ISSN 1064-2307, Journal of Computer and Systems Sciences International, 2017, Vol. 56, No. 2, pp. 218–226. © Pleiades Publishing, Ltd., 2017.
Original Russian Text © V.A. Kostenko, 2017, published in Izvestiya Akademii Nauk, Teoriya i Sistemy Upravleniya, 2017, No. 2, pp. 48–56.

DISCRETE SYSTEMS

Combinatorial Optimization Algorithms Combining Greedy Strategies with a Limited Search Procedure

V. A. Kostenko

Moscow State University, Moscow, Russia

e-mail: kostmsu@gmail.com

Received September 8, 2016; in final form, October 4, 2016

Abstract—The proposed algorithms basically follow a greedy strategy, and a limited search procedure is invoked only at the steps at which the greedy choice cannot lead to the optimal solution. The principle of these algorithms design are illustrated using the problem of finding the maximum number of compatible jobs as an example. The results of applying the proposed algorithms for scheduling computations in distributed systems are described.

DOI: 10.1134/S1064230717020137

INTRODUCTION

The majority of combinatorial optimization problems are nondeterministically polynomial time hard (NP-hard). By a (general) optimization problem [1], we mean a general question that should be given an answer. The problem is determined by the following data:

—a list of all its parameters (input data),

—formulation of conditions that must be satisfied by the answer (solution) to the problem.

An instance of an optimization problem [1] is obtained from the general problem by assigning specific values to all its parameters.

A subproblem (or a particular problem) is an optimization problem [1] in which the same question as in the general problem is formulated on a subset of the set of problem instances; i.e., in the particular problem, some constraints are imposed on the values of the parameters.

Some subproblems of an NP-hard problem are often polynomially solvable. For example, the general problem of finding the maximum number of compatible jobs for a single-machine system that processes without interruptions a given set of jobs with individual prescribed processing intervals is NP-hard. However, the particular problem in which the processing time of each job is equal to the length of its prescribed processing interval can be exactly solved using a greedy algorithm of the complexity $O(n\log n)$ [2]. In this particular problem, the following constraint on the feasible values of the parameters is imposed: the length of the prescribed processing interval of each job is equal to its processing time.

The most widespread techniques for solving combinatorial optimization problems are as follows:

—the dynamic programming method [2–4];

—greedy strategies [2, 5];

—the branch-and-bound method [2, 5];

—iterative techniques [6], such as a random search [7], simulation annealing [8, 9], and genetic and evolutionary approaches [10–12];

—ant colony algorithms [13–16];

—algorithms based on finding the maximum flow in a transportation network [17–20].

In applications, it is typically required to develop and algorithm for solving a particular problem, but this algorithm must provide a prescribed accuracy and its complexity must not exceed a prescribed limit. In other words, the algorithm is not required to find an optimal solution; rather, it should find an acceptably accurate solution in an acceptable amount of time.

The main difficulty in using iterative algorithms is the adjustment of their parameters for the particular problem to guarantee a required compromise between the accuracy and the computational complexity of the algorithm [21]. Presently, there are no theoretical results for resolving this difficulty. In applications,

the values of the algorithm parameters are fitted experimentally. In addition, the accuracy and computational complexity of such algorithms can be estimated only statistically. This implies that there can be instances of the particular problem for which the prescribed accuracy requirements and constraints on the computational complexity are not satisfied.

The main difficulty in using the dynamic programming method and the bound-and-branch technique is the fact that the upper bound on the computational complexity of the majority of general combinatorial optimization problem is nonpolynomial. This is due to the features of these methods. For example, in the case of the bound-and-branch technique, the properties of the lower bound function and the function used to choose the best solution do not guarantee that at least one subset different from the basis one will be cut off. For this reason, the algorithm applied to an instance of the problem can in the worst case examine all solutions.

