Optimization Letters (2007) 1:213–225
DOI 10.1007/s11590-006-0020-7

ORIGINAL PAPER

Heuristic for a new multiobjective scheduling problem

Anne Setämaa-Kärkkäinen · Kaisa Miettinen · Jarkko Vuori

Received: 19 May 2006 / Accepted: 29 May 2006 /
Published online: 15 August 2006

Abstract We consider a telecommunication problem in which the objective is to schedule data transmission to be as fast and as cheap as possible. The main characteristic and restriction in solving this multiobjective optimization problem is the very limited computational capacity available. We describe a simple but efficient local search heuristic to solve this problem and provide some encouraging numerical test results. They demonstrate that we can develop a computationally inexpensive heuristic without sacrificing too much in the solution quality.

Keywords Heuristics · Parallel machine scheduling · Biobjective optimization · Combinatorial optimization · Telecommunications

1 Introduction

Future fourth generation (4G) wireless networks will enable global roaming across many wireless networks [18]. This means that mobile terminals will be able to use multiple wireless connections to networks simultaneously [5]. This

A. Setämaa-Kärkkäinen (✉)
Department of Mathematical Information Technology, University of Jyväskylä,
P.O. Box 35 (Agora), 40014 Jyväskylä, Finland
e-mail: annseta@mit.jyu.fi

K. Miettinen
Helsinki School of Economics, P.O. Box 1210, 00101 Helsinki, Finland
e-mail: miettine@hse.fi

J. Vuori
EVTEK University of Applied Sciences, Vanha maantie 6, 02650 Espoo Finland

provides valuable new potential because data to be transferred often consists of several distinct components. A web page, for example, is composed of multiple pictures and text parts. These different components can be transferred using distinct network connections. The connections may have very different characteristics, like price and transmission rate, which makes the wise selection of connections difficult for the user of the mobile terminal. Therefore, an automatic network connection selection method is needed.

The *network connection selection problem* [14] can be described as follows: A set of available network connections is given with known properties, such as transmission rate and price. Data consisting of distinct components is to be transferred using the network connections. The problem is to select a network connection for each component of the data in such a way that both the time used in transferring and the costs are minimized. This objective is chosen because the users of mobile terminals naturally want the data to be transferred as fast as possible while keeping the costs as low as possible. Usually, the faster the transfer is, the more expensive it is. Therefore, we need to find a compromise solution in which the transfer is fast enough while the costs are not too high.

Our goal is to develop a fast automatic solution method that gives a good compromise between the conflicting objectives. The method should be fast, because otherwise the usability of the mobile terminal impairs. In addition, the method should use little computational capacity because the capacity of mobile terminals is limited.

The network connection selection problem can be seen as a uniform parallel machine scheduling problem, where the goal is to minimize the makespan and the total costs of processing the jobs [14]. (Connections to other problem types are discussed in [14].) Using the three field notation introduced in [4], the problem can be stated as $Q||C_{\max}, \sum f_j$ where f_j denotes the cost of processing job j. The uniform parallel machine scheduling problem with the objective of minimizing the makespan is NP-hard [1]. Therefore, also the problem with the objectives of minimizing both the makespan and the costs is NP-hard. This means that it is unlikely that there exists a polynomial time algorithm capable of solving the problem. Because of the NP-hardness and the need for a fast solution method (also for large problem cases), we focus on developing a heuristic for the problem.

Some multiobjective scheduling problems have been considered in the literature, but the research has concentrated mostly on single machine problems [13,15]. We are not aware of any research on uniform parallel machine scheduling problems with the objectives of minimizing both the makespan and the costs. There are however some studies on unrelated parallel machine scheduling problems with these objectives [6,9]. The algorithms presented require that upper bounds T for the makespan and C for the costs are given. For example in [9], the algorithm forms a schedule with makespan at most $(1+\varepsilon)T$ and costs at most $(1+\varepsilon)C$ if there exists a schedule with makespan T and costs C. This kind of algorithms cannot be used for our network connection selection problem because we cannot determine values T and C a priori.

