RANDOM START AND FORWARD SEARCH APPLIED TO SOLVE MULTI-CRITERION PLANNING PROBLEMS

Dang Thanh Tung[(1)], Baltazár Frankovič[(1)], Con Sheahan[(2)], Ivana Budinská[(1)]

(1) Institute of informatics, Slovak academy of sciences, Dubravska 9, Bratislava 84507, Slovakia
(2) Dept. Of Manufacturing & Operations Eng., University of Limerick, Limerick, Ireland.
Email: utrrtung@savba.sk

Abstract: This paper deals with multi-criterion planning problem. Beside traditional constraints, we assume that tasks can migrate between resources and they are executable by many methods with different results. The random start & forward search algorithm is proposed to solve the mentioned problem, in condition, the time for solving is limited.

Keywords: planning, multi-criterion optimization, forward search.

1. INTRODUCTION

Planning in manufacturing belongs to the category of complicated problems. Beside the complication caused by the task structure, there are another problems related to individual resource's limitation or personal requirements. In addition, the main criterion of solving many planning problems is not only to find a plan with the minimal production time, but also to find such a plan with minimal production cost, maximal quality or utilization of each resource, etc. This paper focuses on proposing a new method for resolving multi-criterion planning problems based on heuristic search.

2. PROBLEM DESCRIPTION

Planning problem in this work is understood as follows: There is a set of tasks **S0**=$\{Task_1,..,Task_m\}$, each of them is executable by several alternative ways with different results.

The task structure is specified clearly for each planning problem. Each task has a set of predecessor and successor tasks, whose execution is associated directly with its execution. A task that does not have a predecessor (or a successor) is called a starting (or ending) task, respectively. Each planning problem can have a number of stating and ending tasks.

Next, there are a number of resources (equivalent or different) for executing these tasks. Each task is executable in several resources (in a special case, each task is executable in whichever resource). Let us assume that all tasks are uninterruptible during execution and one resource can perform only one task at any time.

For each *Task*, let us denote *Me*(*Task*) as a set of executable methods for performing this task. *Q*(*Task*, *method*, *input*) is a multi-optional function $\{Task \times method \times input\_data\} \rightarrow R^r$ describing the quality of the obtained result after executing this *Task*, by using the method $method \in Me(Task)$ and the input data *input*. A variable *input* involves necessary data for executing this task, which might be gathered from the external environment or from executions of other tasks connected with this one. The set R^r is an *r*-dimensional set of real numbers representing *r* types of criteria, which are used to assess a task's execution. Three usual options for assessing a task's execution are cost, duration and quality. *Cost* of a task describes the financial or opportunity cost inherent in performing the task. *Duration* describes the amount of time that a method will take to execute the task. *Quality* describes the "goodness" of performing the task. Of course, different applications have different notions of what corresponds to model quality. For example, a *quality* might include accuracy, speed or completeness of a task result, etc.
Each planning problem is associated with many constraints, e.g. deadlines, limited capacities of resources, predefined quality, etc. Moreover, in order to assess different plans, a common criterion function

is defined. This function is usually multi-parameter function and it has not measurement.
The main objective is to find such a sequence of performing given tasks in order to satisfy all defined constraints and simultaneously to minimize the predefined criterion function.

3. PLANNING AS HEURISTIC SEARCH

It is known that planning problems could be solved by two basic manners. The first kind of methods constructs the global plan by successively adding new tasks to the already examined part. The initial set of tasks is divided into a number of portions and in each phase one of these portions is added to the already examined set of tasks. This method requires recording all intermediate results for future calculation. As a result, the process solving requires exhausting work and the space demanded for backing up information might grow considerably when the amount of tasks is great.

The second manner is to improve the current plan. The process starts with any initial plan. In order to satisfy required criteria this plan is re-constructed, until it satisfies the desired requirements. Here, the essential difficulty is in selecting a part of the current plan to repair. However, theoretically, not necessarily this process brings better solutions. Beside that, the complexity of this approach is the same as in the previous one, if solver wants to ensure the optimality of the achieved solution. The difference is this method could provide a solution at any time. Therefore, it is appropriate to solve problems that do not have fixed deadlines. Due to the main advantage of the second method that allows providing a final solution at any time, the method presented here is built in this principle.