Greedy algorithms are based on the optimality property for subproblems. If the optimal solution of a problem includes the optimal solution of its subproblems, then this problem is said to possess the optimality property for subproblems. Greedy algorithms make a locally optimal choice at each step (solve a subproblem) assuming that finally the optimal solution of the whole problem will be obtained. Greedy algorithms have a low computational complexity, but the domain where they can be used is restricted to the particular problems that possess the optimality property for subproblems. It is not trivial to prove that a particular problem possesses the optimality property for subproblems; most often, this property is proved using the approach discussed in [2]. In greedy algorithms, constraints on the feasibility of solution can be easily verified, which is especially important for practical problems. For example, in scheduling data exchange via a bus with centralized control, there can be from three to seven additional constraints on the feasibility of the schedule [16] depending on the bus controller compared with the classical problem of finding the maximum number of compatible jobs.

Ant colony algorithms can automatically adjust themselves to the instance of the problem to be solved in the process of their operation. To use ant colony algorithms, the problem to be solved must be reduced to a problem of finding a route that has certain properties in a graph. In addition, as in iterative algorithms, there is the problem of adjusting the algorithm parameters to a particular problem.

Algorithms based on finding the maximum flow in a transportation networks have a pseudopolynomial complexity. However, the field of their application is limited to the class of problems that can be reduced to finding the maximum flow in a transportation network. Moreover, in the design of such algorithms, there is the problem of taking into account additional constraints on the feasibility of the solution.

The algorithms proposed in this paper, which combine greedy strategies and limited search, are based on the following basic principles:

—at each step of the algorithm operation, a locally optimal choice according to a greedy strategy is made;

—at each step, it is checked whether the greedy choice closes the way to the optimal solution;

—if the greedy choice closes the way to the optimal solution, then a limited search procedure is invoked.

In Section 1, the proposed method for designing algorithms combining greedy strategies and limited search is described, two schemes for designing such algorithms are presented, and their properties are justified. In Sections 2 and 3, respectively, the applications of this method for designing algorithms for scheduling data exchange via a channel with centralized control and for planning computations in data processing centers are described.

1. A METHOD FOR DESIGNING ALGORITHMS COMBINING GREEDY STRATEGIES AND A LIMITED SEARCH PROCEDURE

We will consider the method for designing algorithms combining greedy strategies and a limited search procedure using the problem of finding the maximum number of compatible jobs as an example. In the general case, this problem can be formulated as follows. Let a set of jobs and a set of resources on which these jobs can be executed be given. Each job has a set of requirements for the quality of its execution. Each resource has a set of characteristics. Moreover, a set of constraints on the feasible assignment of jobs to resources is specified. The parameters of these constraints are the characteristics of the resources and the requirements for the quality of the execution of jobs. It is required to assign the maximum number of jobs to the given resources while satisfying the constraints on the feasible assignment of jobs. If we deal with a single-machine system, then we should only find a place for each job in the execution schedule.

The principle of designing algorithms combining greedy strategies and a limited search procedure is as follows. At each step, the algorithm selects a job from the set of not scheduled jobs and a place where it can be scheduled using a greedy criterion. In the first approach to designing the algorithm, the limited search procedure is invoked if the selected job cannot be scheduled at the next step. In the second approach, the limited search procedure is invoked if, after the tentative scheduling of a job at the next step, jobs that cannot be scheduled appear in the set of unscheduled jobs. For the limited search procedure, the maximal search depth that sets the maximum number of jobs that can be involved in the search is specified.

Below, we outline the algorithms corresponding to these two approaches.

Schemee of Algorithm 1

1. According to the greedy criterion, select a job from the set of unscheduled jobs.

2. According to the greedy criterion, select a resource (a place in the schedule) for this job:

—if no such resource is found, go to Step 3;

—if such a resource is found, then assign the job to it, remove this job from the set of unscheduled jobs and, if this set in not empty, go to Step 1; otherwise, stop the algorithm.

3. Call the limited search (or heuristic) procedure:

—if this procedure is completed successfully, then save the assignment, remove this job from the set of unscheduled jobs and, if this set in not empty, go to Step 1; otherwise, stop the algorithm;

—if this procedure fails, then remove this job from the set of unscheduled jobs and, if this set in not empty, go to Step 1; otherwise, stop the algorithm.

