

CZECH TECHNICAL UNIVERSITY IN PRAGUE
FACULTY OF ELECTRICAL ENGINEERING



Graduation theses

Heuristic Algorithms for Project Scheduling

Prague, 2008

Author: Peter Kovalčík

Declaration

Hereby I declare that I wrote this diploma thesis completely autonomously, and that I cited all of information sources which were used during entire development.

Prague, date 18.1.2008


signature

Thanks

I would like to thank everybody who helped me either directly or indirectly during the working on this thesis. Especially I want to thank to Ing. Premysl Sucha, Ph.D, the thesis supervisor, who always gratefully supported me and led me through the thesis up to its end.

Many thanks belong to my parents and my close friends, who have been supporting me and making a great working atmosphere during my studies. Special thanks goes to my mother Eva, who supports me all the time and who is a respectable person for me.

Abstract

The thesis is concerned about implementation of An Effective Algorithm For Project Scheduling With Arbitrary Temporal Constraints which is a scheduling algorithm solving mostly project management problems, where total project length needs to be minimized. The algorithm has been implemented and modified in Mathworks Matlab environment. The modifications have been done in order to increase the algorithm speed at the price of its less efficiency. The modified algorithm has been implemented into a Java open-source project manager software, GanttProject. The first half of the thesis covers the theory about the scheduling essentials and about the implemented algorithm. The second one closely describes the algorithm implementation process into the both environments and represents the algorithms experimental results.

Abstrakt

Diplomová práca sa zaoberá implementáciou Efektívneho Algoritmu pre Rozvrhovanie Projektov s Ľubovoľnými Dočasnými Obmedzeniami, čo je rozvrhovací algoritmus riešiaci hlavne projektovo-manažérske problémy, kde celková dĺžka projektu musí byť čo najkratšia. Algoritmus bol implementovaný a modifikovaný v prostredí Mathworks Matlab. Modifikácie algoritmu boli vykonané s cieľom urýchliť výpočetnú dobu algoritmu aj za cenu straty jeho efektivity. Modifikovaný algoritmus bol implementovaný do Java open-source aplikácie GanttProject, určenej pre riadenie projektov. Prvá polovica práce pokrýva teóriu základných pojmov rozvrhovania a teóriu popisujúcu implementovaný algoritmus. Druhá časť detailne popisuje postup implementácie algoritmu do oboch prostredí a prezentuje výsledky experimentov, na ktorých bol algoritmus testovaný.

Katedra řídicí techniky

Školní rok: 2006/2007

ZADÁNÍ DIPLOMOVÉ PRÁCE

Student: Peter K o v a l č í k

Obor: Technická kybernetika

Název tématu: Heuristické algoritmy pro rozvrhování projektů

Zásady pro vypracování:

1. Najděte vhodný heuristický algoritmus pro rozvrhování projektů.
2. Otestujte vybraný algoritmus v Matlabu.
3. Změřte rychlost vybraného algoritmu.
4. Najděte vhodný nástroj pro rozvrhování projektů s přístupnými zdrojovými kódy.
5. Naimplementujte vybraný algoritmus do tohoto nástroje a otestujte jej.

Seznam odborné literatury:

Tristan B. Smith and John M. Pyle: *An Effective Algorithm for Project Scheduling with Arbitrary Temporal Constraints*. The Nineteenth National Conference on Artificial Intelligence (AAAI-2004). 2004.

Vedoucí diplomové práce: Ing. Přemysl Šůcha

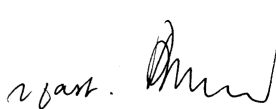
Termín zadání diplomové práce: zimní semestr 2006/2007

Termín odevzdání diplomové práce: leden 2008

prof. Ing. Michael Šebek, DrSc.
vedoucí katedry



prof. Ing. Zbyněk Škvor, CSc.
děkan



V Praze dne 21.02.2007

Contents

Table of Figures	viii
Table of Tables	x
1 Introduction	3
1.1 Motivation	3
1.2 Problem statement	4
1.3 Related work	7
1.3.1 Scheduling algorithms	7
1.3.1.1 Branch-and-Bound algorithms	8
1.3.1.2 Linear programming	8
1.3.1.3 Heuristics	9
1.3.2 Scheduling tools	10
1.4 Contribution	12
2 RCPSP/max Problem	15
2.1 Introduction to scheduling	15
2.2 RCPSP/max problem	17
3 Squeaky Wheel Optimization	19
3.1 Squeaky Wheel Optimization	19
4 An Algorithm For Project Scheduling	23
4.1 Algorithm Implementation	24
4.2 Own Algorithm Modifications	35
4.2.1 Prioritization phase modifications	36
4.2.2 Reducing the code	37