In [16], solving a general multiobjective scheduling problem has been divided into three subproblems: modelling the problem, taking into account the objectives and scheduling. The first phase, modelling the problem, means defining the scheduling problem and the objectives. In the second phase, the problem is to decide how the conflicting objectives will be taken into account, that is, how a compromise solution between the objectives can be obtained. In the last phase, the actual scheduling problem is solved and a solution to the multiobjective scheduling problem is obtained.

In our previous paper [14], we concentrated on modelling the network connection selection problem and taking into account the conflicting objectives. We compared different multiobjective optimization methods that do not need human interaction, studied the nature of the problem and found a method that produces a good compromise between the conflicting objectives. The method combines the two objectives into a scalarizing function that can be minimized in order to obtain a good compromise solution for the multiobjective optimization problem. Unfortunately, in practice, this method cannot be applied in solving the network connection selection problem because it is computationally far too demanding. Thus, we need new approaches.

In this paper, we continue our research on the network connection selection problem and consider the third phase: solving the problem. The goal of our study has been to develop a fast automatic solution method that uses only little computational capacity. Therefore, we have developed a heuristic solution method which we present here. The heuristic uses the scalarizing function found appropriate for solving the network connection selection problem in [14] to measure the solution quality. This idea of using a scalarizing function to measure the solution quality is new, since usually in heuristics developed for scheduling problems different objectives are considered separately. The heuristic is applicable also to other combinatorial multiobjective optimization problems.

The rest of this paper is organized as follows. In the next section,we give a mathematical model of the network connection selection problem. In Sect. 3, we define some concepts of multiobjective optimization and describe a scalarizing function. The scalarizing function is used in the heuristic we propose in Sect. 4. Computational results are given in Sect. 5, and finally, we conclude in Sect. 6.

2 Model

Let us assume that there are m network connections available and data consisting of n components is to be transferred using them. The connections available and their properties are known because the mobile terminal requests this information from network operators before the data transmission. Transmitting all the components is considered as a transaction that is time-sensitive. The time used in the transaction is assumed to be short and, therefore, the properties of the network connections can be assumed to be fixed during the transaction.

Let x_{ij} be a binary variable such that $x_{ij} = 1$, when component j is transferred using connection i, otherwise $x_{ij} = 0$, $i = 1, \ldots, m$, $j = 1, \ldots, n$. A coefficient, *duration*, d_{ij} represents the time used in transferring component j using connection i. Another coefficient c_{ij} represents the cost of using connection i to transfer component j.

Now, the network connection selection problem can be formulated as follows: minimize

$$f_1(x) = \max_{i=1,\ldots,m} \sum_{j=1}^{n} d_{ij} x_{ij} \quad \text{and} \quad f_2(x) = \sum_{i=1}^{m} \sum_{j=1}^{n} c_{ij} x_{ij}$$

subject to

$$\sum_{i=1}^{m} x_{ij} = 1, \quad \text{for all } j = 1, \ldots, n, \tag{1}$$

$$x_{ij} \in \{0, 1\}, \quad \text{for all } i = 1, \ldots, m \text{ and } j = 1, \ldots, n.$$

The objective function f_1 expresses the time used in transferring the components and the objective function f_2 denotes the costs of the transfers. The constraints (1) require that each component is transferred using exactly one network connection.

When the model is solved, the solution tells which connection is used for transferring each component. It does not pay attention to the order in which the components are transferred on each network connection because neither the time used in transfers nor the total costs depend on the order of the components. After solving the problem, the components can be ordered, for example, in the increasing order of the component sizes on each network connection.

It should be noted that since we cannot determine the exact transmission rate a priori, the minimum guaranteed rate of each connection given by the operator of the connection is used in calculating the durations. This ensures that the actual time used in each transmission is never longer than the duration d_{ij}.