Scheme of Algorithm 2

1. According to the greedy criterion, select a job from the set of unscheduled jobs.

2. According to the greedy criterion, select a resource for this job assignment (at least one resource always exists), make a tentative assignment, and check the optimality of the greedy choice:

—if the set of unscheduled jobs contains jobs that cannot be scheduled after scheduling the chosen job, then cancel the tentative assignment and go to Step 3;

—if the set of unscheduled jobs contains no jobs that cannot be scheduled after scheduling the chosen job, then remove the chosen job from this set and, if this set is not empty, go to Step 1.

3. Call the limited search (or heuristic) procedure:

—if this procedure is completed successfully, then save the assignment, remove this job from the set of unscheduled jobs and, if this set in not empty, go to Step 1; otherwise, stop the algorithm;

—if this procedure fails, then remove the current job from the set of unscheduled jobs and, if this set in not empty, go to Step 1; otherwise, stop the algorithm.

Consider properties of the proposed schemes of algorithms combining greedy strategies and the limited search procedure. A function is said to be separable [22] into f_1 and f_2 if it can be represented in the form

$$f(x, y) = f_1(x, f_2(y)).$$

A function is said to be decomposable [22] into f_1 and f_2 if it is separable into f_1 and f_2 and the function f_1 is monotonically nondecreasing with respect to the last argument.

The optimality theorem for decomposable functions [22] claims that

$$\min_{(x,y)} f_1(x, f_2(y)) = \min_{(x)} f_1(x, \min_{(y)} f_2(y)).$$

Proposition. Ifthe objective function $f(x_1, x_2, \ldots x_n)$ of the problem is decomposable into $f_1, f_2, \ldots f_n$; the index of the function is used as the greedy criterion for the choice of the next variable (job) to be optimized; the global minimum of the corresponding function is used as the greedy criterion for the choice of the value of the variable (the place to assign the job to; and there exists a solution containing all jobs from the initially specified set; then Algorithms 1 and 2 produce the optimal solution and the limited search procedure will never be called.

The validity of this proposition follows from the optimality theorem for decomposable functions and the existence of a solution containing all the jobs. Since the optimality theorem implies that all jobs can be scheduled using the greedy strategy and there are no jobs that cannot be scheduled in the final solution,

then the check of the need to call the limited search procedure will always be negative. In other words, the solution will be obtained using only the greedy strategy, and the complexity of obtaining the solution will be equal to the lower bound on the computational complexity of the algorithm.

We illustrate this fact using the particular problem of finding the maximum number of compatible jobs in a single machine system processing without interruptions the given set of jobs with individual prescribed processing intervals for which the processing time of each job equals the length of its prescribed processing interval.

The general optimization problem can be formulated as follows:

The parameters of the problem are the set of jobs $J = \{j\}_{j=1}^{N_{\text{given}}}$ that must be executed in the system; here N_{given} is the number of the given jobs; the set processing times of the jobs $\{t_j > 0\}_{j=1}^{N_{\text{given}}}$, the set of prescribed processing intervals $\{[s_j, f_j)\}_{j=1}^{N_{\text{given}}}$ such that $\forall j$: $f_j - s_j \geq t_j$.

The schedule of the job processing is an ordered (by the starting time of the job processing) set $H = \left\{j_k, s_j^*\right\}_{k=1}^{N_H}$, $j \in J$. Here k is the ordinal number of the job j in the schedule, s_j^* is the starting time of the job j processing in the schedule H, and $f_j^* = s_j^* + t_j$ is the completion time of processing the job j. The set of feasible schedules H'^* is defined by the following set of constraints:

$$g_1\colon (\forall j \in H) \Rightarrow \left((s_j^* \geq s_j) \wedge (f_j^* \leq f_j)\right),$$

$$g_2\colon (\forall j \in H) \Rightarrow \left(f_j^* - s_j^* = t_j\right),$$

$$g_3\colon (\forall (j, l) \in H, \quad j \neq l) \Rightarrow \left(((s_j^* < s_l^*) \vee (s_j^* \geq f_l^*)) \wedge ((f_j^* \leq s_l^*) \vee (f_j^* > f_l^*))\right).$$

The constraints g_1, g_2, and g_3 imply, respectively, that

(1) the interval of each job processing is within its prescribed processing interval;

(2) no interruptions of processing are allowed;

(3) the processing intervals of jobs do not overlap.

