A personal scheduling system using genetic algorithm and simple natural langague processing for usability

Quan T.D, Nguyen H.T, Long H.N.G , Luan M.T, Long T.T, Tien M.Q, Tho Q.T

JSKE research group about scheduling and AI
Ho Chi Minh City University of Technology

***Abstract*[1]— Scheduling in industry is developing in fast pace. However, to use those researches in personal scheduling, the scheduling theory must solve some novel specific problems. Scheduling task is originally combinatorial complex with many constraints. In this paper, we introduce about how we apply an adaptive genetic algorithm in solving the problem named split-able scheduling. Beside that, by applying NLP, the scheduling application is more interactive to end users. Moreover, genetic algorithm offers, in this scenario, a promising approach to problem solving, considering have good results. Therefore, the goal of this work is applying genetic algorithm (GA for short) with some changes from original redirect representation GA, which is implemented in industry before, to solve split-able scheduling problem in personal user perspective.**

Keywords—Evolutionary algorithm, genetic algorithm, Scheduling, Natural langage processing.

I. INTRODUCTION

Scheduling is a process of deciding how to allocate resources over time to perform a set of tasks in satisfying certain constraints [**13**,**4**,**5**,**16**,**19**,**2**]. It plays a very important role in helping people to make plans to use all their resources effectively and have optimal results. Because of that, it is used in many manufacturing and services industries. Scheduling is also very necessary in personal life, not only helps individuals in dealing with a numerous complicated task, but also reduces stress. Good scheduling skill will help people to create a good vision, manage risk, and design a good quality life that balance their personal and their professional work.

In current technology era, people have to deal with hundreds of tasks, emails, and many opportunities day by day. According to a productivity expert, most of us always have about 50 to 150 small tasks at any moment of time [**1**]. Besides that, each task has many attributes and constraints as deadline, duration, appropriate time for doing it more effectively… The situation that individuals cannot handle all their tasks effectively, and then will break their commitments with the others and even with themselves, is the main cause of stressfulness and anxiousness. To be successful in this world, we need the ability to decide swiftly what the most important things to do are, and what task can be eliminated, at any moment.

Current calendar applications such as Google calendar or Microsoft Office Outlook only provides a visual environment that helps people manually pick up available time blocks and arrange their tasks. The problem is that with so many tasks, constraints to be satisfied and especially because of the constantly changes in their real life, people become exhausted when continually rescheduling their tasks with that ineffective and time-consuming way. Consequently, people have to be very strict with their plan and therefore feel uncomfortable, or cannot trust the schedule and then give up.

Currently, there are many algorithms to solve the scheduling problems in industry. However, in personal scheduling, the problem has many different characteristics. Firstly, personal plan is always changeable. Although almost their tasks may have a due date or deadline, they do not have to happen at a fixed time and can be slide depend on new events of the real world. It makes the personal plan have to be flexible. Each task can be preempted by a new task which is more important than. Secondly, individual may want to break a large task into many small ones, but each split part of the task cannot last less than a specified minimum duration. This constraint about the duration of each part of a split-able task does not exist in industrial scheduling.

To solve that problem, the authors have proposed a new approach on personal scheduling using genetic algorithm with some specific heuristic, called split-able scheduling algorithm.

[1] Main author of this paper is Undergraduate student

In addition, to increase usability, the authors apply some *natural language processing* (NLP for short) techniques when processing input task for the system. Users can enter a task by typing a natural sentence, e.g.: "I will read book about Scheduling in 2h before next Saturday." or "I will research about AI about 10 hours next week.". The system then automatically parses that sentence, recognizes many properties of each task and passes these values to the scheduling algorithm. Our approach is implemented as an effective personal scheduling system known as the JSKE system.

The rest of this paper is organized in 5 Sections. Section 2 depicts the general architecture of the JSKE system with some main features. Section 3 illustrates the process of applying NLP to process task input. Section 4 is the main content of paper, which presents our proposed approach for using adaptive algorithm (GA split-able scheduling algorithm) in 'JSKE' system. Section 5 discusses about current status of our system. Finally, section 6 is the conclusion of the paper and the direction for our future research.

II. GENERAL ARCHITECTURE

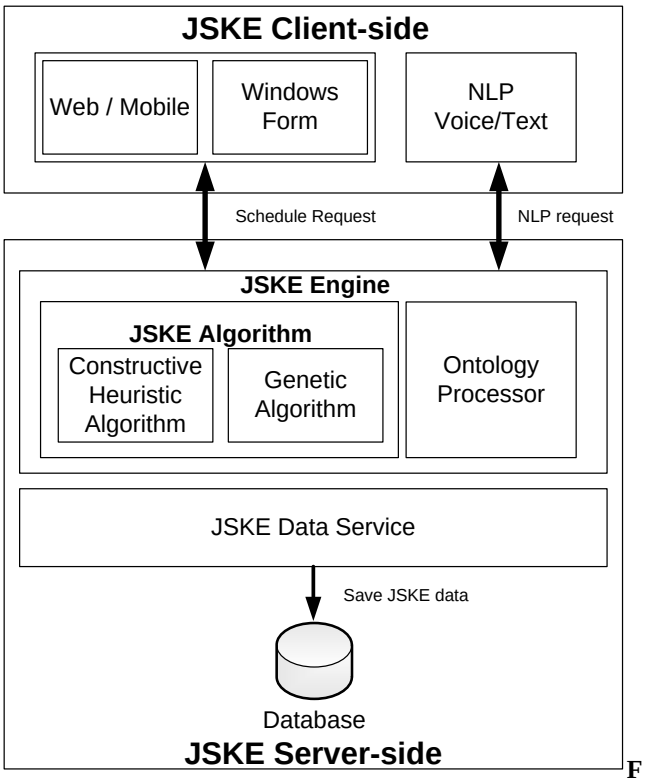

Figure 1 – JSKE Architecture

Figure 1 presents the general architecture of the JSKE system. We use *Passive Model-View-Presenter* **[20]** design pattern to implements the system based on Microsoft Technologies. According to Figure 1, the system has two main parts: one is the *Client-side* which concentrates on displaying and handling the user interface; the other is the *Server-side* that was built to process complex requests from *Client-side* like *Scheduling, Natural Language Processing, etc...*

Both *Client-side* and *Server-side* can be deployed on a single PC for an individual user or on a complex server system for multiple users. The JSKE system can also interact with third party systems like *Google Calendar* or *Microsoft Outlook* to get user specified data like *Calendar, Task List*, etc…

1. *JSKE Client-side*

The main function that *Client-side* holds is providing and handling user interface so that user can access to the system through *Web/Mobile* environment or *Windows Form* application environment. The JSKE Client-side consists of three parts.

