BULLETIN OF THE POLISH ACADEMY OF SCIENCES
TECHNICAL SCIENCES
Vol. 57, No. 1, 2009

The Knapsack-Lightening problem and its application to scheduling HRT tasks

J.R. NAWROCKI*, W. COMPLAK, J. BŁAŻEWICZ, S. KOPCZYŃSKA, and M. MAĆKOWIAK

Institute of Computing Sci., Poznan University of Technology, 60-965 Poznan, Poland

Abstract. In hard real-time systems timeliness is as important as functional correctness. Such systems contain so called hard real-time tasks (HRT tasks) which must be finished by a given deadline. One of the methods of scheduling of HRT tasks is periodic loading introduced by Schweitzer, Dror, and Trudeau. The paper presents an extension to that method which allows for deterministic utilization of cache memory in hard real-time systems. It is based on a new version of the Knapsack problem named Knapsack-Lightening. In the paper the Knapsack-Lightening problem is defined, its complexity is analyzed, and an exact algorithm along with two heuristics are presented. Moreover the application of the Knapsack-Lightening problem to scheduling HRT tasks is described.

Key words: algorithms, computational complexity, knapsack problem, greedy algorithms, branch and bound scheduling, real-time systems, cache memory, periodic loading.

1. Introduction

Assume you are in charge of an expedition consisting of a number of people with knapsacks. Your expedition approaches an old suspension bridge over a precipice. It is rather in a bad condition. To minimize risk of collapsing the bridge, the knapsacks will have to be carried over one by one. Still you are not sure it will be enough. Some knapsacks are really heavy. Then a few local inhabitants appear and they make a proposal: You can leave some goods of your knapsacks on one side of the bridge and – if you pay – they will carry them over to the other side. What is original, the local inhabitants want you to pay not by weight of the products they will carry over, but by product type. For instance, all the water you have in all the knapsacks they are ready to transport just for $2 (they are very clever: they know there is a spring on the other side of the bridge, so they will not have to carry it over at all). Other product types are bread, meat etc. A part of the deal is that you are not allowed to move products between the knapsacks.

You are going to accept the offer. But your budget is limited, so you have to make a wise selection of product types to be transported by the local inhabitants. You know weight of each knapsack and contents of each product type in each knapsack (e.g. in kilograms). To minimize risk of collapsing the bridge you are going to minimize maximum knapsack weight.

To describe the problem more formally the following symbols will be used (**len** s denotes length of sequence s):

p: Price list (sequence of integers greater than 0); p(t) is price to be paid to the local inhabitants for transportation of products of type t (t = 1, 2, .., **len** p).

M: Money (an integer greater than 0); maximum amount of money that can be spent on the transport operation.

d: Decision sequence (sequence of numbers 0 or 1); there is a decision about each item from the price list p (i.e. **len** d = **len** p) and d(t) = 1 means that products of type t will be carried over by the local inhabitants.

D(n): Set of all possible decision sequences of length n (for instance D(2)= {[00], [01], [10], [11]).

w: Weight (sequence of integers greater than 0); w(k) is weight of knapsack k (k = 1, 2, .., **len** w).

c: Contents (a matrix of integers not less than 0); rows correspond to knapsacks and columns to product types; knapsack k contains c(k, t) kilograms of product of type t (t = 1, 2, .., **len** p; k = 1, 2 .., **len** w).

Let H (H stands for Heaviest) be a function assigning to each decision sequence d weight of the heaviest knapsack, i.e. domain of H is D(**len** p) and its values are from **N**. H can be defined as follows:

$$H(d) = \max_{k\in\{1,\ldots,\mathbf{len}w\}} \left\{ w(k) - \sum_{t\in\{1,\ldots,\mathbf{len}p\}} c(k,t) \right\}. \tag{1}$$

The Knapsacks Lightening problem (KL) can be stated in the following way. Given money M, price list p, knapsacks initial weights w, and contents matrix c:

$$\text{minimize} H(d) \text{over} d \in D(\mathbf{len}p), \tag{2}$$

$$\text{subject to} \sum_{t\in\{1,\ldots,\mathbf{len}p\}} p(t)\cdot d(t) \leq M. \tag{3}$$

Solution to the Knapsacks Lightening problem will be denoted as z_{KL}.