A solution of the problem is defined as a feasible job processing schedule containing the maximum number of jobs from the given set $J = \{j\}_{j=1}^{N_{\text{given}}}$:

$$\max_{H \in H^*} |H|.$$

In scheduling theory, this problem is known as the problem of finding the maximum number of compatible jobs; it is known to be NP-hard.

The particular problem of the general optimization problem of the maximum number of compatible jobs for which the processing time of each job is equal to the length of its prescribed processing interval has the same parameters and solution but has additional constraints on the relations between the processing time of the jobs and their prescribed processing intervals: $\forall\ j$: $t_j = f_j - s_j$.

Suppose that the right endpoint of the job's prescribed processing interval (deadline) is used as the greedy criterion of the job choice, and let the greedy rule of scheduling the next job be to put it at the end of the list of jobs scheduled at the preceding steps. In the algorithm designed according to scheme 1, each job will always be scheduled. This follows from the fact the there exists a solution containing all jobs from the given set and from the constraint on the feasible values of the problem parameters ($\forall\ j$: $t_j = f_j - s_j$). These conditions imply that the jobs' prescribed processing intervals do not overlap. Therefore, the limited search procedure will never be called in the solution of this particular problem.

In the algorithm designed according to scheme 2 with the same greedy criteria, the limited search procedure will never be called because after the tentative scheduling of a job, each job in the remaining set can be scheduled. This is implied by the fact that the jobs' prescribed processing intervals do not overlap.

Scheme 1 has a lower computational complexity for the problems in which the majority of jobs can be optimally scheduled using the greedy criterion. If the greedy criteria used in an algorithm designed for the particular problem make it possible to obtain the optimal solution, then the computational complexity of this algorithm applied for the particular problem is equal to the computational complexity of the algorithm based on scheme 2. If the greedy criteria are badly suitable for the particular problem, then the accuracy

of the algorithm designed according to scheme 1 will be lower than that of the algorithm designed according to scheme 2.

2. APPLICATION OF THE PROPOSED APPROACH FOR DESIGNING COMPUTATION SCHEDULING ALGORITHMS

In this section, we consider algorithms designed according to scheme 1 for resource distribution in data processing centers [23, 24] and an algorithm designed according to scheme 2 for scheduling data exchanges via a bus with centralized control [25].

2.1. An Algorithm for Scheduling Data Exchanges via a Bus with Centralized Control

Suppose that a limited set of terminal devices are connected to a bus; these devices are sources or receivers of data passed through the bus. One of these devices is the bus controller that controls the data exchange according to a preliminarily established schedule and controls the states of the other terminal devices. These terminal devices only execute the commands sent to them by the controller. The data exchange is performed asynchronously by the command–response principle. The data are transmitted in the form of messages that can consist of command words, data words, and response words [26].

The problem of designing the data exchange via a bus with centralized control can be considered as a problem of finding the maximum number of compatible jobs for a single-machine system that executes a given set of jobs with individually prescribed processing intervals without interruptions (see Section 2); it is assumed that in this problem, in addition to the constraints g_1, g_2, and g_3, the following constraints are imposed on the feasibility of the schedule:

—g_4: the number of jobs in every chain does not exceed a prescribed value;

—g_5: the total processing time of all the jobs in every chain does not exceed the prescribed value;

—g_6: the time interval between the adjacent chains must be not shorter than the prescribed value.

These constraints are due to the features of the bus controller implementation. A chain of jobs is a sequence of jobs that continuously follow each other; i.e., $TaskChain = \left\{\left[s_i^*, f_i^* \right)\right\}_{i=0}^{k}$, $i = \overline{1, k}$: $s_i^* = f_{i-1}^*$.