3 Concepts of multiobjective optimization

Before we start solving the network connection selection problem, we briefly define some concepts of multiobjective optimization. Let us consider a problem where we want to minimize two conflicting objective functions $f_1(x)$ and $f_2(x)$ simultaneously subject to a general constraint $x \in S$. The vector of objective functions, called *objective vector*, is denoted by $F(x) = (f_1(x), f_2(x))^{\mathrm{T}}$, and the vector $x = (x_1, x_2, \ldots, x_n)^{\mathrm{T}}$ is called a *decision vector*.

Generally, it is not possible to find a solution in which both objective functions attain minimal values. A decision vector $x^* \in S$ and the corresponding objective vector $F(x^*)$ are *Pareto optimal* if there does not exist another decision vector $x \in S$ such that $f_i(x) \leq f_i(x^*)$ for $i = 1, 2$ and $f_j(x) < f_j(x^*)$ for at least one j [11].

Pareto optimality guarantees that we cannot improve any objective function value of the solution without deteriorating the other objective function value. In other words, all the Pareto optimal solutions are mathematically equivalent. Which of them is the best and is selected to be the final solution depends on the problem settings.

An objective vector containing the minimal value of each objective function is called an *ideal objective vector*, z^* [11]. With conflicting objectives, the ideal objective vector is infeasible. The components z_i^* of the ideal objective vector are obtained by minimizing both objective functions f_i separately subject to the constraint $x \in S$.

From the ideal objective vector we get the lower bound of the Pareto optimal set for each objective function. The upper bounds of the Pareto optimal set are the components of a *nadir objective vector* z^{nad}. In the case of two objective functions, the nadir objective vector can be obtained at the same time the ideal objective vector is calculated. The first component of the nadir objective vector is the value of the first objective function in the point where the second objective function attains its minimal value, and the second component is the value of the second objective function in the point where the first objective function attains its minimal value [3].

Multiobjective optimization problems are usually solved by scalarization [11]. *Scalarization* means converting the problem with multiple objectives into a single objective optimization problem or a family of single objective optimization problems. When a multiobjective optimization problem has been scalarized, methods developed for single objective optimization can be used for solving the problem. The objective function of the single objective problem is called a *scalarizing function*.

In [14], we compared different scalarizing functions (including the well-known weighting method) for solving the network connection selection problem. We next describe one of them that was found the most suitable for our purposes. This scalarizing function is used in the heuristic to be presented in Sect. 4.

The following achievement scalarizing function is minimized subject to the constraint of the problem:

$$\max_{i=1,2} w_i \frac{f_i(x) - z_i^{\text{mid}}}{z_i^{\text{nad}} - z_i^*} + \rho \sum_{i=1}^{2} w_i \frac{f_i(x) - z_i^{\text{mid}}}{z_i^{\text{nad}} - z_i^*}, \tag{2}$$

where z^{mid} is a middle point located in the middle of the ranges of the objective functions in the Pareto optimal set, that is,

$$z_i^{\text{mid}} = \frac{z_i^{\text{nad}} + z_i^*}{2}$$

for $i = 1, 2$ and the augmentation coefficient ρ is a small positive scalar. The ratio of the positive weighting coefficients w_1 and w_2 represents here the rate

at which the user of the mobile terminal is willing to trade off values of the objective functions. In (2), denominators are used to scale the terms of the scalarizing function to be of a similar magnitude. This increases the computational efficiency of this type of scalarizing functions, as shown in [12]. The solution to the achievement scalarizing function (2) is Pareto optimal [11].

Calculating ideal and nadir objective vectors means additional computations. To avoid that, we use approximations of the ideal and the nadir objective vectors. We use vector $(0,0)^{\mathrm{T}}$ to approximate the ideal objective vector. This vector is infeasible because no data can be transferred without any costs and without using any time. The nadir objective vector is estimated from the problem data as follows. The first component of the vector is the objective function value $f_1(x)$ related to a solution x where every component is transferred using the slowest network connection. The second component of the vector is the objective function value $f_2(x)$ related to a solution x where every component is transferred using the most expensive connection. These estimates are larger than or equal to the components of the real nadir objective vector. Though the estimate of the second component may be very rough, the estimates are sufficient for our purposes [14], and no optimization problems need to be solved to get them.