*e-mail: Jerzy.Nawrocki@put.poznan.pl

The aim of the paper is to investigate the Knapsacks Lightening problem. In Sec. 2 it is shown that Knapsacks Lightening is strongly NP-hard. A polynomial transformation is described from the Max-Min Knapsack problem [1, 2] to Knapsacks Lightening. The transformation indicates that Knapsacks Lightening is, in some sense, a generalization of the Max-Min Knapsack problem. Two greedy heuristics are presented in Sec. 3. One of them is used by an exact algorithm discussed in Sec. 4 (it is based on the branch-and-bound strategy). In Sec. 5 computational experiments are described that aimed at evaluation of the proposed greedy heuristics. In those experiments we used the exact algorithm presented in Sec. 4. Section 6 presents application of the Knapsacks Lightening problem to scheduling of hard real-time tasks for computers with cache memory.

2. Complexity of Knapsacks lightening

Complexity of the Knapsacks Lightening problem is described by Theorem 1.

Theorem 1. The Knapsacks Lightening problem is strongly NP-hard.

Proof. A polynomial transformation from the Max-Min Knapsack problem (which is known to be strongly NP-hard [1, 2]) to Knapsacks Lightening will be shown. The former will be denoted as MMK, the latter as KL.

Decision version of KL, KL-D, contains all the parameters listed in the previous section and additional parameter h which is an integer not less than zero. The question is if there exists $d \in$D(**len** p) such that

$$H(d) \leq h$$

and (3) holds.

The decision version of MMK, MMK-D, has the following input [1, 2]: knapsack capacity C, sequence of item sizes s (each size $s(i)$ is an integer greater than 0), number of usage scenarios U (U is greater than 0), value matrix $v(u, i)$ describing value of item i in usage scenario u ($v(u,i) > 0$ for $i = 1, ..., \mathbf{len}s$; $u = 1, ..., U$) and an integer l greater than 0. Let x be a selection sequence (**len** $x =$ **len**s). Each $x(i) \in \{0, 1\}$ and $x(i) = 1$ means that item i is to be packed to the knapsack. Let also $X(n)$ denote set of all selection sequences of length n. Moreover, let L (L for Lowest) be a function assigning to each selection sequence x the lowest knapsack value over all the scenarios, i.e.

$$L(x) = \min_{u \in \{1,...,U\}} \Big\{ \sum_{i \in \{1,...,\mathbf{len}s\}} v(u,i) \cdot x(i) \Big\}. \quad (4)$$

In MMK-D the question is if there exists $x \in X$(**len** s) such that

$$L(x) \geq l, \quad (5)$$

and

$$\sum_{(i \in \{1,...,\mathbf{len}\ \mathbf{s}\}} s(i) \cdot x(i) \leq C. \quad (6)$$

One can solve MMK-D by transforming it to KL-D in the following way:

$M :=$ C

$p(t) :=$ $s(t)$ for $t = 1, ..., \mathbf{len}s$ (product types of KL correspond to items in MMK);

$w(k) :=$ V, where $V = \max_{u \in \{1,...,U\}} \{ \sum_{i \in \{1,...,\mathbf{len}s\}} v(u,i) \}$ for $k = 1$, ..., U (KL knapsacks correspond to usage scenarios of MMK and all the knapsacks in KL have the same initial weight equal to maximum possible value of the knapsack in MMK);

$c(k,t) :=$ $v(k,t)$ for $k = 1, ..., U$; $t = 1, ..., \mathbf{len}s$ (contents of product type t in knapsack k equals value item t in scenario k);

$h :=$ $V - l$.

That transformation is polynomial and parameters M, p, w, c, h are bounded by the polynomial of the MMK-D problem size.

Now it will be shown that

$$\exists_{d \in D(\mathbf{len}\ p)} H(d) \leq h \wedge \sum_{t \in \{1,...,\mathbf{len}\ p\}} p(t) \cdot d(t) \leq M, \quad (7)$$

if and only if

$$\exists_{x \in X(\mathbf{len}\ s)} L(x) \geq l \wedge \sum_{i \in \{1,...,\mathbf{len}\ s\}} s(i) \cdot x(i) \leq C. \quad (8)$$

Assume

$$x(i) = d(i). \quad (9)$$

From the transformation it follows that

$$\mathbf{len}\ p = \mathbf{len}\ s \wedge \mathbf{len}\ w = U \wedge w(k) = V \wedge c(k,t) = v(k,t).$$

Thus, (1) can be rewritten to the following form:

$$H(d) = \max_{k \in \{1,...,U\}} \{ V - \sum_{t \in \{1,...,\mathbf{len}\ s\}} v(k,t) \cdot d(t) \},$$

where $d \in D(\mathbf{len}\ p)$.