The data exchange scheduling algorithm [25] is designed according to scheme 2. As the greedy criterion for selecting the next job from the list of still unscheduled jobs, we use $f_i' = s_i + t_i$, which is the earliest possible job completion time. At each step, the optimality of the greedy choice is checked. It consists of a tentative scheduling of the next job in the list and then checking that no jobs that cannot be scheduled appear in the list of remaining jobs.

At each step of the algorithm, the selected job is scheduled without violating the schedule feasibility and the nearest bus cycle beginning at which the next job can be executed is calculated. The nearest bus cycle is found taking into account the additional constraints on the schedule feasibility and on the execution time of the scheduled job. Next, the prescribed processing intervals of the unscheduled jobs are updated, and, at the next step, a job is chosen according to the greedy criterion.

The limited search procedure forms a list of jobs whose prescribed processing intervals overlap with the processing interval of the job considered at the current step of the algorithm. The size of this list is a parameter of the algorithm—it determines the search depth. The search algorithm produces a set of compatible jobs of the maximum length. If there are more than one such sets, the one with the earliest bus release time is chosen, and the first job in this set is scheduled at the current step.

The selection of a job J with the minimum possible completion time makes it possible to avoid the search when the additional constraints $\{g_i, i \in (4,5,6)\}$ are checked. Indeed, if the job selected according to this rule violates these constraints, then the set of unscheduled jobs cannot contain a job that can be scheduled without violating these constraints.

It turned out that, for the particular data exchange scheduling problems for on board aviation systems, more than 90% of all jobs are scheduled using the greedy criterion, i.e., no limited search procedure is called for them. For these problems, the complexity of the algorithm is on average 10% of the upper bound on the complexity of the algorithm. The upper bound on the complexity of the algorithm is $O(N_j^2 + N_j F(L))$, where $F(L)$ is the complexity of the limited search of depth L. In [25], various algorithms for such a search were proposed and investigated. If the processing time of every job in the given set equals the length of its prescribed processing interval, then the algorithm produces an exact schedule.

2.2. Resource Distribution Algorithms in Data Processing Centers

We consider data processing centers (DCs) providing Infrastructure-as-a-Service (IaaS). The task of scheduling computations in a DC is to map requests for creating virtual machines and virtual data storage systems connected by virtual data exchange channels to computing servers and data storage servers and in designing routes for the virtual data exchange in the data exchange network of the DC. If the DC operates in the IaaS mode, the customer and the service provider should be able to set a Service Level Agreement (SLA) for all types of resources—data storage systems, computing, and network resources. The computing resources, data storage systems, and network resources should be considered as allocatable resources and they should be allocated consistently in the sense of satisfying the service level agreement. The resulting mappings of the virtual resources to the DC physical resources must meet the service level agreement. Below we will use the mathematical statement of the problem described in [23].

The model of the DC physical resources is described by the labeled graph $H = (P \cup M \cup K, L)$, where P is the set of computing nodes, M is the set of data storages, K is the set of commutation elements of the DC communication network, and L is the set of physical data transmission channels. On the sets P, M, K, and L, vector functions of a scalar argument that determine, respectively, the characteristics of the computing nodes, data storages, commutation elements, and data transmission channels are defined. In what follows, they are called the labeling functions of the graph H. For example, for a computing node, the number of cores and the amount of main and disk memory can be specified.

Every resource request is described by a labeled graph $G = (W \cup S, E)$, where W is the set of virtual machines, S is the set of virtual data storage systems (storage elements), and E is the set of virtual data transmission channels. On the sets W, S, and E, vector functions of a scalar argument specifying the characteristics of the requested virtual element (the required quality of service) are defined.

By the resource request assignment, we mean the mapping

$$A\colon G \to H = \{W \to P,\ S \to M,\ E \to \{K,\ L\}\}.$$

There are three types of relations between the request characteristics and the corresponding physical resource characteristics. We denote by x the characteristics of a request element and by y the corresponding characteristic of the physical resource. Then, these relations can be written as follows.