4 Heuristic

Because the capacity of mobile terminals is limited, our aim is to develop an algorithm that uses as little computational resources as possible. In other words, the heuristic should produce a good enough solution while being as simple as possible. In addition, the network connection selection problem requires that the solution is obtained fast. Otherwise, the user of the mobile terminal is not satisfied.

In [14], we studied the settings of the network connection selection problem using different problem instances. The instances represented different kinds of cases of the problem that may occur. In all the examples studied, we could identify a solution that is a good compromise between the objectives and, thus, a logical choice for the final solution. We studied different scalarizing functions in order to find a method that produces a solution near the good compromise solution. Our computational tests showed that the achievement scalarizing function (2) was the most suitable and robust for that purpose.

The problem has binary-valued variables which makes minimizing the scalarizing function using exact methods, such as branch-and-cut methods [10], time-consuming. Therefore, we need to develop a heuristic that gives a solution near the optimal solution to the scalarizing function (2). Our heuristic to be presented next is a simple local search method that uses the scalarizing function (2) to measure the solution quality.

Because we cannot use a lot of computational capacity in our heuristic, we use simple moves called 1-exchanges [17] for improving an initial solution. A 1-exchange means in this case that a component j that is assigned to a connection i is moved from connection i to another connection i'. This move

should improve the solution, otherwise it is not performed. We want to use 1-exchanges in a systematic way, and not to use random selection. The reason for this is that if we use random selection, 1-exchanges may be used for some component many times, whereas for some components the 1-exchanges are not applied at all, if there are only few iterations of 1-exchanges in the heuristic. Therefore, 1-exchanges are used for each component j in turn, and the potential new connection i' is chosen in the following way: The algorithm considers all the connections except connection i, and connection i' is the connection that gives the best solution when component j is transferred using it.

As already mentioned, we use the scalarizing function (2) to measure the solution quality. In other words, the lower the value of the scalarizing function is, the better the solution is. The parameters of the scalarizing function are given the following values: the augmentation coefficient ρ was set to 0.001 and the coefficients of the objective functions (time and costs) w_1 and w_2 were set to 1 and 2, respectively. These values were used also in [14]. This idea of using a scalarizing function to measure the solution quality is new: In heuristics developed for scheduling problems the objectives are usually considered separately.

The initial solution plays a very important role in getting a good final solution. It is desirable that the initial solution enables the improvements to lead the search near a good compromise solution. Therefore, we use the scalarizing function (2) also when forming the initial solution: We set each component in the initial solution to the network connection that gives the lowest scalarizing function value. This kind of an initial solution ensures that the scalarizing function measures the solution quality already when the first feasible solution is constructed.

To be more precise, the initial solution is formed as follows. A set C containing the components that have already been assigned to a network connection is initially empty. At each iteration, we consider a component i not yet in the set C, add it to the set and assign it to a connection. We define a *partial problem* as a problem consisting of all the connections and the components currently in C. The connection to which the component is assigned is the one that gives the lowest scalarizing function value for the partial problem. (Note that the assignments of the other components in C are fixed.) In order to be able to use the scalarizing function, the nadir objective vector of the partial problem has to be approximated. The vector is approximated as presented in Sect. 3.

Now we can sum up the heuristic algorithm as follows:

1. Initialize by setting $C = \emptyset$.
2. Construction of the initial solution. For each component $i = 1, \ldots, n$:
   (a) Add component i in the set C.
   (b) Approximate the nadir objective vector for the partial problem consisting of all the network connections and the components in C.
   (c) Assign component i to the network connection that gives the lowest value of the scalarizing function (2) for the current partial problem when the assignments of the other components in the partial problem

are fixed. The nadir objective vector approximated in the previous step is used in the scalarizing function.