Since V does not depend on k and $\max_k \{ -f(k) \} = -\min_k \{ f(k) \}$ one obtains:

$$H(d) = V - \min_{k \in \{1,...,U\}} \{ \sum_{t \in \{1,...,\mathbf{len}\ s\}} v(k,t) \cdot d(t) \},$$

where $d \in D(\mathbf{len}\ p)$.

According to the transformation **len** $p =$ **len** s, so $D(\mathbf{len}\ p) = X(\mathbf{len}\ s)$. Therefore

$$H(d) = V - L(d). \quad (10)$$

Since $h = V - l$, $M = C$, $p(t) = s(t)$, (9), and (10), condition (7) is equivalent to (8). Thus, MMK-D and KL-D are equivalent under the presented transformation. This ends the proof.

3. Greedy heuristics

Since the considered problem is strongly NP-hard, thus it is justified to design fast heuristic algorithms. First, a "workbench" will be presented which can be used by different greedy heuristics. Then, the heuristics will be discussed.

3.1. Knapsacks lightening workbench. Assume PRODUCT_TYPE(M, w, p, c) returns product type t that is to be removed from the knapsacks. PRODUCT_TYPE is a common interface to different greedy heuristics such as MostEffective or GreatestImpact that are described in the next subsections:

PRODUCT_TYPE(M, w, p, c) = MostEffective(M, w, p, c),
PRODUCT_TYPE(M, w, p, c) = GreatestImpact(M, w, p, c).

Then elaborating a decision can be described using Ada language:

```
type Price_List is array
  (Positive range <>) of Positive;
type Decision_Sequence is array
  (Positive range <>) of
    Natural range 0 .. 1;
type Weight_List is array
  (Positive range <>) of Natural;
type Contents is array (Positive range <>,
   Positive range <>) of Natural;
type Product_Type_Access is access
  function(
    M : in Positive;
    W : in Weight_List;
    P : in Price_List;
    C : in Contents;
    D : in Decision_Sequence
          ) return Natural;

NONE : constant Natural := 0;

procedure Decision(
  M0           : in  Positive;
        -- budget
  W0           : in  Weight_List;
        -- initial weights of knapsacks
  P            : in  Price_List;
        -- prices of products
  C0           : in  Contents;
        -- initial contents of knapsacks
  Product_Type : in Product_Type_Access;
        -- heuristics
  D            : out Decision_Sequence
        -- evaluated decision
                  ) is
 M : Natural := M0;
 W : Weight_List(W0'Range) := W0;
 T : Natural;
 C : Contents(C0'Range(1),C0'Range(2))
           := C0;
begin
 D := (others => 0);
 T := Product_Type(M, W, P, C, D);
 while T /= NONE loop
  D(T) := 1;
  for K in W'Range loop
   W(K) := W(K) - C(K,T);
   C(K,T) := 0;
 end loop;
 M := M - P(T);
 if M = 0
  then T := NONE;
  else T := Product_Type(M, W, P, C, D);
 end if;
 end loop;
end Decision;
```

Let RemoveProducts describe knapsack weights after removing products according to decision d. More precisely, RemoveProducts$(w, c, d) = w'$ if and only if

$$\forall_{k\in\{1,\ldots,\textbf{len}w\}} w'(k) = w(k) - \sum\nolimits_{t\in\{1,\ldots,\textbf{len}p\}} c(k,t)\cdot d(t).$$

Assume that Heaviest$(w) = k$ if and only if k is the heaviest knapsack (i.e. weight $w(k)$ is maximum). That function is undefined when w is an empty sequence.

Weight of the heaviest knapsack under decision suggested by DECISION(M, w, p, c) can be computed as follows:

$$w' = \text{RemoveProducts}(w, c, \text{DECISION}(M, w, p, c)$$

$$\text{Result}(M, w, p, c) = w'(\text{Heaviest}(w')).$$

3.2. The MostEffective heuristic. Efficiency of an item from the point of view of knapsack k is defined as $c(k,t)/p(t)$ (that resembles traditional understanding of the concept – see e.g. [1]).