The first part is the *Web/Mobile* which provides a user interface that can be displayed on browsers such as: Firefox, IE, Chrome, Safari, and Opera… These user interfaces were customized so that users can access them by using PC, Laptop or mobile devices (smart phones, mobile phones).

The second part is the *Windows Form* which provides a user interface for a standalone application that runs on *Windows* operation system.

The last part is the *NLP Voice/Text* which can be integrated into *Web/Mobile* or *Windows Form*. This NLP system provides friendly user access control by utilizing *Natural Language Processing*. It records all input from user under the form of plain text or voice, which we will implement in the future, then send to *Ontology Processor*.

2. *JSKE Server-side*

In this section, we explain more about the *Server-side* of our system. The *Server-side* consists of three main parts.

The first part, also the main part of the *Server-side*, is *JSKE Engine*. This Engine consists of *Constructive Heuristic Algorithm, Genetic Algorithm* and *Ontology Processor* which can handle all requests from user. In addition, it controls data access, authentication, authorization, privacy preserving for user data and encryption-decryption in the form of public private key and symmetric key as well.

- *JSKE Algorithm* manages all JSKE algorithms for scheduling purpose. It decides which classes of algorithm will be used then provides the data input and output for the algorithm. *JSKE Algorithm* was designed to support many algorithms which fit JSKE standards. This means that third party can develop their own algorithms then attach them to JSKE. There are two kinds of algorithm that *JSKE Algorithm* supports. One is *Constructive Heuristic Algorithm*, which was developed by using heuristic rules providing user with expected results in NP-times. The other is *Genetic Algorithm*, which optimizes the result from *Constructive Heuristic Algorithm* so that user can get a better schedule result.
- *Ontology Processor* analyzes user's queries which

are natural language by using *Ontology* (described in Section 3.)

The second part is *JSKE Data Service*, which provides methods for *JSKE Engine* to access data, which is stored at a database. The last part is Database, which stores User Data and System Configuration.

III. ONTOLOGY-BASED NATURAL LANGUAGE PROCESSING

In order to render users with a convenient communication environment, our system supports a mechanism that allows users to submit tasks to be scheduled using natural descriptions. To capture the semantics conveyed in those natural descriptions, we rely on an ontology-based approach presented in [**17**]. Basically, this process consists of the following steps:

- *Domain ontology acquisition*: A domain ontology which first needs defining reflects fundamental concepts and behaviors in the domain of discourse. Since the knowledge represented by ontology is machine-understandable, our system can make full use of the ontology to analyze the meaning of a simple natural phrase.
- *Ontological concepts and instances recognition*: In this step, the system recognizes ontological concepts and instances, based on what are well-defined in the ontology.
- *Ontological relations reconstruction*: From the recognized concepts and instances, the corresponding relations between them are reconstructed. This step also involves some techniques for inconsistency resolution.
- *Conceptual graph generation*: This final step generates a *conceptual graph*, which is a high-level knowledge representation, for further processing.

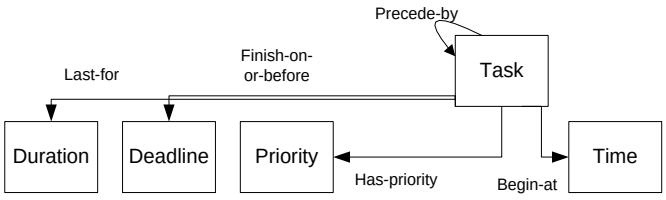

Figure 2 – A simple ontology

Let us demonstrate on how this technique works by the following example. In Figure 2, a simple ontology is given (the full ontology used on our research is presented in Appendix A). Suppose that the user submit the scheduled task in natural language as follows.

(T1): "*Meeting* of research group by *3pm tomorrow* – *urgent*"

Processed by the system, the mentions of *meeting*, *3pm tomorrow* and *urgent* are respectively recognized as instances of concepts *Task*, *Time* and *Priority*. Then, the ontological relations are extracted accordingly and the final conceptual graph is constructed as given in Figure 3.

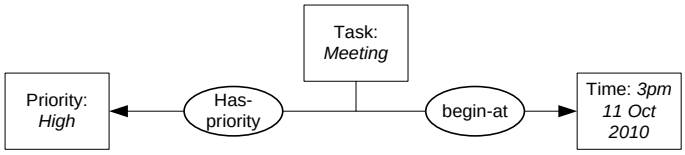

Figure 3 - A conceptual graph generated

Obviously, the generated conceptual graph provides structured and well-defined information for the subsequent task of scheduling. In case some significant information is missing, the system will query the user for further instructions or use default value.

Supposed that the user later gives another scheduled task as follows:

(T2): "After the *meeting*, find suitable *time* for *reading* course material in about *1 hour*".

Similarly, as the way the system processes task (T1), we have another conceptual graph generated as illustrated in Figure 3. In this time, the system will then realize that there is no specific time given and it needs to schedule suitable period for this task.