1. The physical resource capacity must not be overloaded:

$$\sum_{i \in R_j} x_i \le y_j;$$

here R_j is the set of requests assigned to the physical resource j.

2. The types of the physical and virtual resources must match: $x = y$.

3. The physical resource must have the requested characteristics: $x \le y$.

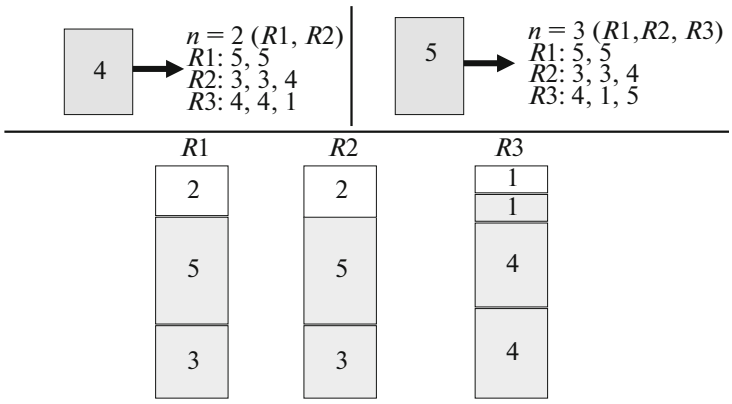
A mapping A is called correct if relations 1–3 hold for all physical resources and all their characteristics.

The residual graph of available resources H_{res} is defined as the graph H in which the values of the functions are redefined such that they satisfy relation 1. The value of each characteristic of the physical resource is decreased by the sum of the values of the corresponding characteristic of the virtual resources mapped to this physical resource.

The input of the algorithm is the residual graph of available resources H_{res} and the set of resource requests $\{G_i\}$. The set $\{G_i\}$, in addition to the newly arrived requests, may include running virtual resources that are allowed to migrate. If $\{G_i\}$ contains virtual resources, then, for the elements of the graph H_{res} associated with virtual resources, the values of the labeling functions of H are redefined (increased) for the characteristics that must satisfy relation 1.

It is required to satisfy the maximum number of requests from the set $\{G_i\}$ such that the mappings $\{A_i\colon G_i \to H, \quad i = \overline{1, n}\}$ are correct. The virtual resources associated with the physical ones and included in the set $\{G_i\}$ must not be removed.

The output of the algorithm is a set of mappings of the resource requests to physical resources $\{A_i\colon G_i \to H, \quad i = \overline{1, n}\}$ and a set of replications $\{R_i\}$, $i = 0,1, \ldots$, of storage elements.



Example of an execution of a limited search procedure.

The most complete survey of the available algorithms for scheduling computations in DCs can be found in [27]. In practice, greedy algorithms are mainly used because they have low computational complexity. A drawback of the greedy algorithms is that their accuracy can be significantly different on different particular problems. In [23, 24], DC scheduling algorithms based on a combination of greedy strategies and limited search were proposed. They are designed according to scheme 1 described above, and they allow one to specify a balance between the computational complexity and the algorithm accuracy. In [24], an algorithm with the sequential scheduling of requests was proposed (according to a greedy criterion, a request to be scheduled is chosen, and then physical resources are allocated to all its elements). This algorithm ensures a compact mapping of requests to the DC physical resources with respect to the total length of the routes for the request's virtual channels. In [23], an algorithm with the sequential consistent scheduling of different types of resources was proposed (first, the virtual machines of all requests are mapped to physical resources, then the storage elements are mapped, and finally routes for the virtual channels are constructed). This algorithm is more accurate than the first algorithm in the case when the critical resources are computing servers and data storage servers.

This algorithm involves three steps.

Step 1. Mapping the virtual machines to the computing servers.

Step 2. Mapping the virtual data storage systems of the data storage servers.