3. Set the initial solution formed in the previous step as the current solution.
4. IMPROVEMENTS. For each component $i = 1, \ldots, n$, use 1-exchange:
   (a) Select the connection that gives the lowest value of the scalarizing function (2) when component i is transferred using it, instead of using the connection the component is assigned to in the current solution.
   (b) If the value of the scalarizing function is smaller than in the current solution, set component i to the selected connection and set this as the current solution.
5. Repeat Step 4 if the stopping criterion is not satisfied.

The stopping criterion can be a certain number of improvement rounds (Step 4) done or no improvement in the solution in the previous round of improvements. The stopping criterion can also be a combination of these criteria.

We have not yet specified in which order the components are dealt with in the heuristic. The 1-exchanges can be used for the components in a random order, which means, for example, the order in which the components of the data happen to be, or in the decreasing order of the component sizes. These orders can also be used in forming the initial solution. The decreasing order is intuitively appealing, because the larger the component is the more influence it has on the solution.

Without ordering the components, the computational complexity of the heuristic is $O(mn)$ (note that usually $m < n$). The ordering of the components can be done using quicksort, which has the worst case complexity of $O(n^2)$ and the mean complexity of $O(n \log n)$ [2]. Then, the computational complexity of the heuristic is $O(n^2)$ if the components are ordered.

We have tested the heuristic using both random order and the decreasing order. Thus, in the following computational results, there are two versions of the heuristic: in *heuristic1*, the components are dealt with in a random order, whereas in *heuristic2* the 1-exchanges are applied to the components in a decreasing order of their sizes. The heuristic is stopped in our computational tests when the solution has not improved during the previous round of improvements.

Finally, we want to remark that the scalarizing function used to measure the solution quality can be changed during the heuristic. In other words, we can use different scalarizing functions in forming the initial solution and in the improvemet phase, or we can use another scalarizing function after few rounds of improvements to refine the final solution.

5 Computational results

The two versions of the heuristic were tested with 16 reality-based test instances. By reality-based instances we mean that the components of each problem instance form a real web page and the properties of the network connections

(that is, price and transmission rate) are estimates for connections of the near future.

Because we cannot predict exactly what kind of a pricing model the operators will use in the future, we have used two different pricing models. In the first model, the pricing is linear in the size of the component. Thus, the price is, for example, $8.2 \cdot 10^{-8}$ euros per bit (and naturally the price is different for each connection). The second model is more elaborate: the price is calculated using formula

$$c_{ij} = K_0 + K_1 \exp(K_2 \cdot \text{rate}(i)) \cdot \text{size}(j), \tag{3}$$

where rate(i) is the rate of connection i (bits per second), *size*(j) is the size of component j in bytes and K_0, K_1, and K_2 are connection-specific coefficients. The coefficients K_0, K_1, and K_2 used for GSM connections are 9.0, 0.31, and 0.000043, respectively, and for UMTS connections the coefficients are 6.0, 8.8, and 0.0000017, respectively. The cost given by formula (3) has to be multiplied by 10^{-5} in order to get the cost in euros.

The test instances are briefly summarized in Table 1: For each instance, the number of components n and the number of network connections m as well as the total size of the components are given. The table reports also which of the pricing models is used in each test instance. We would like to note that the number of network connections m is small in the test instances because mobile terminals cannot use too many connections simultaneously.

As mentioned earlier, the solution obtained by minimizing the scalarizing function (2) is a good Pareto optimal compromise between the conflicting objectives. That is why we compare the performance of our heuristic to that solution. This optimal solution is calculated using CPLEX 8.0, which uses a branch-and-cut method to solve integer programming problems [7]. Both the heuristic and CPLEX were run in a computer with a 550 MHz HP PA-RISC processor. The relative optimality tolerance used in CPLEX was 10^{-8}, with the exception of

Table 1 Test instances (with n components and m connections)