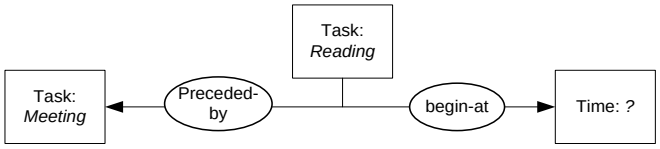

Figure 3. Another conceptual graph

IV. ADAPTIVE SPLIT-ABLE SCHEDULING ALGORITHM

In this paper, we consider the scheduling problem where a set of n split-able tasks $T=\{T_1, \ldots, T_n\}$ has to be scheduled without preemption on a single machine. There is another set of fixed tasks that has constant start time and end time $FT = \{FT_1, FT_2, ..., FT_m\}$. In manufacturing scheduling literature, these fixed tasks could be considered as non-available time windows (*noatw* for short). This constraint represents maintaining phase in which the tasks could not be executed.

Each task T_i is described by processing time p_i, release date r_i, due date d_i, weight w_i, and the smallest time duration of a split $minsplit_i$. We denote by $preced_i$ the set of tasks which must be completed before beginning of task T_i. Let C_i

be the completion time of T_i; $split_{ij}$ be the j-th split of task T_i.

The decision of schedule should define a task T_i according to one of two following situations:

- earliness if the task completes before its due date ($C_i \le d_i$),
- tardiness if the task completes after its due date ($C_i > d_i$).

The purpose of personal scheduling is not making strict disciplinary to user, but helping users analyze how they are dealing with life tasks and their productivity in completing them. One of the obvious criteria for measuring these things is the number of tardiness tasks in their schedule described by using the following notion:

$$U_j = \begin{cases} 0, \text{if } T_j \text{ is an earliness task} \\ 1, \text{otherwise} \end{cases}$$

The approach with this objective function aims to minimize number of important tardiness tasks as much as possible.

According to the classical three-field scheduling notations [**7**], the studied problem is noted as ***1*** **|** r_i**,** d_i**,, *prec, noatw, minsplit*$_i$ **|** $\sum w_i U_i$.

According to Karp, the scheduling problem ***1*** **|** **|** $\sum w_i U_i$ is NP-Hard [**12**]. Without *noatw* constraint and $r_i=0$ for all tasks T_i , a splitting task schedule for a one machine problem with monotone objective function can be polynomially transformed into a nonsplit-able task schedule without increasing the objective function. This implies that **1 |** ***minsplit*$_i$ **|** $\sum w_i U_i$ is NP-hard as well. Consequently, the considering scheduling problem is also NP-Hard.

Let see an example about the difficulty in solving our problem

- Input of task T_1:

$p_1 = 6$, $minsplit_1 = 2$, $r_1 = 0$, $d_1 = 12$,

$preced_1 = \emptyset$, $w_1 = 1$.

- Input of task T_2:

$p_2 = 4$, $minsplit_2 = 2$, $r_2 = 0$, $d_2 = 14$,

$preced_2 = \emptyset$, $w_2 = 1$.

The time axis is represented in Figure 4 (the hatched part is employed by a fixed task, which starts at time t=5 and ends at time t=9).

Figure 4 – Fixed task constraint

Two tasks T_1 and T_2 could be scheduled as follows (demonstrated in Figure 5):

- Processing order: T_1 first and then T_2,
- T_1 is divided into two splits: $split_{11} = 4$, $split_{12} = 2$, then $C_1 = 11$, $U_1 = 0$,
- T_2 has only one split: $split_{21} = p_2 = 4$, then $C_2 = 15$, $U_2 = 1$.

The objective function is then equal to 1.

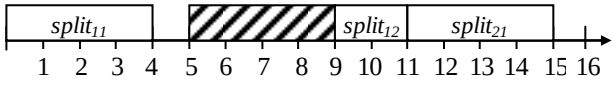

Figure 5 – Case of $T_1 << T_2$

With respect of execution task order (all splits respect this order, e.g. if $T_1 << T_2$ then all splits of T_1 are process before all splits of T_2), T_2 also executed before beginning to process task T_1. The result is as follows.

- Processing order: T_2 first and then T_1,
- T_2 has one split which starts at time $t=0$, and finishes at time $t=4$, then $U_2 = 0$,
- T_1 also has one split which starts at time $t=9$, and finishes at time $t=15$, then $U_1 =1$.

The objective function is then equal to 1.

Figure 6 – Case of $T_2 << T_1$

However, the optimal solution of this instance could be defined by split both of tasks. The sequence of splits in optimal solution is ($split_{11}$, $split_{21}$, $split_{12}$, $split_{22}$) with $split_{11}$ = 60, $split_{12}$ = 50, $split_{21}$ = 15, $split_{22}$ = 25. Then, $C_1 = d_1 =$ 12 and $C_2 = d_2 = 14$. The objective function is consequently, equal to zero. The execution sequence is presented as in Figure 7.

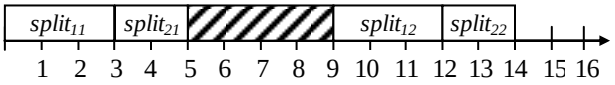

Figure 7 – Optimal solution

By studying the above example, finding a precise algorithm for our problem is not feasible. In this paper, we propose an adaptive *genetic algorithm* (GA) to solve the problem. This GA algorithm could be cooperated with a constructive heuristic. The constructive heuristic and GA algorithm will be described as follows.

A. *Constructive heuristic*

For solving the considering scheduling problem, we propose firstly an algorithm called "***Constructive heuristic method***". We call this ***"Constructive heuristic"*** because the

solving problem is completely NP-hard, therefore in the present, it's difficult to find a mathematical dominant rule for this heuristic. Consequently, the following heuristic only depends on some reasonable rules of human thinking for this problem.

We define a pair $[s_i, e_i]$ is annotated for a time block between two consecutive fixed tasks. Initially, set A is a set of these time blocks $A = \{[s_1, e_1], [s_2, e_2], ... [s_n, e_m]\}$.