4. RANDOM START AND PLAN REPAIRING - A GENERAL SCHEME

The proposed method has three basic stages: The first stage chooses a random initial plan for repairing. In the second stage, the initial plan is decomposed into a number of disjoint sub-plans, and one of them is chosen for rescheduling. In the last stage, tasks in the selected part are rescheduled to improve the current plan. During reorganization, historical data can be taken from the database (if they exist) to evaluate newly created plans. This cycle continues until time expires.

In general, generating an initial plan could be made at random by taking any plan, if the problem solving is too complicated. However, the importance is not to generate already examined plans.

In the second stage, the current plan is decomposed to a number of disjoint parts. Then, one or several of disjoint parts are selected for rescheduling.

On the basis of predefined constraints the solver calculates and chooses parts of the current plan for rescheduling (explanation in more can be found in Section 5). Of course, there are many different constraints, which eventual plans have to satisfy, e.g., deadline for the last task of a plan, minimal cost or quality. Beside that, there are such constraints that are generated from particular task relationships.

In the last phase (the most important one) the solver tries to modify an order of tasks in the selected part, with the expectation to achieve a better plan (according to the common measure). The solver takes the last best-achieved plan as a target for comparison with newly created ones.

Repairing process is an iterative one, in which the solver successively modifies an order of each task in the selected parts, until all variants are examined or plan(s) with the desired parameters is achieved. If better results are not achieved, more parts can be selected for rescheduling. Repairing process might continue to improve the plan's quality until time limitation expires. Plans are selected on the basis of the common measurement. A plan quality is represented by a vector $\{z_1, z_2, ..., z_r\}$, where $_{i\in[1,r]}|z_i$ is one of parameters. Plans could be assessed by a coefficient *evalu(plan)* defined by the following equation:

$$evalu(plan) = \alpha_1 z_1 + ... + \alpha_r z_r \qquad (1)$$

or

$$evalu(plan) = \sqrt{(\alpha_1 z_1)^2 + \ldots + (\alpha_r z_r)^2} \qquad (2)$$

where $\alpha_{1,..,r} \geq 0$ are weights defined by the user, which express the priority of each criterion over others in selecting final plans; and they satisfy a condition:

$$\alpha_1 + \alpha_2 + \ldots + \alpha_r = 1. \qquad (3)$$

The criterion function *evalu(plan)* could also be more complicated, if the user evaluates each parameter by different manners, e.g. the cost might multiply to the second degree, but plan time and makespan are evaluated by linear combination.

5. DECOMPOSITION AND SELECTION FOR RESCHEDULING

The first difficulty is selecting parts of the current plan for rescheduling. Let us divide the initial set of tasks into two parts: one consists of rescheduled tasks (marked as *Set_resche*) and the second one consists of the rest of tasks (marked as *Set_irre*). Tasks in *Set_irre* have unchanged orders and methods for execution during the rescheduling process. Two most important criteria for choosing parts for rescheduling are time and cost requirement; however, there could be more criteria, such as, quality, idle time, etc.

5.1 Selection of parts for rescheduling from the time point of view

Let assume that there is a predefined *deadline* when the last task has to be terminated. For each task $Task_i$ we define:

– T_m_i is minimal duration of its execution (among all possible methods for performing this task),

– T_i is duration of this task by using the current method,
– $Start\_t_i$, is its start time, End_t_i is its termination time.

$$End\_t_i = Start\_t_i + T_i \geq Start\_t_i + T\_m_i \quad (4)$$

Each task has a set of predecessor and successor tasks. Denote

– $Task_i \Rightarrow Task_k$ means $Task_k$ requires results of $Task_i$.
– $pre(Task_i)$ is a set of predecessors of $Task_i$: $\forall Task_k \in pre(Task_i) \mid Task_k \Rightarrow Task_i$.
– $succ(Task_i)$ is a set of successors of $Task_i$: $\forall Task_k \in succ(Task_i) \mid Task_i \Rightarrow Task_k$.
– $Task_i$ is called a starting task if $pre(Task_i)$ is empty; and it is the ending task if $succ(Task_i)$ is an empty set. There might be a number of starting and ending tasks - let us denote S_Tasks and E_Tasks as a set of starting and ending tasks, respectively.
– $min\_duration_i$ denotes minimal time interval from start of $Task_i$ till end of the last task.