5	Project Management Software, GanttProject	39
5.1	Project Management	39
5.2	Project Management Software	40
5.3	GanttProject	45
5.3.1	GanttProject structure	46
5.3.2	Scheduling algorithm implementation	49
5.3.2.1	Create a menu item to invoke the Resource Leveling function	49
5.3.2.2	Create a Java class to handle the event when the Resource Leveling function is invoked	49
5.3.2.3	Create a Java class implementing the scheduling algo- rithm function	52
5.3.2.4	Involve the newly created classes to the overall GanttPro- ject structure	52
5.3.3	Compiling and running GanttProject	53
5.3.4	GanttProject vs. Microsoft Project 2007	56
6	Experimental Results	59
6.0.1	Generating the instances	59
6.0.2	Testing the instances	61
6.0.2.1	Running the benchmarks	61
6.0.2.2	The benchmark results	62
7	Conclusions	67
	Bibliography	70

List of Figures

1.1	Task-on-Node graph	4
1.2	Example: Building a house, task-on-node graph	6
1.3	Example: Building a house, resulting schedule	6
1.4	Microsoft Project 2007	10
1.5	Microsoft Project, two-task problem example	11
1.6	Sciforma Corporation PS8	11
2.1	Graphic representation of task parameters	15
2.2	Two tasks with generalized precedence constraints	17
3.1	Construct/Analyze/Prioritize cycle	19
3.2	Example task-on-node graph	20
3.3	The first SWO iteration schedule	21
3.4	The second SWO iteration schedule	22
3.5	The third SWO iteration schedule	22
4.1	Initial priorities calculation	26
4.2	Two tasks with generalized precedence constraints	27
4.3	EST_i calculation, predecessors order	29
4.4	LST_i calculation, successors order	29
4.5	Bulldozing method example, task-on-node graph	31
4.6	Bulldozing method example	31
4.7	Bulldozing method example	32
4.8	LeftBulldozing method example	34
5.1	The Project Management Triangle	40
5.2	Example 5.1: resource Group A overallocation	42
5.3	Example 5.1: resource Group B overallocation	43
5.4	Example 5.1: Group A, Resource Leveling function result	43

5.5	Example 5.1: Group B, Resource Leveling function result	44
5.6	The basic GanttProject functions	46
5.7	The GanttProject structure	47
5.8	The relationship between the <i>ResourceLevelingAction</i> class and the <i>ResourceLevelingAlgorithm</i> class	51
5.9	The call sequence between the <i>ResourceLevelingAction</i> class and the <i>ResourceLevelingAlgorithm</i> class	52
5.10	Involving the new classes into GanttProject structure	53
5.11	Create a new project from CVS	54
5.12	Checkout the project from CVS repository	55
5.13	Running GanttProject	56
5.14	Microsoft Project, two-task problem example	57
5.15	MS Project 2007, Resource Leveling function example	57
5.16	GanttProject, Resource Leveling function example	58
6.1	Progen/Max	60
6.2	Scheduling Algorithm Tester	62
6.3	The C_{max} comparison	63
6.4	The C_{max} comparison with higher percentage of maximal time lags	63
6.5	The C_{max} comparison of ALG and modified ALG for high scale problem instances	64
6.6	The <i>CPU time</i> comparison	65
6.7	The <i>CPU time</i> comparison with higher percentage of maximal time lags	65
6.8	The <i>CPU time</i> comparison of ALG and modified ALG for high scale problem instances	66

List of Tables

List of Abbreviations

Abbreviation	Term
BaB	Branch-and-Bound
CPU	Central Processing Unit
CVS	Concurrent Versions System
GUI	Graphical User Interface
GPL	General Public License
ILP	Integer Linear Programming
JDK	Java Development Kit
JRE	Java Runtime Environment
MS	Microsoft
RCPSP	Resource-Constrained Project Scheduling Problem
RLP	Resource Leveling Problem
SWO	Squeaky Wheel Optimization

Notation	Term
C_{max}	Makespan - total project length
EST	Earliest Task Start Time
LST	Latest Task Start Time
m	Number of resources
n	Number of tasks
p	Task processing times
S	Task start times
T	Task
W	Precedence matrix with general precedence constraints

Chapter 1

Introduction

1.1 Motivation

Nowadays, scheduling becomes more and more important part of our life. Many people face scheduling problems every day, e.g. at the airport (someone is responsible for assigning the planes to gates), in the hospital (creating nurse worksheets), at the constructions (creating workers worksheet, construction process plan) and so on. In general, *scheduling* is the process of assigning a set of tasks (or commonly called jobs) to a set of resources (humans, machines, processors, etc). The result is a timetable, schedule or plan. Before defining scheduling more precisely, let us give an intuition about typical applications.