Function MostEffective returns the most effective (and feasible with regard to budget M) product type. More precisely, MostEffective$(M, w, p, c) = t^*$ means that product type t^* is a solution to the following problem:

$$\text{maximize} c(\text{Heaviest}(w), t)/p(t) \text{ over} t = 1, ..., \textbf{len}\ p, \quad (11)$$

$$\text{subject to } p(t) \le M. \quad (12)$$

If there is no feasible solution to (11)–(12), then MostEffective returns the NONE value.

Let z_{ME} denote weight of the heaviest knapsack for the MostEffective heuristic, i.e. z_{ME} = Result(M, w, p, c) assuming that PRODUCT_TYPE(M, w, p, c) = MostEffective(M, w, p, c).

Theorem 2. Ratio z_{ME}/z_{KL} can be arbitrarily large.

Proof. Assume there are two knapsacks and two product types. Let's assume also the contents of each knapsack k is as presented in Table 1 (N is a parameter and $N > 2$).

Table 1
Contents of two knapsacks with two product types

$c(k,t)$	$t=1$	$t=2$
$k=1$	2	N
$k=2$	0	N

Assume $p(1) = 1$ and $p(2) = N$. Let $M = N$. The heaviest knapsack is knapsack number 1. Efficiency of the first type is $c(1,1)/p(1) = 2$, and for the second type it is $c(1,2)/p(2) = 1$. Thus MostEffective will choose product type 1, and there will be not enough money to transport product type 2. That results in $z_{ME} = N$. The optimal solution is to choose product type 2. Then $z_{KL} = 2$. Thus, for $N \to \infty$ also $z_{ME}/z_{KL} \to \infty$.

3.3. The GreatestImpact heuristic. Let MaxWeight(w, c, t) return maximum knapsack weight after removing products of type t from all the knapsacks. More formally, MaxWeight$(w, c, t) = m$ if and only if

$$\forall_{k\in\{1,\ldots,\textbf{len}\ w\}} w(k) - c(k,t) \le m$$

and

$$\exists_{k\in\{1,\ldots,\textbf{len}\ w\}} w(k) - c(k,t) = m.$$

Function GreatestImpact returns product type that has greatest impact on the knapsack maximum weight. More precisely, GreatestImpact$(M, w, p, c) = t^*$ means that product type t^* is a solution to the following problem:

$$\text{minimize MaxWeight}(w, c, t) \text{ over } t = 1, ..., \textbf{len}\ p \quad (13)$$

$$\text{subject to } p(t) \le M. \quad (14)$$

If there is no feasible solution to (11)–(12), then GreatestImpact returns the NONE value.

Let z_{GI} denote weight of the heaviest knapsack for the GreatestImpact heuristic, i.e. z_{GI} = Result(M, w, p, c) assuming that PRODUCT_TYPE(M, w, p, c) = GreatestImpact (M, w, p, c).

Theorem 3. Ratio z_{GI}/z_{KL} can be arbitrarily large.

Proof. Assume there are two knapsacks, and four product types. Let's assume also the contents of each knapsack k is as presented in Table 2 (N is a parameter and $N > 2$).

Table 2
Contents of two knapsacks with four product types

$c(k,t)$	$t=1$	$t=2$	$t=3$	$t=4$
$k=1$	1	1	0	N
$k=2$	1	1	N	0

Assume $p(1) = p(2) = 1$ and $p(3) = p(4) = N$. Let $M = 2N$. GreatestImpact will choose product type 1, 2 and one of 3, 4. That results in $z_{GI} = N$. The optimum solution is to choose products 3 and 4. Then $z_{KL} = 2$. Thus, for $N \to \infty$ also $z_{GI}/z_{KL} \to \infty$.

4. Exact algorithm

An exact algorithm can serve two purposes: it can be used to solve small problem instances, and to evaluate precision of various heuristics. In this section we describe a branch-and-bound algorithm and evaluate its efficiency (in sense of execution time).

Lower bound is acquired as minimal value of results obtained by the GreatestImpact and MostEffective heuristic algorithms for a given problem instance.