In the following definitions I present the necessary conditions for feasible plans.

Definition 1: A plan is realizable if the following conditions are satisfied for all *i, k*:

– if $Task_i \Rightarrow Task_k$, then $Start\_t_i + T\_m_i \leq Start\_t_k$,
– if $[Start\_t_i , Start\_t_i + T\_m_i] \cap [Start\_t_k, Start\_t_k + T\_m_k] \neq 0$ then $Task_i$ and $Task_k$ are executed in two different resources. □

For each plan, let us define the parameters *Time_end*, which represent the time when the last task is terminated. In order to fulfil the deadline condition, an inequality $Time\_end \leq Deadline$ has to be valid. It follows:

$$Start\_t_i \leq Deadline - min\_duration_i \quad (5)$$

Equation (5) shows out the *necessary condition* that each task has to fulfil in order to keep the specified deadline. In other words, Equation (5) specifies the latest time when each $Task_i$ has to start in order to keep a deadline. All tasks that violate this condition must be included in the selected part (*Set_resche*) for rescheduling. Here the challenge is how to identify the value $min\_duration_i$ for an arbitrary $Task_i$.

In this paper we assume that all the resources are not equivalent and each task can be executed in only several ones of them (a task can be migrated between these equivalent resources). For example, a job-shop scheduling problem, in which each task could be executed in only one type of resources, not in all. In order to estimate values *min_duration* in a general case, when all resources are not the same, the following method is proposed.

Genetic Heavy Weight Task (GHWT): let us denote *last_time* as the time when a resource finishes the last operation. At the beginning setting *last_time* = 0 for each resource.

- Each resource takes a task with the longest duration (*T_m*), which is executable at this moment by this resource.
- In each turn: choose a resource with a minimal *last_time*. If tasks that are executable in this resource exist (with all predecessors already assigned), then the one with the longest duration (*T_m*) will be selected for execution, otherwise the next resource with the second minimal *last_time* is selected, etc. Record a *Start_t* and update a value *last_time* by *T_m* of the newly assigned task.
- Stop when all tasks are assigned.
- After constructing a plan, calculate a value *min_duration* for each task as follows:

$$min\_duration_i = Time\_end - \min_j Start\_t_j + T\_m_i \quad (6)$$

where $Task_j \in Succ(Task_i)$ and $\forall$i=1,..,m values $Start\_t_i$ are stored before.

It is easy to verify that GHWT has a complexity ($\cong O(m^2)$), since in each turn the number of tasks that this algorithm examines is maximally *m*. Variables *min_duration* have also another function – they are used to estimate the time when a plan finishes.

5.2 Selection of parts for rescheduling from the cost point of view

Let us assume that there is a maximal defined cost (*max_cost*) that a plan can have. To simplify we propose that an execution cost of each task depends on only the method selected for execution. On the basis of such assumption the total cost of a plan is calculated as the sum of the cost of all tasks execution. Let us define:

- c_i is a cost of $Task_i$ by using the current method for execution.
- min_c_i is the minimal cost of $Task_i$ among all applicable methods.
- max_c_i is the maximal cost of $Task_i$ among all applicable methods.
- $total\_cost = \sum c_i|_{\forall i}$ is the total cost of a plan.

It is provable that:

$$max\_cost - \sum_{Task_i \in Set\_irre} c_i \geq \sum_{Task_i \in Set\_resche} \min\_ c_i \quad (7)$$

Equation (7) expresses the necessary condition that the selected part of tasks for rescheduling has to fulfil. More details could be found in [Dang, 2003].