Step 3. Constructing virtual channel routes in the DC data exchange network for the mappings of virtual machines and virtual data storage systems to physical resources.

The outline of the procedure of executing Steps 1 and 2 is as follows (see [23]).

1. Select the next request G_i from the set of resource requests $\{G_i\}$ using the greedy criterion K_G.

2. Select the next element e (a virtual machine $e \in W$ ($W \in G_i$) or a storage element $e \in S, S \in G_i$) using the greedy criterion K_e.

3. Form the set of physical resources P_h ($P_h \subseteq P$ or $P_h \subseteq M$, respectively) to which the selected element e can be mapped, i.e., the set of resources for which the mapping of e to them is correct.

3.1. If the set P_h is empty, then call the limited search procedure. If the procedure fails (returns false), then the request G_i cannot be mapped. Remove the earlier mapped elements of G_i and redefine the values of the physical resource characteristics, which must satisfy relation 1. If the set $\{G_i\}$ is not empty, then go to Step 1; otherwise, stop the algorithm.

3.2. If the set P_h is not empty, then select a physical element $p_h \in P_h$ for mapping using the greedy criterion K_{ph}, map the request element e to the physical element p_h, and redefine the values of its characteristic. If there are unmapped elements of the request, then go to Step 2. If the set $\{G_i\}$ is not empty, then go to Step 1; otherwise, stop the algorithm.

The principle of the limited search procedure is illustrated in the figure.

The search depth is limited by the given number n that sets the maximum number of physical elements among which reassignment (remapping) may be made (e.g., for $n = 2$, the mapped elements can be removed and remapped not more than from two physical servers). The left part of the figure illustrates the situation in which a virtual machine requesting four cores cannot be mapped to any of the physical servers even though the total number of free cores is sufficient for mapping. The virtual machines mapped to the servers $R1$, $R2$, and $R3$ are shaded, and the number of occupied cores is shown; the unshaded box shows the number of free cores on each server. If we remap the virtual machines assigned to the servers $R1$ and $R2$, the fragmentation is decreased, and the virtual machine can be mapped; i.e., in this case, the search depth $n = 2$ is sufficient. The right part of the figure illustrates the situation in which the virtual machine requests five cores. In this case, the virtual machine cannot be mapped if the search depth is two. However, if the search depth is $n = 3$, then the virtual machine can be assigned. Therefore, the accuracy can be improved by increasing the search depth; however, the computational complexity of the algorithm also increases. For a particular problem, the computational complexity of the algorithm can be reduced by choosing appropriate greedy criteria K_G, K_e, and K_{ph}, which reduces the number of calls of the limited search procedure.

CONCLUSIONS

The proposed method for designing algorithms that combine greedy strategies and a limited search procedure makes it possible to specify the balance between the computational complexity and the algorithm accuracy by choosing the maximal allowed depth of the search. The method can be fine-tuned for a particular problem by choosing the appropriate greedy criteria. The use of the proposed method for designing algorithms for scheduling computations in real-time systems and data processing centers showed that, for the majority of problems, more than 90% of all jobs are scheduled using the greedy criterion; i.e., no limited search procedure is called for them; for these algorithms, the run time insignificantly exceeds the run time of the corresponding greedy algorithm, but they can find an optimal solution more often than purely greedy algorithms.

ACKNOWLEDGMENTS

This work was supported by the Russian Foundation for Basic Research, project no. 16-07-01237.