Instance	n	m	Total size (bytes)	Pricing model
geu1	10	3	80,400	Linear
geu2	15	5	154,600	Linear
geu3	25	5	156,141	Linear
city	9	5	65,671	Linear
mit	11	5	70,776	Linear
jyu	12	5	22,795	Linear
hut	14	5	73,706	Linear
helsinki	15	5	61,168	Linear
wireless	15	5	34,028	Linear
yahoo	17	5	44,384	Linear
geizhals	18	5	24,003	Linear
bbc	23	5	27,554	Linear
gigantti	24	5	39,094	Linear
gsm-umts	14	5	73,706	Formula (3)
gsm-umts2	18	5	24,003	Formula (3)
gsm	17	3	44,384	Formula (3)

instance geu3 in which the tolerance was 10^{-4}. (A smaller tolerance 10^{-6} caused CPLEX to run 38 days of CPU time without establishing the final solution.) The relative optimality tolerance of 10^{-4} guarantees that the solution given by CPLEX is within 0.01% of the optimal solution value.

The computational results are presented in Tables 2 and 3. Table 2 contains (in the last column) the objective function values of the optimal solutions to the scalarizing function (2), to which the heuristic solutions are compared. In order to ease comparison, instead of reporting the solutions given by the two versions of the heuristic, the differences of the heuristic solutions to the optimal solution are given in the table. The differences in the objective function values (time, costs) are in parentheses, and they are calculated by subtracting the value of the optimal solution from the value of the heuristic solution. The differences in the value of the scalarizing function (2) between the heuristic and optimal solutions are also presented in Table 2. The average differences in the scalarizing function value are 0.0026 (heuristic1) and 0.0023 (heuristic2), which are very good results. It is also significant that in the cases of two test instances the solution given by both versions of the heuristic is the same as the optimal solution to the scalarizing function (2).

In Table 3, the CPU times used by the heuristics and CPLEX are given, as well as the number of improvement rounds done in each heuristic. The number of improvement rounds applied in the heuristics was small in all the problem instances tested. In one instance (geu3) four rounds were needed, whereas in the other cases at most two rounds were applied. Thus, if we had stopped the iteration in the heuristics after two rounds of improvements, the solutions would have remained the same except in the instance geu3 with the version

Table 2 Differences between heuristic solutions and optimal solutions and differences measured using the scalarizing function (2)

	heuristic1		heuristic2		Optimal solution
Instance	(time, costs)		(time, costs)		
geu1	(0.0, 0.0514)	0.0001	(0.0, 0.0514)	0.0001	(4.7778, 0.4721)
geu2	(0.0, 0.0047)	0.0000	(0.0, 0.0009)	0.0000	(5.0, 0.7664)
geu3	(0.1035, -0.0288)	0.0012	(0.1222, -0.0304)	0.0014	(4.4076, 1.0318)
city	(0.0, 0.0008)	0.0000	(0.0, 0.0008)	0.0000	(2.3224, 0.3441)
mit	(-0.0785, 0.0015)	0.0017	(-0.0785, 0.0015)	0.0017	(2.1137, 0.4712)
jyu	(0.0728, 0.0027)	0.0058	(0.0306, 0.0028)	0.0024	(0.7182, 0.1134)
hut	(0.0, 0.0)	0	(0.0, 0.0)	0	(4.3076, 0.3450)
helsinki	(0.0581, -0.0174)	0.0017	(0.0229, -0.0066)	0.0007	(1.7297, 0.4039)
wireless	(0.0472, -0.0045)	0.0025	(0.0472, -0.0045)	0.0025	(0.9641, 0.2247)
yahoo	(0.1538, -0.0258)	0.0062	(0.1538, -0.0258)	0.0062	(1.2594, 0.2934)
geizhals	(0.0253, 0.0016)	0.0019	(0.0253, 0.0016)	0.0019	(0.6848, 0.1570)
bbc	(0.0, 0.0243)	0.0001	(0.0, 0.0243)	0.0001	(0.8407, 0.1434)
gigantti	(0.0461, -0.0137)	0.0021	(0.0461, -0.0023)	0.0021	(1.1033, 0.2584)
gsm-umts	(0.0, 0.0)	0	(0.0, 0.0)	0	(8.8614, 2.2052)
gsm-umts2	(0.1033, -0.0016)	0.0078	(0.0559, 0.0191)	0.0069	(1.2099, 1.1936)
gsm	(0.0508, 0.0087)	0.0112	(0.0508, 0.0087)	0.0112	(6.6995, 0.5904)