Upper bound is based on, what we call, *LP-separate relaxation*. It is a combination of two relaxations: the well-known LP-relaxation (see e.g. [1]) and separate relaxation. The former replaces $\{0, 1\}$ decision sets with $< 0, 1 >$ intervals. Separate relaxation breaks the initial problem up into a set of single-knapsack lightening problems, which can easily be transformed to the classical knapsack problem. To define LP-separate relaxation of KL more formally assume

$$D'(m,n) = \{d'(k,t) : \forall_{k\in\{1,\ldots,m\}} \forall_{t\in\{1,\ldots,n\}} 0 \le d'(k,t) \le 1\}.$$

Let $d' \in D'$(**len** w, **len** p). Then function $H(d)$ defined by (1) can be redefined in the following way:

$$H'(d') = \max_{k\in\{1,\ldots,\textbf{len } w\}}\{w(k) - \sum_{t\in\{1,\ldots,\textbf{len } p\}} c(k,t)\cdot d'(k,t)\}. \tag{15}$$

A relaxed version of KL can be stated as follows:

$$\text{minimize } H'(d') \tag{16}$$

$$\text{subject to } \forall_{k\in\{1,\ldots,\textbf{len } w\}} \sum_{t\in\{1,\ldots,\textbf{len } p\}} p(t)\cdot d'(k,t) \le M$$
$$d' \in D'(\textbf{len } w, \textbf{len } p). \tag{17}$$

Problem (4.2), (4.3) is equivalent to the following one:

$$\text{maximize } \sum_{t\in\{1,\ldots,\textbf{len } p\}} c_k(t)\cdot d_k(t), \tag{18}$$

$$\text{subject to } \sum_{t\in\{1,\ldots,\textbf{len } p\}} p(t)\cdot d_k(t) \le M$$
$$0 \le d_k(t) \le 1, \tag{19}$$

where $c_k(t) = c(k,t)$, $d_k(t) = d'(k,t)$. That problem can be easily solved using the split concept [1].

5. Experimental evaluation of greedy heuristics

In this section the experimental evaluation of two greedy heuristic algorithms proposed in Sec. 3 is presented. The exact branch-and-bound algorithm presented in Sec. 4 was used as a base to estimate average and maximal relative error of both heuristic algorithms.

The computational experiment was performed for 3 values of proportion of $M/\sum p(t)$ equal to 30%, 60% and 90% respectively. In the first part of the experiment the response time of exact algorithm was measured. Figure 1 summarizes results of this phase. It is easy to notice the exponential time complexity of the algorithm – in practice execution time for instance of over 25 product types becomes unacceptable.

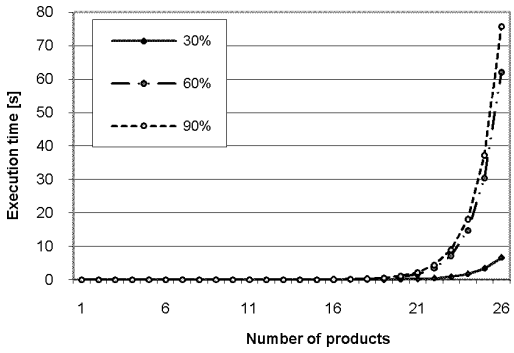

Fig. 1. Exact algorithm execution time vs. number of products for 3 different proportions of $M/\sum p(t)$

Another important and fortunate factor is that the response time depends heavily on the value of proportion $M/\sum p(t)$ (the smaller the value of the proportion the faster we get the answer). It is a fortunate situation since usually the size of cache memory is significantly (over 4 times) smaller than the sum of the sizes of all elements.

5.1. Precision. In the next step of the computational experiment the accuracy of both heuristic algorithms was evaluated. The test suite comprised overall 15,600 test cases.

The input data was randomized 200 times for a given number of products (1 to 26) and proportion of $M/\sum p(t)$ in the following way:

- first, the prices of all products were randomized in the range $< 1, 100 >$,
- then the budget was computed as the product of sum of the prices of all products and the current proportion of $M/\sum p(t)$,
- next, the contents of all knapsacks was randomized in the range $< 0, 100 >$,
- finally, the weights of knapsacks were computed as the sum of all products in the knapsack.

Figures 2 and 3 depict average error of heuristic algorithms for all tested proportions of $M/\sum p(t)$ respectively. Presented results allow to draw a general conclusion that for number of products greater than approximately 5 it is better to use MostEffective heuristic algorithm because it provides results closer to an optimal value than the ones obtained by its counterpart. Once again a fortunate situation occurs – for small values of $M/\sum p(t)$ proportion MostEffective is significantly more accurate than GreatestImpact.