We defined a function named *tryFill(task)*. This function tries to fit a task into time blocks as much as possible. Notice that if we try to fill a time block by a split of the task, we must consider whether the remaining of the task satisfies the minimum split constraint. If not, we must resize the split in order the remaining of the task satisfies the minimum split constraint. Let us see these following examples:

Task, block	[0, 7]	[8, 11]	[14, 19]	Note
P_1 = 11, $minsplit_1$ = 3	$Split_1$ = [0, 7] Dur_1 = 7	$Split_2$ doesn't fit in this block	$Split_2$ = [14,18] Dur_2 = 4	Correct
P_2 = 10, $minsplit_2$ = 3	$Split_1$ = [0,7] Dur_1 = 7	$Split_2$ = [8, 11] Dur_2 = 3		Correct
P_3 = 8, $minsplit_3$ = 3	$Split_1$ = [0, 7] Dur_1 = 7	$Split_2$ = [8, 9] Dur_2 = **1**		Violate[2]
P_3 = 8, $minsplit_3$ = 3	$Split_1$ = [0, 5] Dur_1 = 5	$Split_2$ = [8, 11] Dur_2 = 3		correct

Let *timeWalker* be a variable that indicate a specific point of time. Set B is a set of candidate tasks could process at *timeWalker*.

We define function *getCandidateTask(time)* be a function which returns all candidate tasks at a time point. Set C is a set which includes all scheduled tasks. The heuristic algorithm is implemented as follow:

- Step 1: create set A include all time blocks. *timeWalker* = *0*. Set $C = \emptyset$.
- Step 2: according to the task set T, we use *tryFill* function to find tasks that surely is late, denote set T' and delete from T.
- Step 3: set B = *getCandidateTask(timeWalker)* from task set T.
- Step 4: choose from set B a task T_i which has a minimum value of $(\frac{d_i}{w_i} . \frac{1}{F_i})$ among the tasks of B, i.e.

$$\frac{d_j}{w_j}.\frac{1}{F_j} = \min\left(\frac{d_j}{w_j}.\frac{1}{F_j}\right) \forall T_j \in B$$

[2] Because $minsplit_3 = 3$ but $dur_2 = 1$

With F_i is the ratio of a task splits filling its all time blocks. Easy to see that max value of F_i is 1. More precisely, let task T_i filled some blocks in set A and start time of the first block is *timeWalker*: $[timeWalker, e_{[h]}]$; $[s_{[h+1]}, e_{[h+1]}]$; ... ; $[s_{[h+a]}, e_{[h+a]}]$ (a is the number of block, which has duration smaller than $minsplit_i$).Therefore, task T_i is divided into (k-h + 1) splits which have corresponding durations: dur_1 , dur_2 , ..., dur_{k-h+1}. They are calculated by *tryFill* function (see the table above for more details). We have:

$$F = \frac{dur_1 + dur_2 + ... + dur_{k-h+1}}{e_{[h]} - timeWalker + e_{[h+1]} - s_{[h+1]} + ... + e_{[k-h+1]} - s_{[k-h+1]}}$$

$$= \frac{p_i}{e_{[h]} - timeWalker + e_{[h+1]} - s_{[h+1]} + ... + e_{[k-h+1]} - s_{[k-h+1]}}$$

If F_i is small, it means that task T_i can easily find some blocks to fit in. So task T_i should be filled after other tasks, for example task T_j, whose F_j is larger.

- Step 5: put task T_i into set C and delete from set T, update *timeWalker* be the end of the last split of task T_i. Update set $A = \{ [timeWalker, e_{[k+a]}] , [s_{[k+a+1]} , e_{[k+a+1]}], ..., [s_n, e_n] \}$. Go back to step 3 if there exists any unscheduled task in set T.

By using this heuristic algorithm, we could create a good initial population for the genetic algorithm which will be introduced as follows.

B. *Genetic algorithm*

The GA was first introduced by [**11**]. It is an adaptive search algorithm, which encompass semi-random search method whose mechanism based on the evolution process in nature. In contrast to other search algorithms, GA works in a population, not only with an individual. Each individual in GA is assigned a fitness value, which is objective function value in scheduling problems. A reproduction step is used for producing a next generation population which has better fitness values than old population. In this step, GA applies cross over and mutation method on current population. However, beside some advantages of GA, we must analyze carefully all specific properties of solving problem and decide on a proper presentation, an objective function, genetic operators, genetic parameter and good initial population to improve the quality of searching process. A survey of [**8**] summarizes more about some techniques in GA. In addition, the original GA uses binary values to represent a chromosome. However, in some scheduling problems, GA with real values has been proven out performance than binary GA [**9**]. The following sub-sections describes how we apply GA into our problem.

1. Overview:

The process of applying GA has following main steps:

- Step 1: Generate first population. In this step, GA

cooperates with the constructive heuristic we mentioned before to get a good initial population as near the optimum as possible.

- Step 2: Loop until the terminated condition is true
  - Choose chromosomes to mate.
  - Mate chromosomes by crossover operator.
  - Mutate some random chromosomes to avoid premature convergence.

The process is clearly illustrated by below picture:

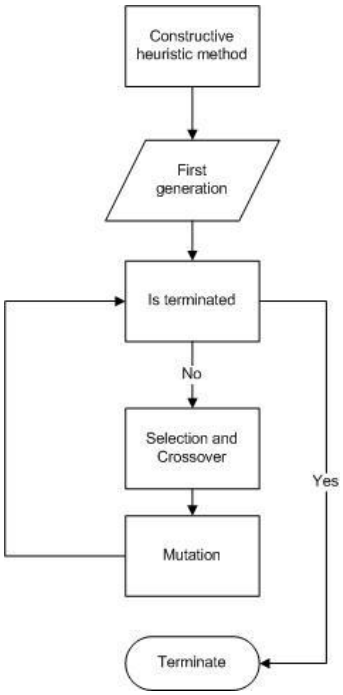

Figure 8 – GA process overview

Next, we will explain more details about some important processes in our GA.