REFERENCES

1. M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness,* Series of Books in the Mathematical Sciences (W. H. Freeman, New York, 1979).
2. T. Cormen, Ch. Leiserson, and R. Rivest, *Introduction to Algorithms* (MIT, Cambridge, MA, 2001).
3. M. Minoux, *Programmation mathématique: Théorie et algorithmes* (Tec & Doc Lavoisier, 2007; Nauka, Moscow, 1990) [in French].
4. R. Bellman, *Dynamic Programming* (Princeton Univ. Press, Princeton, NJ, 1957).
5. *Computer and Job-Shop Scheduling Theory,* Ed. by E. G. Coffman (Wiley, New York, 1976; Nauka, Moscow, 1984).
6. V. A. Kostenko, "Scheduling algorithms for real-time computing systems admitting simulation models," Program. Comput. Software **39**, 255–267 (2013).
7. L. A. Rastrigin, *Statistical Search Methods* (Nauka, Moscow, 1968) [in Russian].
8. A. V. Kalashnikov and V. A. Kostenko, "A parallel algorithm of simulated annealing for multiprocessor scheduling," J. Comput. Syst. Sci. Int. **47**, 455 (2008).
9. D. A. Zorin and V. A. Kostenko, "Algorithm for synthesis of real-time systems under reliability constraints," J. Comput. Syst. Sci. Int. **51**, 410 (2012).
10. J. N. Holland, *Adaptation in Natural and Artificial Systems* (Univ. of Michigan Press, Ann Arbor, Michigan, 1975).
11. Yu. A. Skobtsov, *Fundamentals of Evolutional Computations* (Donetsk. Nath. Tech. Univ., Donetsk, 2008) [in Russian].
12. D. I. Batishchev, E. D. Gudman, I. P. Norenkov, and M. Kh. Prilutskii, "Combining heuristics method for solving combinatorial problems of resouce ordering and allocation," Inform. Tekhnol., No. 2, 29–32 (1997).
13. M. Dorigo, "Optimization, learning and natural algorithms," PhD Thesis (Politech. di Milano, Milano, 1992).

14. V. A. Kostenko and A. V. Plakunov, "An algorithm for constructing single machine schedules Based on ant colony approach," J. Comput. Syst. Sci. Int. **52**, 928 (2013).
15. S. D. Shtovba, "Ant algorithms: theory and application," Programmirovanie, No. 4, 1–15 (2005).
16. C. Blum and M. Sampels, "Ant colony optimization for FOP shop scheduling: a case study on different pheromone representation," in *Proceedings of the Congress on Evolutionary Computation* (IEEE Comput. Soc. Press, Los Alamitos, CA, 2002), Vol. 2, pp. 1558–1563.
17. A. Federgruen and H. Groenevelt, "Preemptive scheduling of uniform machines by ordinary network flow technique," Manage. Sci. **32** (3) (1986).
18. T. Gonzales and S. Sanhi, "Preemptive scheduling of uniform processor systems," Assoc. Comput. Machin. **25** (1) (1978).
19. D. S. Guz and M. G. Furugyan, "Computation scheduling in multiprocessor real-time automatic control systems with constrained processor memory," Autom. Remote Control **66**, 295 (2005).
20. E. O. Kosorukov and M. G. Furugyan, "Some algorithms for resource allocation in multiprocessor systems," Moscow Univ. Comput. Math. Cybernet. **33**, 202 (2009).
21. V. A. Kostenko and A. V. Frolov, "Self-learning genetic algorithm," J. Comput. Syst. Sci. Int. **54**, 525 (2015).
22. N. M. Novikova, *Principles of Optimization, Course of Lectures* (Mosk. Gos. Univ., Moscow, 1998) [in Russian].
23. P. M. Vdovin and V. A. Kostenko, "Algorithm for resource allocation in data centers with independent schedulers for different types of resources," J. Comput. Syst. Sci. Int. **53**, 854 (2014).
24. I. A. Zotov and V. A. Kostenko, "Resource allocation algorithm in data centers with a unified scheduler for different types of resources," J. Comput. Syst. Sci. Int. **54**, 59 (2015).
25. V. A. Kostenko and E. S. Gur'yanov, "An algorithm for scheduling exchanges over a bus with centralized control and an analysis of its efficiency," Program. Comput. Sci. **31**, 340–346 (2005).
26. GOST (State Standard) 26765.52-87: Main serial interface of electronic modules system.
27. Jiangtao Zhang, Hejiao Huang, and Xuan Wang, "Resource provision algorithms in cloud computing: A survey," Network Comput. Appl. C **64**, 23–42 (2016).

Translated by A. Klimontovich