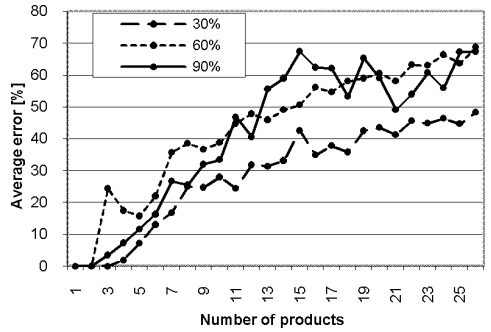

Fig. 2. Average error of heuristic algorithm GreatestImpact compared to Branch-and-Bound result for all proportions of M/$\sum$p(t) vs. number of products

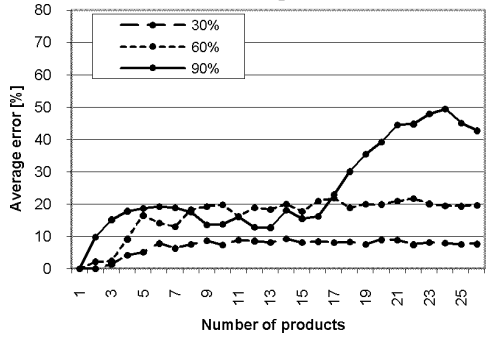

Fig. 3. Average error of heuristic MostEffective compared to Branch-and-Bound result for all proportions of M/$\sum$p(t) vs. number of products

Unfortunately, the average relative error increases with increasing number of products.

A similar dependence can be observed while comparing maximum relative error of Most Effective and GreatestImpact heuristic algorithms (Figs. 4 and 5).

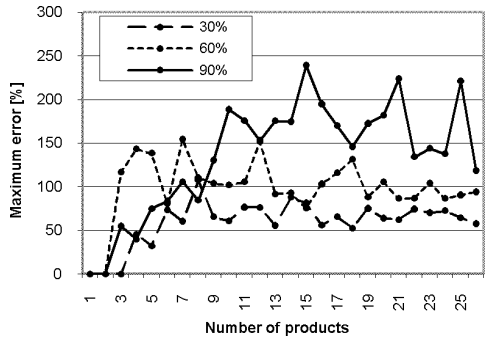

Fig. 4. Maximum error of heuristic algorithm GreatestImpact compared to Branch-and-Bound result for all proportions of M/$\sum$p(t) vs. number of products

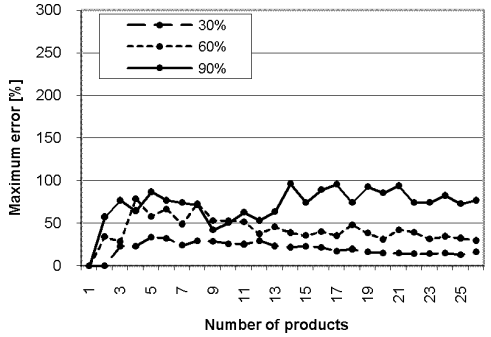

Fig. 5. Maximum error of heuristic algorithm MostEffective compared to Branch-and-Bound result for all proportions of M/$\sum$p(t) vs. number of products

5.2. Execution time. The last step of the computational experiment comprised comparison of performance of MostEffective and GreatestImpact algorithms. The results of this comparison (Figs. 6 and 7) prove polynomial time complexity of both algorithms. Unfortunately, more accurate one – MostEffective is slower and its time complexity increases faster with increasing number of products than it is in its counterpart situation.

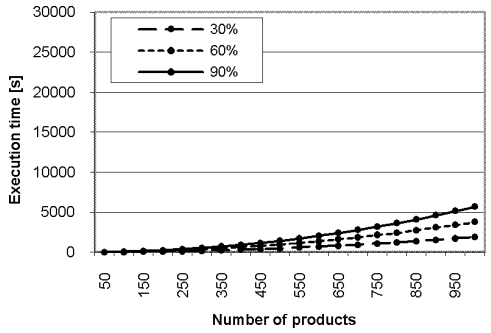

Fig. 6. Execution time of heuristic algorithm GreatestImpact for all proportions of $M/\sum p(t)$ vs. number of products