Table 3 CPU-times and numbers of improvement rounds

	heuristic1		heuristic2		Optimal time(s)
Instance	time(s)	rounds	time(s)	rounds	
geu1	<0.01	1	0.01	1	0.05
geu2	0.02	1	0.01	1	0.08
geu3	0.04	4	0.04	2	79.64
city	0.01	1	0.01	1	0.07
mit	0.01	1	0.01	2	0.08
jyu	0.01	1	0.01	2	0.08
hut	0.01	0	0.01	0	0.07
helsinki	0.01	2	0.01	2	0.9
wireless	0.01	0	0.01	0	2.41
yahoo	0.01	1	0.01	1	1.13
geizhals	0.01	2	0.01	1	7.85
bbc	0.01	0	0.01	0	0.33
gigantti	0.02	1	0.02	1	3488.51
gsm-umts	0.01	0	0.01	0	0.07
gsm-umts2	0.01	2	0.01	1	54.72
gsm	0.01	0	0.01	0	0.42

heuristic1. The difference to the optimal solution in that case would have been $(0.1624, -0.0324)$ and the difference in the value of the scalarizing function 0.0019. In other words, this solution would have also been very good. However, the time needed by the heuristic would have remained the same. It should be noted that in five test instances, the initial solution was already very close to the optimal solution and the solution did not change in the improvement phase.

Both versions of the heuristic are fast: the CPU times used were typically around 0.01 seconds and always less than 0.04 s in all the test instances. Thus, the speed did not vary a lot between different instances. On the contrary, the times needed by CPLEX varied much, and in most instances the times were so long that they would not be acceptable. This is natural since exact solution methods, such as the branch-and-cut methods, are known to be slow. Yet, the solutions produced by the heuristics were very close to those of CPLEX, as mentioned above. Our heuristic may seem simple but in our problem setting it is a significant benefit. From the practical point of view, it is very important that we can say that we managed to achieve our goal of developing a computationally inexpensive method for solving the network connection selection problem without sacrificing the solution quality.

There was not much difference between the two versions of the heuristic, when we consider the quality of the solutions, the time used, and the number of improvement rounds taken. It is important to note that the ordering of the components done in the version heuristic2 did not increase the time used when compared to the other version. On the other hand, the solutions given by the different versions of the heuristic are very similar, even the average difference to the optimal solution is almost the same. Therefore, we cannot claim superiority over either of the versions.

We must point out that the computer used in the computational tests is quite efficient. Therefore, one might ask whether it would be at all possible to run

the heuristic in a mobile terminal. Actually, mobile terminals cannot yet run the heuristic as fast as the computer we used for the computational results, but the computational capacity of mobile terminals is growing fast. It is likely that in few years mobile terminals using, for example, an Intel XScale processor [8] will be able to run the heuristic as fast as our computer now.

6 Conclusions

In this paper, we have considered the network connection selection problem, which is a multiobjective scheduling problem, where a set of components is to be transferred using distinct network connections. The objective of the problem is to minimize both the time used for the transfers and the costs of the transfers.

We have presented a simple but efficient local search heuristic for the network connection selection problem. The idea of the heuristic is to improve an initial solution by simple moves called 1-exchanges. In a 1-exchange, a component is moved to be transferred using another connection if the solution quality improves. The core of our method is the novel idea of measuring the solution quality using a scalarizing function (found appropriate for producing a good compromise between the conflicting objectives in our previous study). The same scalarizing function is used also when forming the initial solution. The results obtained are very encouraging. The ideas of the heuristic are applicable also to other combinatorial multiobjective optimization problems and we can recommend using scalarizing functions as parts of heuristics in this field.