One of the typical applications is Project management. Project management is a process of organizing and managing resources to achieve a specific goal (e.g. minimizing total project length, minimizing total cost, etc). A project might be a building construction, a chemical process or a computer system implementation.

Scheduling also plays an important role in computer science. The fast expansion of use of computers in public and private sector not only provided a tool to practically solve scheduling problems, but also created a new source of these problems itself: problems that arise in computer control (e.g. software scheduling problems, parallel processing, computer networks etc).

As we mentioned before, there are typical scheduling applications, each of this applications contains scheduling problems that need to be solved. To find a solution, scheduling algorithms are used. The result is the schedule that achieves required goal. This thesis is concerned about implementation of *An Effective Algorithm For Project Scheduling With Arbitrary Temporal Constraints* (Smith a Pyle, 2004), which is a scheduling algo-

rithm solving mostly project management problems, where total project length needs to be minimized. The algorithm has been implemented and modified in Mathworks Matlab environment. The modifications have been done in order to increase the algorithm speed at the price of its less efficiency. The modified algorithm has been implemented into GanttProject (<http://ganttproject.biz/>), which is a Java based open-source project manager software.

1.2 Problem statement

The scheduling problem is given by a list of tasks, a list of precedence constraints and a list of available resources. Problem should be represented by task-on-node graph¹ (see Figure 1.1).

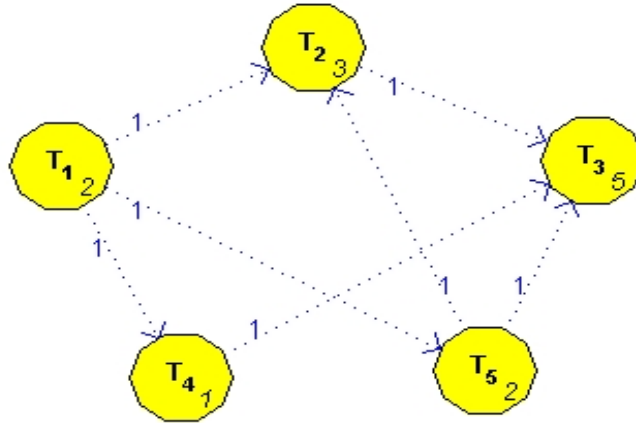


Figure 1.1: Task-on-Node graph

There can be two types of precedence constraints:

- The (classic) precedence constraints (shown in Figure 1.1), constraints where each arc between two nodes is weighted by 1. They represent only an existence of relationship between two tasks (exists or not) without information about relationship characteristics.
- The generalized precedence constraints represent as well relationship existence as relationship characteristics.

¹Graph where tasks are represented by nodes and dependencies between tasks are represented by edges.

Assuming a generalized precedence constraints, there can be two edge types:

1. The *forward edge*, the edge from node T_i to node T_j with positive time lag w_{ij} . It indicates that T_j must start at least w_{ij} time units after S_i (start time of task T_i).
2. *Backward edge*, the edge from node T_j to node T_i with negative time lag w_{ji} . It indicates that S_j must be no more than w_{ji} time units after S_i .

The objective is to find a schedule with minimal *makespan* $(C_{max})^2$ satisfying task durations and relations, and resource constraints.

Example 1.1: Building a house

The table below shows a simple project plan for building a house (note that task durations are not real).

Task number	Task	Duration	General precedence constraint
1	Foundations	15 days	
2	Walls	10 days	Starts at least 5 days after finishing the foundations (task 1).
3	Roof	12 days	Starts after finishing the walls (task 2).
4	Doors and windows	2 days	Starts after finishing the roof (task 3).
5	Cross buntions	2 days	Starts after finishing the doors and windows (task 4).
6	Kitchen	2 days	Starts at least 3 days after starting the cross buntions (task 5).
7	Bedroom	3 days	Starts no more than 4 days after start building the kitchen (task 6).
8	Bathroom	2 days	Starts at least 2 days after starting the cross buntions (task 5).
9	Living room	3 days	Starts at least 3 days after starting the cross buntions (task 5).