6. PLAN REPAIRING

After selecting a part (or several parts) of tasks for rescheduling, repairing process is started. Starting with the unchanged part of the initial plan, step-by-step the solver adds a number of tasks from part *Set_resche* to a plan. The best-first search method is used to find the best intermediate plan, but with some modifications (to be explained later). This process continues until all tasks from set *Set_resche* are added to the plan.

6.1 Forward search applied in plan repairing

The main idea of forward search (FS) is explained as follows: Starting with a plan that is constructed of tasks from *Set_irre*(called Part 1), solver tries to explore all possible neighboring sub-plans, which could be created by adding some tasks from set *Set_resche* to Part 1. It is similar to the best-first search method (BFS); the difference is that, in order to increase the quality of eventual plans, first performs a branch-and-bound search to a certain depth. Afterwards, it selects one of the best intermediate plans for a new root and continues searching from this one. As a result, there is a larger chance to reach a better choice than the BFS.

Comparing temporary plans requires parameters of the temporary plan and the remaining part of a plan created by unassigned tasks. Because FS is used, the parameter of the temporary plan is identified immediately. Identifying the second parameter is more difficult, because it requires examining all potential variants that unassigned tasks can create. For that reason, I have adopted an idea from [Kumar *et al*., 1994, *chapter* 8], [Nguyen *et al.,* 2002] to predict these parameters.

Referring to Figure 1, after choosing a part of tasks for rescheduling, all values $min\_duration_i$ for $Task_i \in Set\_resche$ are loaded from the database. Next, the total cost of all the selected tasks is calculated easily. Without loss of generality, let us assume that only one level ahead will be calculated, and $Task_4$ and $Task_6$ are added at level 1 to the temporary plan. Both tasks use their first method for execution with corresponding parameters $Task_4$: $\{T_4, c_4\}$ and $Task_6$: $\{T_6, c_6\}$. $Task_4$ is executed by resource 1, and $Task_6$ by resource 2.

Solver uses the stored values *min_duration* to estimate the time when the final plan will finish. Let us denote $t_1, t_2, t_3, \ldots$ as the time when resources perform their last operation (indexes 1, 2, 3 correspond to the number of resources) and *Time_end* as the time when a plan will finish. T_{remain} is the time interval after $Task_4$ and $Task_6$ are finished up to the end of a plan. An estimate of values T_{remain} and *Time_end* is made as follows:

$$T_{remain} \cong \max[(min\_duration_4 - T\_m_4), (min\_duration_6 - T\_m_6)] \quad (8)$$

and

$$Time\_end \cong \max[t_1 + T_4 + (min\_duration_4 - T\_m_4), t_2 + T_6 + (min\_duration_6 - T\_m_6), t_3, .] \quad (9)$$

Due to the definition of *min_duration* introduced in Section 5, values T_{remain} and *Time_end* cannot exceed the real time when a plan finishes. Because all variables of Equation (8) and (9) are known, values T_{remain} and *Time_end* can be specified immediately without a complicated calculation. Both Equation (8) and (9) can be extended for a general case where instead of $Task_4$ and $Task_6$ they can work with all tasks that are assigned at that moment to a temporary plan. In a general case, let $Task_{i1}$ be added to resource 1, $Task_{i2}$ to resource 2, $Task_{i3}$ to resource 3, etc.., then values T_{remain} and *Time_end* can be estimated as:

$$T_{remain} \cong \max[(min\_duration_{i1} - T\_m_{i1}), (min\_duration_{i2} - T\_m_{i2}), \ldots] \quad (10)$$

and

$$Time\_end \cong \max[t_1 + T_{i1} + (min\_duration_{i1} - T\_m_{i1}), t_2 + T_{i2} + (min\_duration_{i2} - T\_m_{i2}), \ldots] \quad (11)$$

Variables T_{remain} and *Time_end* are close to the real values, if all tasks are executed by the shortest methods. To get more precise values of these variables, calculating T_{remain} and *Time_end* as well, when all the remaining tasks are executed by the longest methods is useful. The process to get these values is similar as above. Estimates of time when a plan will finish could be calculated as a combination of these values.