So far in our study, we have assumed that the time needed for the transmission is short. Then, we can assume that each connection has a fixed rate and a fixed price during the transmission. A topic of future research is to consider the problem with longer transmission times when the properties of the environment can change during the transmission.

Acknowledgements This research was supported by the GETA graduate school. The authors also wish to thank Dr. Marko M. Mäkelä and Dr. Heikki Maaranen for discussions on the heuristic and Mr. Aki Suihkonen for collecting data for the test instances.

References

1. Błażewicz, J.: Selected topics in scheduling theory. In: Martello, S., Laporte, G., Minoux, M., Ribeiro, C. (eds.) Surveys in Combinatorial Optimization, vol. 31 of Annals of Discrete Mathematics, p. 1–59. Elsevier, Amsterdam (1987)
2. Cormen, T.H., Leiserson, C.E., Rivest, R.L.: Introduction to Algorithms. The MIT Press and McGraw-Hill Book Company, New York (1990)
3. Ehrgott, M., Tenfelde-Podehl, D.: Computation of ideal and Nadir values and implications for their use in MCDM methods. Eur. J. Operat. Res. **151**(1), 119–139 (2003)
4. Graham, R.L., Lawler, E.L., Lenstra, J.K., Rinnooy Kan, A.H.G.: Optimization and approximation in deterministic sequencing and scheduling: a survey. In: Hammer, P.L., Johnson, E.L., Korte, B.H. (eds.) Discrete optimization II, vol. 5 of Annals of Discrete Mathematics, pp. 287–326. North-Holland Publishing Company, Amsterdam (1979)
5. Gustafsson, E., Jonsson, A.: Always best connected. IEEE Wirel. Commun. **10**(1), 49–55 (2003)
6. Hall, L.A.: Approximation algorithms for scheduling. In: Hochbaum, D.S. (ed.) Approximation Algorithms for NP-hard Problems. PWS Publishing Company, Boston (1997)

7. ILOG CPLEX 8.0 User's Manual. ILOG, (2002)
8. Intel XScale Microarchitecture, Technical Summary. Intel Corporation, (2000)
9. Jansen, K., Porkolab, L.: Improved approximation schemes for scheduling unrelated parallel machines. Math. Oper. Res. **26**(2), 324–338 (2001)
10. Korte, B., Vygen, J.: Combinatorial Optimization: Theory and Algorithms. Springer, Berlin Heidelberg New York (2002)
11. Miettinen, K.: Nonlinear Multiobjective Optimization. Kluwer, Dordrecht (1999)
12. Miettinen, K., Mäkelä, M.M., Kaario, K.: Experiments with classification-based scalarizing functions in interactive multiobjective optimization. Eur. J. Oper. Res. (2006) (to appear)
13. Nagar, A., Haddock, J., Heragu, S.: Multiple and bicriteria scheduling: a literature survey. Eur. J. Oper. Res. **81**(1), 88–104 (1995)
14. Setämaa-Kärkkäinen, A., Miettinen, K., Vuori, J.: Best compromise solution for a new multiobjective scheduling problem. Comput. Oper. Res. **33**(8), 2353–2368 (2006)
15. T'Kindt, V., Billaut, J.-C.: Multicriteria scheduling problems. In: Ehrgott, M., Gandibleux, X. (eds.) Multiple Criteria Optimization: State of the Art Annotated Bibliographic Surveys, pp. 445–491. Kluwer, Dordrecht (2002)
16. T'kindt, V., Billaut, J.-C.: Multicriteria Scheduling: Theory, Models and Algorithms. Springer, Berlin Heidelberg New York (2002)
17. Trick, M.A.: Scheduling multiple variable-speed machines. Oper. Res. **42**(2), 234–248 (1994)
18. Varshney, U., Jain R.: Issues in emerging 4G wireless networks. IEEE Comput. **34**(6), 94–96 (2001)