The problem representation could be rewritten to task-on-node graph with generalized

² C_{max} (makespan) = $\max (T_i + p_i)$ the length of schedule.

precedence constraints.

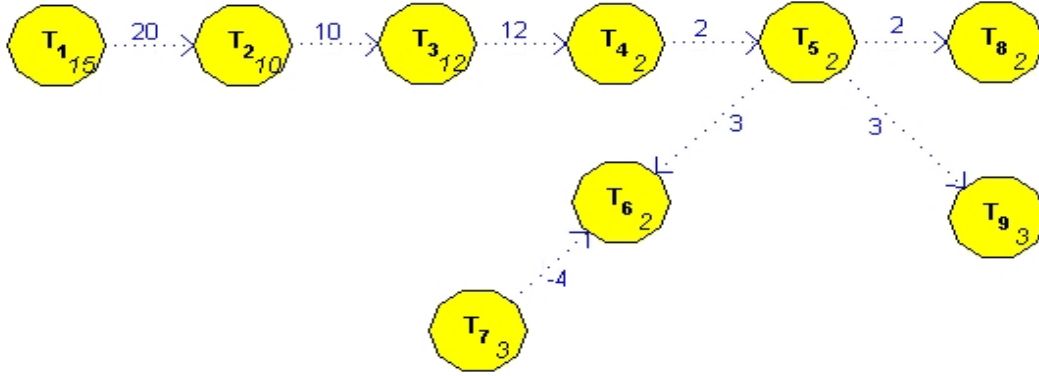


Figure 1.2: Example: Building a house, task-on-node graph

The objective is to find a schedule minimizing total project length with respect to generalized precedence constraints. The solution of our problem could be represented by the schedule shown in Figure 1.3.

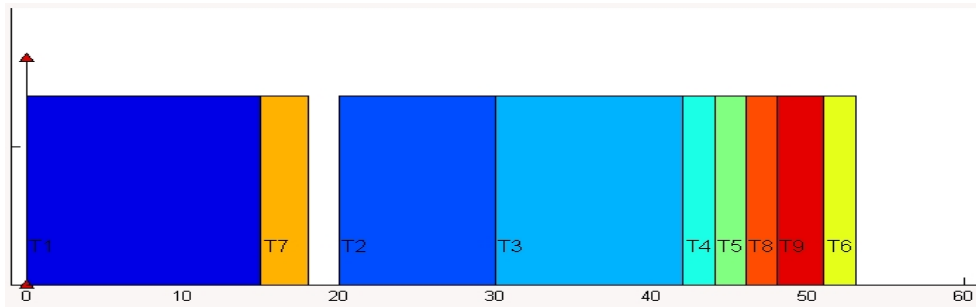


Figure 1.3: Example: Building a house, resulting schedule

The problem is referred to NP-hard class problems which means that no polynomial time algorithm³ exists for its solution. There are three approaches to solve NP-hard problems:

1. Algorithms like Branch-and-Cut or Branch-and-Bound, which find an optimal solution at the expense of worst-case computation time⁴.

³Polynomial time algorithm is an algorithm, where its computational time is in worst case upper bounded by polynomial function of the input problem size.

⁴Computation time is a time that takes algorithm to find an optimal solution.

2. Heuristic algorithms are used to find a solution among whole solution space, but they do not guarantee that the found solution is the best one. They work with respect to the computation time.
3. Approximation algorithms work in polynomial time, but there is no guarantee that the found solution is optimal. The only guarantee is that relative error between approximation algorithm solution and optimal solution is lower than a certain coefficient.

The primary goal of this work is to find and implement the most suitable algorithm to minimize the total project length, with respect to generalize precedence constraints between tasks. This problem is known as the *Resource-Constrained Project Scheduling Problem with Arbitrary Temporal Constraints* (RCPSP/max). There have been many scheduling algorithms solving RCPSP/max. We decided for An Effective Algorithm for Project Scheduling With Arbitrary Temporal Constraints which is a heuristic algorithm invented by Tristan B. Smith and John M. Pyle (2004). At the core of selected heuristic is a *Squeaky Wheel Optimization* (Joslin a Clements, 1998) method with an effective resolution mechanism called *Bulldozing*. Selected method is competitive to Branch-and-Bound (BaB) and Integer Linear Programming (ILP) algorithms. It consistently solves all feasible problems with respect to algorithm computation time, which is mostly lower than above mentioned methods.

1.3 Related work

Related work can be characterized along two dimensions: scheduling algorithms and scheduling tools.