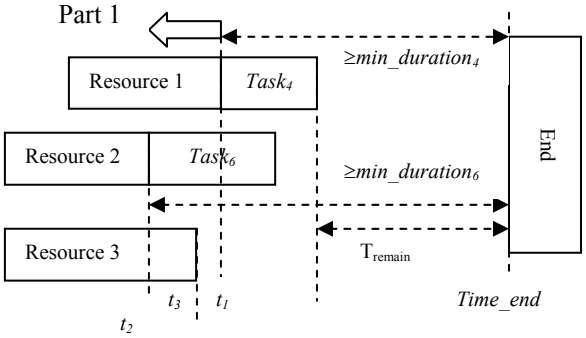

Figure 1: Estimation of execution time

An estimate of the cost of the second part of a plan could be made in a similar way. Let c_{remain} denote the total cost of all unassigned tasks after adding $Task_4$ and $Task_6$ to a temporary plan. An estimate of c_{remain} is computed as the total sum of the minimal cost of each unassigned task:

$$c_{remain} \cong min\_c_{resche} - (min\_c_4 + min\_c_6) = min\_c_{remain} \quad (12)$$

min_c_{resche} is an estimate cost of all unassigned tasks when they are executed by the method with lowest cost; min_c_4 and min_c_6 are minimal performance cost of $Task_4$ and $Task_6$. All variables of Equation (12) are known; therefore the cost estimate could be specified immediately. Similarly, it is possible to ensure that the cost estimate calculated by Equation (12) does not exceed the real cost of reaching the goals from the current state. This equation could also be extended for a general case with an arbitrary number of tasks. Let us use the same assumption as above: $Task_{i1}$ is added to resource 1, $Task_{i2}$ to resource 2, $Task_{i3}$ to resource 3, etc..; then the cost estimate is calculated as the total sum of the minimal cost of each unassigned task:

$$c_{remain} \cong min\_c_{resche} - (min\_c_{i1} + min\_c_{i2} + min\_c_{i3} + ..) = min\_c_{remain} \quad (13)$$

Of course, min_c_{remain} defined as in Equation (12) or (13) is minimal cost of the remaining part of a plan, which requires executing all the rest of tasks by such methods that guarantee minimal cost. In order to get more precise values of c_{remain} another variable, called max_c_{remain} is used, which expresses maximal cost of

the rest of tasks. c_{remain} then could be calculated as a linear combination of the variables (min_c_{remain} and max_c_{remain}), e.g.,

$$c_{remain} = \beta_1 \, max\_c_{remain} + \beta_2 \, min\_c_{remain} \qquad (14)$$

where $\beta_1 + \beta_2 = 1$. On the basis of the achieved results, coefficients $\{\beta_1, \beta_2\}$ are modified to adjust to the criterion function, in order to find the most appropriate combination. Calculation of max_c_{remain} is similar to the calculation of min_c_{remain}, so explanation is omitted. In the next part the method for plan repairing is presented.

7. RANDOM START AND PLAN REPAIRING BASED ON FORWARD SEARCH

In this section the algorithm used for solving a multi-parameter planning problem is presented.

First: using the GHWT method for calculating T_{remain} (presented in Section 5.1).

The RSaFS Algorithm:

Phase 1: choose a random initial plan.
Phase 2: choose parts *Set_irre* and *Set_resche* for rescheduling – Section 5.
Phase 3: initialization:

a. *solution* ← the current temporary plan
b. *g(solution)* and *h(solution)* are a vector of all parameters of the current sub-plan and the remaining part of a plan, which consists of unassigned tasks

1. calculate all possible configurations to *k* steps ahead; constant *k* is defined in interval [1,10].
2. for each constructed temporary plan, estimate values of a minimal time termination, minimal and maximal cost of the remained part,
3. choose a temporary plan with the highest promising results according to the defined measure (Equation (1) or (2)) (*calculating parameters of the eventual plan is presented below*);
4. update the *solution*, then return to step 1
5. stop when all tasks are assigned. Update the currently best solutions,
6. compare the time and cost criteria, if the cost criterion has more important influences upon the criterion function, then increase β_2. Otherwise, increase β_1. Restart phase 1;
7. if a newly achieved plan has significantly different parameters from the estimated ones (essentially, time when a plan finishes), recall GHWT to recalculate all estimate values T_{remain}, but on the basis of the new plan.