2. Representation, decode function and fitness function:

Each individual in GA is represented as a chromosome. The performance of GA algorithm depends much on how we represent a schedule result. There are many ways for mapping a schedule to a chromosome, e.g. operation-based representation, task-based representation, preference list-based representation, task pair relation-based representation, priority rule-based representation, disjunctive graph-based representation, completion time-based representation, machine-based representation, which are provided in in-depth survey of Cheng et al.[**6**].

In this paper, we propose to use *Random keys* (RK) representation [**3**] with some minor changes for being more appropriate to the current problem. The advantages when using RK is analyzed more in [**15**]. We define $numOfSplit_j$ is the maximum number of splits each task could be split. However, in a schedule result, it's obvious that a task does not always has $numOfSplit_j$ splits.

Each element in a chromosome represents a *priority* of a corresponding task split. This *priority* definition has two impacts on the position and processing time of the corresponding split. Firstly, it represents the factor of split duration compared with other splits of the same task. More clearly, the higher *priority* a split has, the longer that split last. Secondly, *priority* also shows the importance of that split compared with other splits, which are feasible for the current scheduled time, from other tasks. That means in the decoding process the split which has the highest *priority* will be chosen to put as the next split in the result schedule.

Firstly, we implement an algorithm **(1)** for calculating the processing time of each task split :

- **P_j is the set of possible task j's split priority which are extracted from a chromosome. avg_j is the average value of all elements in set P_j .**
- **Let S_j is the array which will include the duration of task j's splits in the result schedule.**
- **Set sumPriority = Sum of elemenets in set P_j**
- **For each split priority sp_k in P_j**

  If $sp_k > avg_j$ then add value:

  $(sp_k$ / sumPriority * p_j) to S_j

However, we can see a flaw in the algorithm above as there aren't cases that a task j has $numOfSplit_j$ in the decoded schedule result because there's always at least one sp_k that is less than avg_j. To overcome this, for each task, we add another priority element,which is called *fake priority*, to the chromosome therefore any task could have the maximum number of splits in a decoded schedule result. Now we implement the main decode function:

- **Step 1: For each task j , we use algorithm (1) to decode from the chromosome to S_j.**
- **Step 2: For all feasible split at the current time which are the first element of all S_k , choose the split with the highest priority and set as the next split.**
- **Sterp 3: if there are still some tasks that aren't decoded, go back to step 2.**

A feasible task split at a time point is a split that can be processed at that time. We will go through with an example:

Task a: $p_a = 7$, minsplit = 2, $r_a = 0$ (we don't need duedate in the decoding process)

Task b: $p_b = 4$, minsplit = 2, $r_b = 2$

Task c: $p_c = 9$, minsplit = 3, $r_c = 0$, preced = {a}

The decoding chromosome is:

0.7 0.1 0.3 0.1 | 0.4 0.2 0.1 | 0.5 0.4 0.6 0.3

By applying the algorithm **(1)** we have:

$S_a = [2 , 5]$; $P_a = \{0.7 , 0.3\}$

$S_b = [4]$; $P_b = \{0.4\}$

$S_c = [4, 5]$; $P_c = \{0.5 , 0.6\}$

By applying the decode algorithm we have the schedule result:

$Split_{a1}$ (4 time units) - because 0.7 is the current highest priority

→ $split_{b1}$ (4 time units) – although $split_{c1}$ has the current highest priority, it is not feasible at the current time because task a haven't finished yet.

→ $split_{a2}$ (5 time units)

→ $split_{c1}$ (4 time units)

→ $split_{c2}$ (5 time units).

The final result is:

$split_{a1} \rightarrow split_{b1} \rightarrow split_{a2} \rightarrow split_{c1} \rightarrow split_{c2}$.

3. Create initial population:

In original theory of GA, the first population is created by using a random method. However, with some complex scheduling problems, other authors usually apply some heuristic methods for creating initial population. In our context, we implemented a heuristic algorithm that shows quite good results.

4. Selection, crossover and mutation:

a. Selection:

In GA, we specify a keeping rate parameter X_{rate}, which is the percent population of a generation, is kept for the next generation. The chromosomes, which are kept, have highest fitness values. Choosing a good keeping rate is an experimental process. If X_{rate} is too high the time for convergence is long. In contrast, if X_{rate} is too low, the numbers of good traits in the next generation is limited. By changing some value of X_{rate}, we know that 50% for X_{rate} is good enough. After choosing keeping chromosomes, the main GA process continues with Roulette Wheel Weighting algorithm for selecting chromosomes, which will run through a next crossover phase.

b. Crossover

As mentioned above, chromosomes are represented by real values. Hence, we will use the crossover technique in [**10**] instead of crossover method in original binary GA.

c. Mutation:

Random mutations alter a certain percentage of the bits in the list of chromosomes. Mutation is the second way a GA explores a cost surface. It can introduce traits not in the original population and keeps the GA from converging too fast before sampling the entire cost surface. A single point mutation in original GA changes bit 0 to 1 and vice versa. In real-value GA, we utilize mutation method in [**16**] for chromosomes.

V. CURRENT STATUS OF SYSTEM

The standalone JSKE System currently supports the Constructive Heuristic Algorithm, Genetic Algorithm and NLP techniques to schedule and input task.

After the completeness of the Window Form JSKE system, we are currently developing the *Web/Mobile* JSKE system on web platform, base on the current *JSKE Engine*. This system will concentrate on optimizing the Genetic Algorithm performance. Besides that, it also pays attention to NLP techniques, which support voice recognition and natural query to control the system.

Here are some screenshots of the current JSKE System that runs on Windows platform. These screenshots are the most important part of the current JSKE System.

The first screen shot is the one in Figure 9. This describes the way user can input their tasks using NLP technique. They must first put plain text that describes their tasks in the "*Provide Task Information*" text area. After that they click on "Send to NLP" button in order to send the task to the NLP Engine. If the plain text is successfully processed, then we will see a screenshot like Figure 10. In fact, Figure 10 is just the most important part of the "*Task Form*", which is a *i* that was written for user to input their task manually.