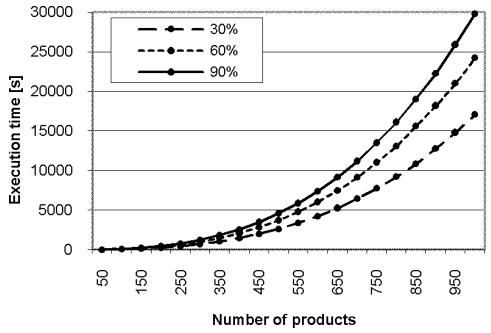

Fig. 7. Execution time of heuristic algorithm MostEffective for all proportions of $M/\sum p(t)$ vs. number of products

All tests were performed using AMD Athlon 64 3.8+ GHz CPU testbed equipped with 512 MB of RAM and using Turbo Delphi 10.0.

The general conjuncture after analysis of the results of all tests is to use a combination of exact algorithm for small instances of the problem (number of products less than 15) and MostEffective algorithm for larger instances.

6. Knapsacks lightening and periodic loading with cache memory

In hard real-time systems (HRT systems) timeliness is as important as functional correctness. Aircraft control systems, nuclear power plants, flight control systems, car's ABS – all of them contain so called hard real time tasks (HRT tasks) which must be finished by a given deadline.

One of the methods of scheduling of HRT tasks is periodic loading introduced by Schwartzer, Dror, and Trudeau [3] and further investigated by Zeng et al. [4]. In this approach we are given a set of periodic, non-preemptive tasks to be scheduled on one processor (in case of a distributed system, tasks are first allocated to single machines and then scheduled on each of the processors). Processor time is split into a sequence of time frames of the same size. In this context scheduling means allocating instances of periodic tasks to time frames. Each task is given a frame period and one has to choose for each task a starting frame when its first instance will be executed. It should be done in such a way that the maximum load over all the frames is minimized (load of a frame equals total execution time of all the task instances assigned to that frame). A schedule is feasible if the maximum load is not greater than the frame size. Minimization of periodic load is strongly NP-hard [5]. However, a number of quite effective heuristic algorithms have been proposed [5, 3].

Periodic loading, as considered by Schweitzer, Dror, and Trudeau [3], does not take into account cache memory. Cache memory can speed-up execution of a program 10 times or more [6], so it seems too important to be neglected. But for many years in the area of HRT applications cache memory was considered a nuisance. The reason was unpredictability. Traditional cache memories were oriented towards minimization of average execution time while in the area of HRT applications what really matters is worst-case behavior (many of HRT applications are mission-critical). Recently some hardware and software solutions have been proposed that make cache memories more predictable. One of them is line threading [6] that allows fetching code and data of an HRT task into cache memory and locking them there. Objects locked in cache memory are called residents. By adding a cache memory and by placing some objects in it one can shorten load of some frames (hopefully, load of mostly loaded frames) and thus one can make a non-feasible schedule a feasible one. Since usually a cache memory is considerably smaller than the main memory, a combinatorial problem arises how to choose residents of the cache memory to get a feasible schedule. The approach studied in this section is quite simple. Scheduling is split into two stages: initial scheduling and schedule improvement.

6.1. Initial scheduling. At this stage a tentative schedule is generated which is based on the original version of the periodic loading problem [3]. Duration of each task is estimated with an assumption that during its execution the task will be outside of the cache memory. Output of this stage is a sequence of frames with task instances assigned to them (see Fig. 8). What really matters is load $\lambda(\varphi)$[1] of each frame φ that can be computed by simply adding durations of all the tasks instances assigned to a given frame φ. Frame loads, when transformed to Knapsack Lightening, will correspond to initial knapsack weights $w(k)$.

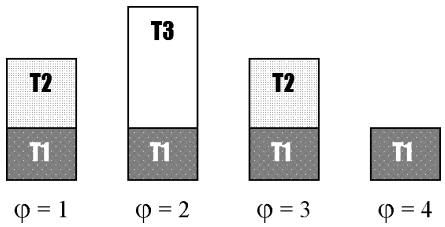

Fig. 8. Four frames with three periodic tasks: T1, T2, T3 (their periods are 1, 2, and 4, respectively). Assuming duration of T1 and T2 is 5, and duration of T3 is 10 one gets the following frame loads: $\lambda(1) = 10$, $\lambda(2) = 15$, $\lambda(3) = 10$, $\lambda(4) = 5$

6.2. Schedule improvement. Given a design description one decomposes the system into a set of elements that can be placed in the cache memory. Those elements correspond to procedures, arrays, class objects etc. Each element ε has its size $\sigma(\varepsilon)$ representing number of memory cells required to store it in the cache memory. Ω describes the size of cache memory, and the total size of elements placed in it cannot exceed Ω.