Phase 1 is easy for understanding. An initial plan could be chosen at random.

Phase 2 has been explained in detail in Section 5. In Step 4 of Phase 3, parameters of an eventual plan are calculated on the basis of values of vectors *g(solution)* and *h(solution)*. The time when a plan finishes could be estimated by using Equation (10) and (11) in which values T_{remain} are taken from the procedure that has been called before starting the repairing phase. The cost of a final plan is estimated by merging the cost of *g(solution)* and *h(solution)*.

On the basis of the estimated values, the newly constructed plan with highest evaluation is chosen as a new state of a search. Then, the set of unassigned tasks is reduced by the tasks that have been added to the selected plan. Steps 7 and 8 are used to get more precise cost and time estimates, since a situation might happen when the first estimate values are far from the realistic ones.

8. SIMULATION RESULTS AND COMPARISON

The algorithm selected to compare with RSaFS is a heuristic search plan – HSP-r [Bonet & Geffner, 2001]. HSP-r is essentially the best-first search method, but it deals with only the cost parameter. Both the algorithms are implemented in C++.

The case of study chosen for solving is a standard scheduling problem, which could be described as follows: there are 10 different products, each of them consists of 10 different tasks (or operations), which are executable in specific resources. Each task could be executed by two methods with different duration and cost. In general, when duration increases, the cost decreases and vice versa, but these variables are not directly dependent one from the other. There are five groups of resources; each of them consists of two equivalent machines. Tasks can be migrated among these equivalent machines.

Simulation results are shown in Table 1. Each algorithm has 5s for running (there is only time of using processor for calculation, without time needed for operation system or generating input data). The criterion function used to evaluate plans is: $eval(plan) = 0.6*time + 0.4*total\_cost$, where *time* is the time when the last task is accomplished, and *total_cost* is the total cost of execution of a plan. (β_1, β_2) are coefficients used to make estimates of the cost. The estimated cost is calculated as follows:

$$c_{remain} = \beta_1 \, max\_c + \beta_2 \, min\_c, \qquad (15)$$

max_c and *min_c* are the maximal and minimal costs of the rest of tasks.

In Table 1, RSaFS-10, or -all, mean that after finishing the first step, only 10 the best states, or all states are taken for consideration within the framework of two-step forward search, respectively. HSP-r uses the best-search algorithm and it chooses only one best temporary state to continue its search. In comparison, RSaFS explores many states at once and, moreover, it performs a search forward in several steps. Therefore, there is a larger probability than in HSP-r that the selected state is the optimal one. The next significant difference is that RSaFS allows improving the current estimation of the cost by changing coefficients β_1 and β_2 (from Equation (15)) during searching in order to adjust to the criterion function. HSP-r uses a fixed estimate of the cost, but as the achieved results show, there is not a

good combination for every situation, which would always guarantee the best solution. RSaFS modifies coefficients β_1 and β_2 in such a way, in order to focus on finding such plans, which have a better chance to optimize the criterion function. For example, in these experiments, when a plan with parameters (*execution_time*= 80 and *total_cost*=450) is found, it is easy to see that the cost has bigger influence on the criterion function than the time of execution. Therefore, solver tries to find a plan with smaller cost, increases coefficient β_2 and decreases β_1. Thus, the cost estimate prefers plans with small cost, close to the minimal cost of execution, to those with short time of execution. If this trend achieves better solutions, the process continues. Otherwise, solver tries with other coefficients. In all experiments this process converts to the conclusion that calculation of estimated cost by using only the minimal cost (β_1=0 and β_2=1) brings the best results of all. However, this conclusion is not always true for every planning problem. For example, when the cost of plan is much lower than the time of execution, then using the maximal cost only (equivalently with preferring to use methods that have a short duration for executing tasks) for making estimates brings the best results. In cases when the criterion function has more parameters or it is not a linear function, setting appropriate coefficients $\{\beta_1, \beta_2,\ldots\}$ might not be an easy task, but modifying them frequently in order to adjust to the criterion function achieves really better results than using fixed parameters.