After entering all tasks by using NLP Form in Figure 9 or Task Form in Figure 10, we can see the screenshot like Figure 11. This is the main User Interface of JSKE System. We can use this User Interface to manage our task list, configure the system, tracking for Diary and task progress and most of all run the schedule function. The schedule function will send appropriate data to *JSKE Engine* to schedule an expected calendar for user. After the schedule is completed, user can view the schedule result in the form of a calendar in Microsoft Outlook like Figure 12.

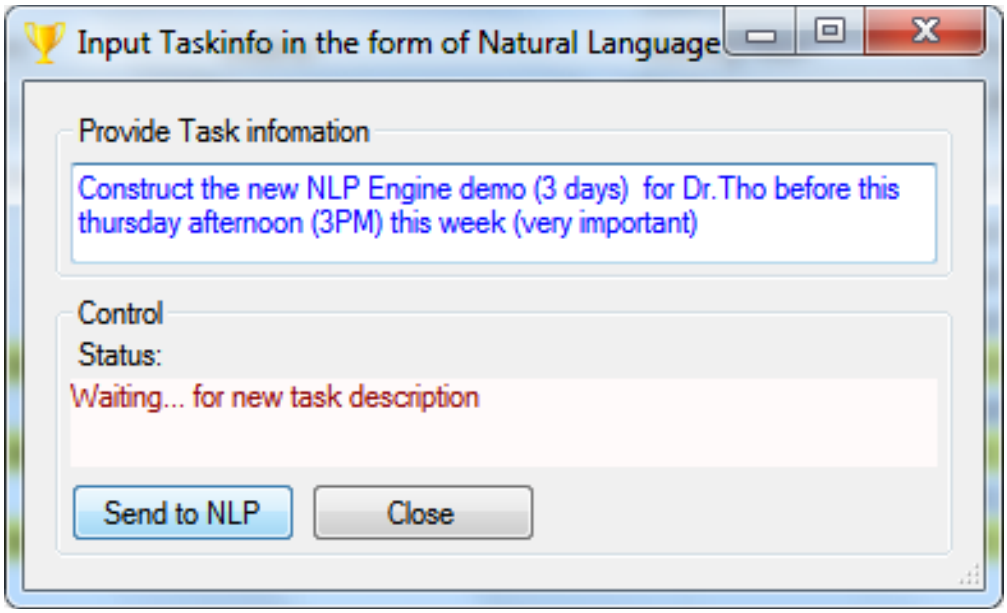

Figure 9 - Input task using NLP

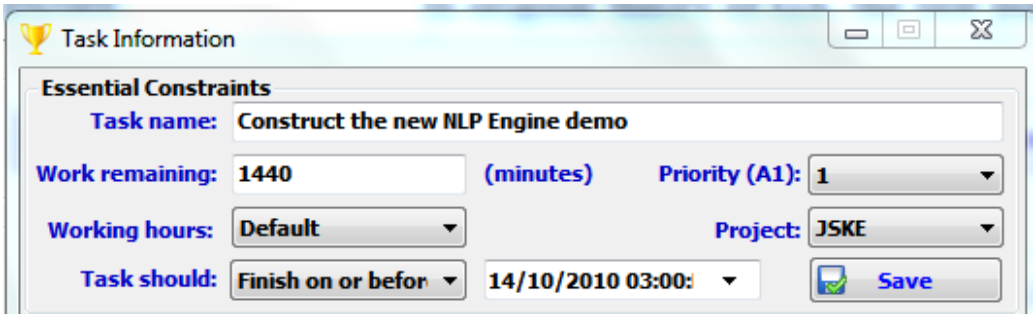

Figure 10 - The most important part of Task Form

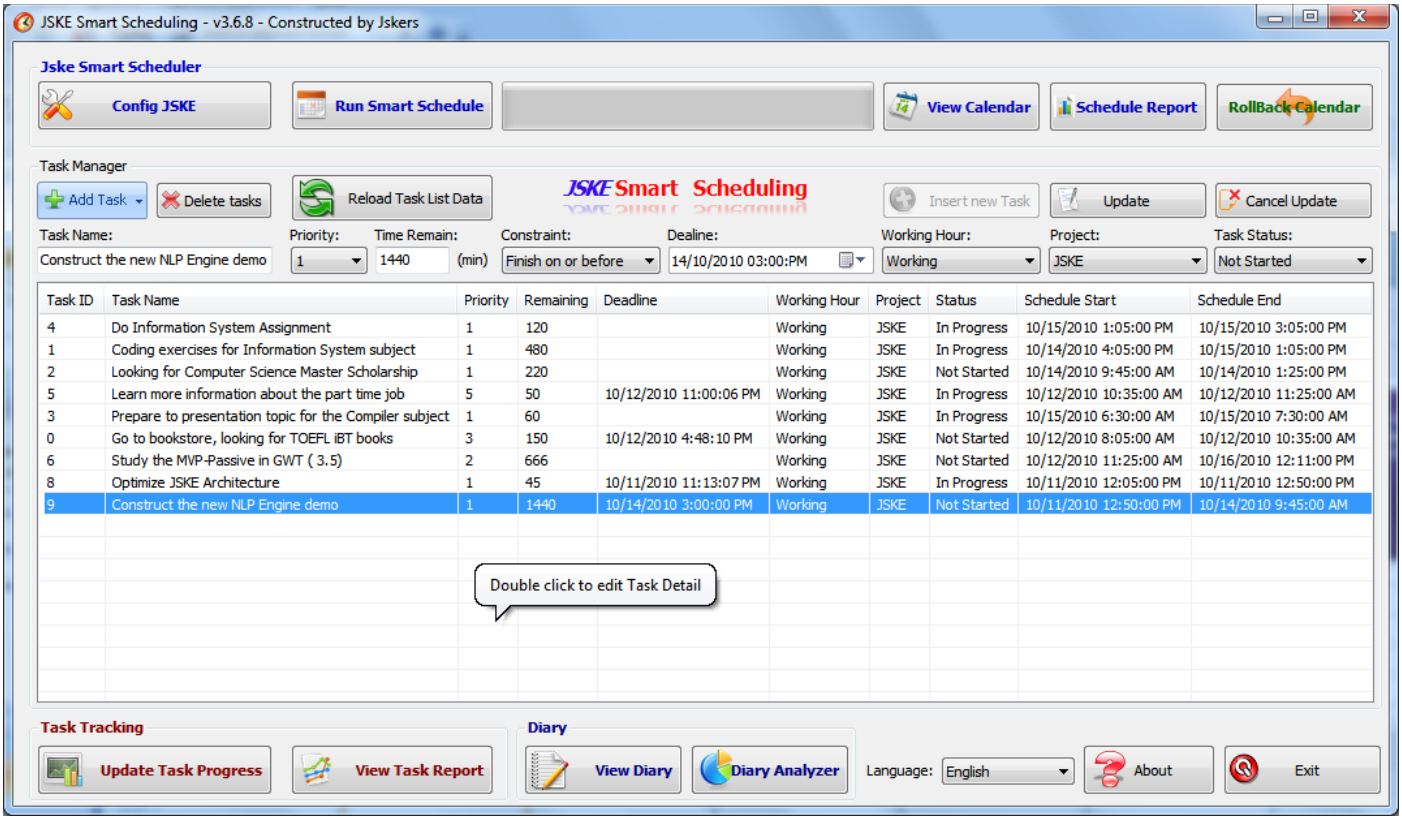

Figure 11 - Main UI of Standalone JSKE System

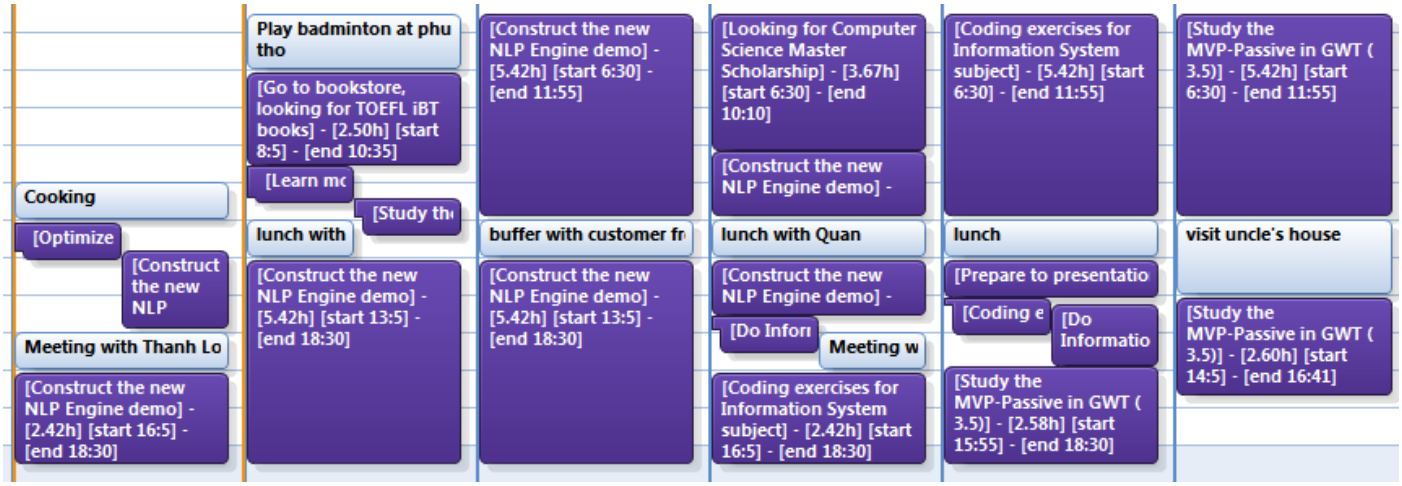

Figure 12 - The output calendar on Microsoft Outlook 2007

VI. CONCLUSION

By applying scheduling theory in personal scheduling, JSKE system helps users to organize their task effectively. However, scheduling tasks is a boring activity because without computer aid, users must repeatedly enter their task information. In future, we will try to apply more AI technologies such as data mining, decision support system into JSKE to improve user's experience and scheduling results. Moreover, one of the most challenges in personal scheduling is rescheduling each time user add one more task as fast as possible but also keep the good quality of the old schedule. Therefore, another considering research focuses on rescheduling using an evolutionary algorithm.

