k_1,\ldots,k_\beta)}$ to determine the best move of up to α jobs between machines i and j. If by a preprocessing the values $\delta^A_{move(k_u,\overline{A}(k_u))}$ are known, the overall complexity of $op^{\overline{\alpha}-move}(i,j)$ is $\mathcal{O}\left(\alpha^2(n_{i_1}+n_{i_2})^\alpha\right)$. For the neighborhood $\mathcal{N}_2$ we have to evaluate all operators $op^{\overline{\alpha}-move}(i,j)$ with $i<j$ to build up the graph $G(A)$. This can be realized in $\mathcal{O}(\alpha^2 m^2 n^\alpha)$. Thus, for constant values of α the neighborhood $\mathcal{N}_2$ can be evaluated in polynomial time.

The best neighbor in a neighborhood consisting of $\alpha-move$ operators working on machines i and j dominates

$$\binom{n_i+n_j}{\alpha}$$

neighbors. This means, that the operator $op^{\overline{\alpha}-move}(i,j)$ retrieves the best solution out of

$$\sum_{\beta=1}^{\alpha}\binom{n_i+n_j}{\beta}$$

solutions contained in the neighborhood $\mathcal{N}_0$ consisting of operators $\beta-move$ with $1 \leq \beta \leq \alpha$.

Consider a matching M containing the edges $(i_1,j_1),\ldots,(i_k,j_k)$ in $G(A)$. This matching is contained in the neighborhood $\mathcal{N}_2$ of up to (in m) exponential size. Each edge (i_l,j_l) corresponds to an operator $op^{\overline{\alpha}-move}(i_l,j_l)$. Therefore, the considered matching M using operator $op^{\overline{\alpha}-move}(i_l,j_l)$ for each edge (i_l,j_l) represents the best solution in a neighborhood of size

$$\prod_{l=1}^{k}\left(\sum_{\beta=1}^{\alpha}\binom{n_{i_l}+n_{j_l}}{\beta}\right).$$

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

4 Concluding remarks

We have presented a general approach to build up a neighborhood of up to exponential size out of a smaller basic neighborhood. This approach is applied using different basic neighborhoods. In further research we will examine, how the approach performs regarding quality and solution time in comparison to the basic neighborhoods and which type of basic neighborhoods lead to best results.

References

[1] R.K. Ahuja, E. Özlem, J. B. Orlin, A.P. Punnen [2002]: *A survey of very large-scale neighborhood search techniques*. In: Discrete Applied Mathematics 123, 75-102.

[2] J. Edmonds [1965]: *Maximum Matching and a Polyhedron with 0,1-vertices*. In: Journal of Research of the National Bureau of Standards, 69B, 125-130.

[3] H.N. Gabow [1973]: *Implementation of Algorithms for Maximum Matching on Nonbipartite Graphs*. Ph.D. Thesis, Departement of Computer Science, Stanford University, Stanford, California.

[4] M.R. Garey, D.S. Johnson [1979]: *Computers and intractability. A guide to the Theory of NP-Completeness*. W.H. Freeman and Company, New York.

[5] R.L. Graham, E.L. Lawler, J.K. Lenstra, A.H.G. Rinnooy Kan [1979]: *Optimization and approximation in deterministic sequencing and scheduling: A survey*. In: Annals of Discrete Mathematics 5, 287-326.

[6] E.L. Lawler [1976]: *Combinatorial Optimization: Networks and Matroids*. Holt, Rinehart and Winston, New York.

[7] J.K. Lenstra, A.H.G. Rinnooy Kan, P.Brucker [1977]: *Complexity of machine scheduling problems*. In: Annals of Discrete Mathematics 1, 343-362.

[8] W.E. Smith [1956]: *Various optimizers for single-stage production*. In: Naval Research Logistics Quarterly 3, 59-66.