There is one advantage of HSP-r over RSaFS; that is, HSP-r achieves quicker solutions than RSaFS due to the fact that a forward search within the repairing phase explores much more states than HSP-r does. In general, HSP-r could be considered as a special case of RSaFS when a new state of a search is selected after performing only one-step forward investigation. If the time available for running a program is too short, HSP-r might achieve better solutions, since RSaFS cannot examine as many plans as HSP-r can.

9. CONCLUSSION

The simulation results indicate that for resolving a concrete type of planning problems like the scheduling problem that has been chosen as the case of study, RSaFS has achieved significant improvements in comparison with HSP-r, if they have the same time of solving. There are also other algorithms, e.g., a number of different algorithms based on graphplan (Blum & Furst, 1997), little different or modified from the graphplan. Due to complicated problems associated with maintaining and memorizing data as discussed in Section 2 are seen as an inappropriate method for solving the types of planning problems – planning in manufacturing - that are dealt in this work. For that reason these algorithms are not selected for comparison. The scheduling problem that has been chosen for simulation is a special type of the general planning problems. However, applying RSaFS to other applications could be the objective of future research.

Acknowledgement: This paper is partially supported by APVT and VEGA grant agencies under grants No APVT 51 011602 and VEGA 2/1101/21.

REFERENCES

Blum A. and Furst M. (1997): Fast Planning Through Planning Graph analysis. *Artificial Intelligence*, 90, (1-2), p.281-300.

Bonet B. and Geffner H. (2001): Planning as Heuristic Search. *Artificial Intelligence*,129, p.5-33.

Dang T.-Tung (2003): Dissertation work, 2003.

Hoffmann J. (2000): A Heuristic for Domain Independent Planning and Its Use in an Enforced Hill-Climbing Algorithm. *In Proc. the 12th Int. Symposium on Methodologies for Intelligent Systems*, p. 216-227.

Kumar V., Grama A., Gupta A., and Karypis A. (1994): Introduction to Parallel Computing. Design and Analysis of Algorithms. *The Benjamin/Cummings Publishing Company, Inc*., California, 1994. ISBN 0-8053-3170-0.

Nguyen X., Kambhampati S. and Nigenda R. (2002): Planning Graph as the Basis for deriving Heuristics for Plan Synthesis by State Space and CSP Search. *Artificial Intelligence*, 135, p.73-123.

Tran Viet D., Hluchý L., Nguyen Giang T. (2000): Parallel Program Model for Distributed Systems. *In: Proc. of 7th European PVM/MPI Users' Group Meeting*, Hungary, Springer Verlag, p.250-257.

Table 1: Simulation results

(β_1, β_2)	Experi. Num.	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*
1-0	HRS	180	168	192	183	181	188	185	198	178	180
0-1	HRS	155	108	147	115	142	155	165	162	121	120
1-0	RSFS-10	170	162	178	161	182	176	185	181	141	181
1-0	RSFS-all	181	169	192	165	181	178	183	198	141	181
0,5-0,5	RSFS-10	139	132	156	122	164	**119**	163	163	107	105
0,5-0,5	RSFS-all	135	132	134	122	180	155	175	**128**	140	106
0,3-0,7	RSFS-10	135	141	149	94	168	137	177	144	89	112
0,3-0,7	RSFS-all	146	121	**126**	106	111	141	171	144	95	**99**
0-1	RSFS-10	138	128	165	103	168	154	172	149	108	123
0-1	RSFS-all	**110**	**101**	137	**89**	**88**	122	182	131	**77**	106