REFERENCES

[1] D. Allen, "Getting things done, the art of Stress-free productivity", Penguin Group, pp 41, 2003.

[2] K.R. Baker and D. Trietsch, Principle of sequencing and scheduling, Wiley, 2009.

[3] J.C. Bean, "Genetics and Random keys for Sequencing and Optimization", Journal of computing, vol. 5, pp. 154-160, 1994.

[4] J. Blazewicz, K. Ecker, E. Pesch, G. Schmidt, and J. Weglarz. Handbook on scheduling : from theory to applications. Springer, 2007.

[5] P. Brucker, Scheduling algorithms, 4th edition, Springer-Verlag, Berlin, Germany, 2004.

[6] R. Cheng, M. Gen, and Y. Tsujimura, "A tutorial survey of job-shop scheduling problems using genetic algorithms—i. representation," Computers and Industrial Engineering, vol. 30, no. 4, pp. 983–997, 1996.

[7] R.L. Graham, E.L. Lawler, J.K. Lenstra, and A.H.G. Rinnooy Kan. "Optimization and approximation in deterministic sequencing and scheduling : a survey", Annals of Discrete Mathematics, vol. 5, pp. 287–326, 1979.

[8] D.E. Goldberg , Genetic algorithms in search, optimization and machine learnin" , Addison Wesley, 1989.

[9] E.Hart, P. Ross and D. Corne, "Evolutionary scheduling: a review", Genetic Programming and Evolvable Machines, vol. 6, no. 2, pp. 191-220, 2005.