To take into account impact of cache memory on load $\lambda(\varphi)$ of each frame φ, for each element ε a profit $\pi(\varepsilon, \varphi)$ is specified that describes how much $\lambda(\varphi)$ will be decreased by placing ε in the cache memory. Thus, when ε (and only ε) is in the cache memory, *effective load* of frame φ is $\lambda(\varphi) - \pi(\varepsilon, \varphi)$. If set of elements S are placed in the cache memory, *effective load* of φ equals:

$$\lambda(\varphi) - \Pi(\varphi), \quad \text{where} \quad \Pi(\varphi) = \Sigma_{\varepsilon \in S} \pi(\varepsilon, \varphi). \tag{20}$$

Notice that $\pi(\varepsilon, \varphi)$ allows to describe cases when an element (e.g. an array) is shared by a number of frames (tasks) and has different impact on different frames.

The question is what is minimal value of the maximum frame load (the lower the maximum load the greater the chance that the schedule will be feasible). Transformation of that problem to Knapsack lightening is straightforward (frames correspond to knapsacks, frame loads to initial knapsack weights, system elements to product types, cache size to budget limit, and the number of memory cells required for each task to price of product type).

7. Conclusions

In hard-real time systems the resources are usually limited and there is a trade-off between resources availability and execution time. An example of the trade-off is the problem of dynamic storage management in hard-real time systems [7]. Another issue, discussed in the paper, is cache memory management in hard-real time systems. The paper presents an

[1]To avoid confusion with symbols used in the previous sections we decided to use Greek letters for symbols introduced in Sec. 6.

extension to the traditional periodic loading approach. That extension allows for deterministic utilization of cache memory in hard real-time systems. It considerably increases the possibility of finding a feasible schedule. The results of the computational experiments show significant improvement of minimization of the value of maximum time-frame load. For large instances of the problem the MostEffective heuristic presented in Sec. 3.2 is recommended, while for small instances (below 30 periodic tasks – see Fig. 1) one can use an exact algorithm described in Sec. 4, which is based on the branch-and-bound approach. To speed-up the exact algorithm one can apply parallelization similar to that presented in [8].

Acknowledgements. We would like to thank Silvano Martello, Piotr Zielinski and Bartosz Bogacki for helpful discussions concerning the paper. This research has been financially supported by the Polish Ministry of Science and Higher Education under grant N519 188 933.

REFERENCES

[1] J. Błażewicz, P. Formanowicz, W. Kubiak, M. Przysucha, and G. Schmidt, "Parallel branch and bound algorithms for the two-machine flow shop problem with limited machine availability", *Bull. Pol. Ac.: Tech.* 48 (1), 105–115 (2000).

[2] W. Complak, "Deterministic approach to the notebook memory management in systems strongly conditioned by time", *Doctoral Dissertation*, Poznań University of Technology, Poznań, 2001, (in Polish).

[3] H. Kellerer, U. Pferschy, and D. Pisinger, *Knapsack Problems*, Springer Verlag, Berlin, 2004.

[4] J.R. Nawrocki and A. Urbański, "Optimization of region-based storage allocation", *Bull. Pol. Ac.: Tech.* 42 (4), 605-617 (1994).

[5] J.R. Nawrocki and A. Czajka, "Task classyfying in systems strongly conditioned by time with the use of the metod of cyclic loading", *Silesian University of Technology Periodicals: Automatics* 1389, 169–179 (1998), (in Polish).

[6] P.J. Schweitzer, M. Dror, and P. Trudeau, "The periodic loading problem: formulation and heuristics", *INFOR* 26 (1), 40–62 (1988).

[7] G. Yu, "On the max-min 0-1 Knapsack problem with robust optimization applications", *Operations Research* 44 (2), 407–415 (1996).

[8] D.D. Zeng, M. Dror, and H. Chen, "Efficient scheduling of periodic information monitoring requests", *EJOR* 173 (2), 583–599 (2006).