[10] R.L. Haupt and S.E. Haupt, Practical Genetic Algorithm, Wiley-Interscience (2nd ed.), 2004.

[11] J.H. Holland, "Adaption in natural and artificial systems", Technical report, University of Michiga, 1975.

[12] R.M. Karp, "Reducibility among combinatorial problems". In *Complexity of computer computations (Proc. Sympos., IBM Thomas J. Watson Res. Center, Yorktown Heights, N.Y., 1972)*, pp. 85-103. Plenum, New York, 1972.

[13] J.Y-T. Leung, Handbook of scheduling : algorithms, models, and performance analysis. Computer and information science series, Chapman and Hall/CRC (ed.), Boca Raton, Florida, 2004.

[14] J.M. Moore, "An n job, one machine sequencing algorithm for minimizing the number of late jobs", Management Science, vol.15 pp.102–109, 1968.

[15] B. Norman and J.C.Bean, "Random keys genetic algorithm for scheduling : Unabridged version", Internal Report, Department of Industrial and Operation engineering, University of Michigan, 1995.

[16] M. Pinedo, Scheduling : theory, algorithms, and systems. 2nd edition, Precentice Hall, Upper Saddle River, New Jork, USA, 2002.

[17] T.T. Quan and S.C. Hui, "Ontology-based Natural Query Retrieval using Conceptual Graphs", In Proc. of Tenth Pacific Rim International Conference on Artificial Intelligence (PRICAI 08), Vietnam, 2008.

[18] L. Schrage, "A Proof of the Optimality of the Shortest Remaining Processing Time Discipline", Operations Research, vol. 16, no. 3, pp. 687-690, 1968.

[19] V. T'kindt and J-C. Billaut, Multicriteria scheduling : theory, models and algorithms, Springer (2nd ed.), 2006.

[20] Microsoft :"Model-View-Presenter Pattern"
http://msdn.microsoft.com/en-us/library/ff647543.aspx

APPENDIX A – THE WORKING ONTOLOGY

Conceptual schema of the working ontology constructed for this research

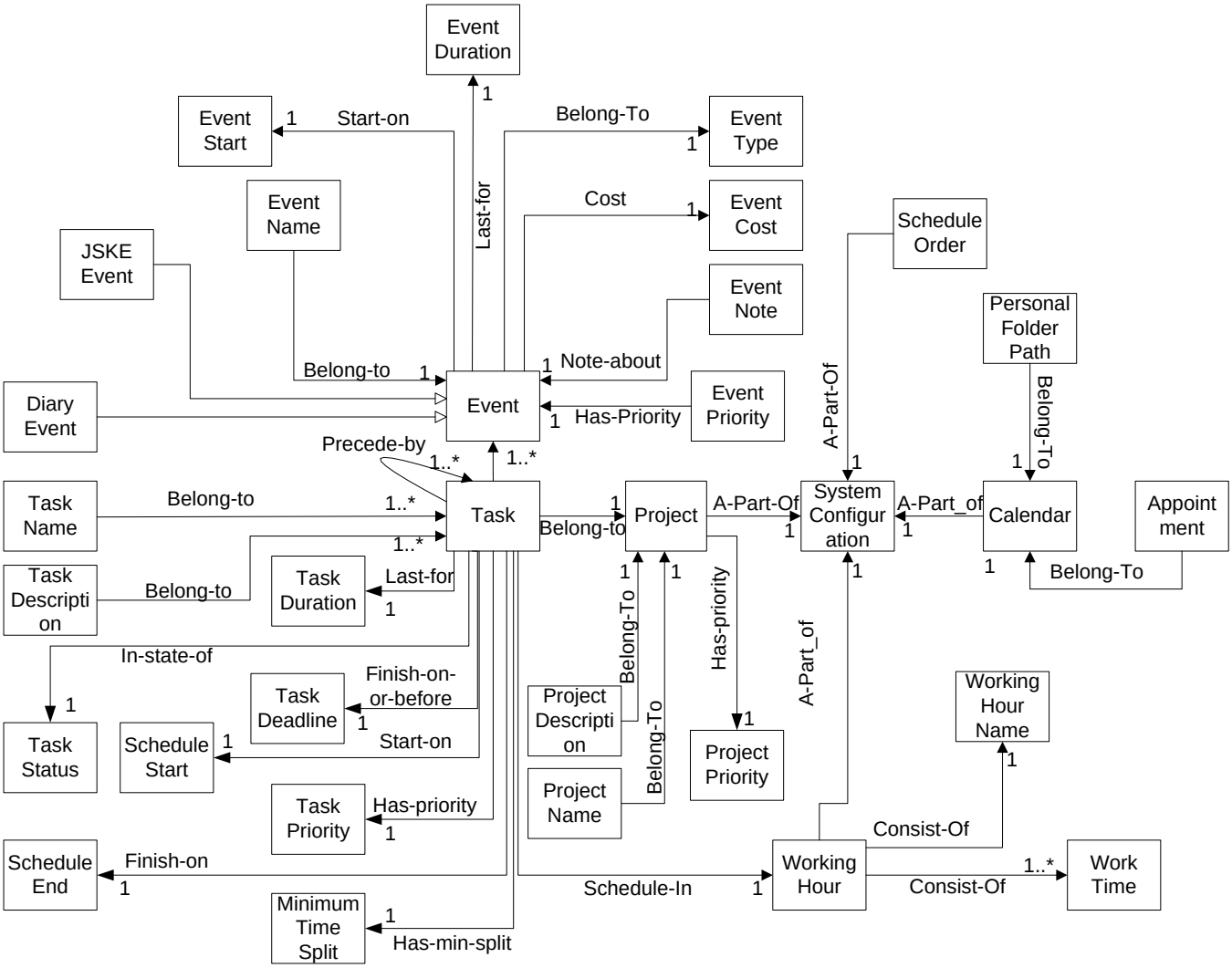

Figure 11 – Conceptual schema of the current working ontology