1.3.1 Scheduling algorithms

As we mentioned before, the problem consists of a set of activities and a set of renewable resources. All algorithms assume that resource capacities, activity processing times and activity dependencies are fixed throughout project, and the objective is to minimize the total project duration (makespan). Preemption is not allowed. Previous research on algorithms solving RCPSP/max problem uses mathematical programming techniques

and implicit enumeration, e.g., linear programming and branch-and-bound. Nowadays, when project scale is growing, heuristic algorithms are more interesting.

1.3.1.1 Branch-and-Bound algorithms

There have been developed various branch-and-bound algorithms to find an optimal solution. There are different branching and pruning methods used to solve the RCPSP/max problem. The static RCPSP has been extensively studied by Brucker (1998). A major difficulty in this problem is to maintain the resource limitation over the horizon time.

A Branch-and-Bound Algorithm for a Single-machine Scheduling Problem With Positive and Negative Time-lags (Brucker et al., 1999) shows that complex scheduling problems can be reduced to the problem of scheduling tasks with arbitrary time-lags given by relations on a single-machine. The second part introduces the branch-and-bound algorithm for solving single-machine problem.

A Branch-and-Bound Procedure for the Generalized Resource-Constrained Project Scheduling Problem (Demeulemeester a Herroelen, 1997) is based on a depth-first solution strategy that solves resource capacity violations by delaying a minimal subset of activities. Nodes in the solution tree represent resource and precedence feasible partial schedules. Branches coming from a parent node correspond to minimal combinations of activities, the delay of which resolves resource conflicts at each parent node. Precedence based lower bound and two dominance rules (left-shift rule, cutset rule) are introduced in order to restrict the growth of the solutions tree.

1.3.1.2 Linear programming

Scheduling with Start Time Related Deadlines (Sucha a Hanzalek, 2004) is based on integer linear programming method. The algorithm complexity is not polynomial but method is able to find an optimal solution.

Linear programming based algorithms for preemptive and non-preemptive RCPSP (Damay et al., 2007). The algorithm uses variables associated to subsets of independent activities that can be processed simultaneously. These activities are called antichains. Precedence and non-preemption constraints are represented by antichains and eliminated by the linear formulation.

1.3.1.3 Heuristics

Local Search Algorithms for a Single-Machine Scheduling Problem with Positive and Negative Time-Lags (Hurink a Keuchel, 2001) is a local search algorithm where basic method is a tabu search approach which starts with an 'infeasible' sequence of the jobs and tries to guide the search to a feasible sequence. This method can be considered as general metaheuristic and mostly succeeds in feasible solution. One of the biggest disadvantages of this algorithm is that the computational times are very often high.

In an article, *Self-Adapting Genetic Algorithms with an Application to Project Scheduling* (Hartmann, 1999), genetic algorithm is presented using mechanisms or processes such as selection, crossover and mutations, similar to Darwinian natural selection biological model. The goal is to produce a better solution by selecting only the best existing solutions and their recombination.

Squeaky wheel optimization (Joslin a Clements, 1998) is an iterative search technique for solving optimization problems. A solution is constructed by a greedy algorithm based on the priorities assigned to all elements of the problem (elements with higher priority are handled earlier). Constructed solution is then analyzed and elements priorities are changed. The result of this analysis is a new priority order used by the greedy algorithm to construct the next solution. This Construct/Analyze/Prioritize cycle continues until some accepted or required solution is found.

A Constraint-Based Method for Project Scheduling with Time Windows (Cesta et al., 2000). At the core of the algorithm is a Constraint satisfaction problem solving (CSP) search procedure, which generates a set of task start times and removes resource conflicts from otherwise temporary feasible solution. This is done by a conflict sampling method that selects conflict sets involving tasks with higher capacity requests.

The article, *A Competitive Heuristic Solution Technique for Resource-Constrained Project Scheduling* (Tormos a Lova, 2001), presents heuristic approach, based on multi-pass method, combining random sampling procedures with backward-forward method. The algorithm component parameters are specified through a step-wise computational analysis.

1.3.2 Scheduling tools

In this work, when we talk about scheduling tools, we mostly mean project management software that uses scheduling algorithms to perform a *Resource Leveling*⁵ function.

Microsoft Project 2007 (<http://office.microsoft.com/project>) is a proprietary, widely used project management software. Optimization of the project schedule is done by manually assigning a priority number to a task. MS Project allows user to set resource leveling properties (see Figure 1.4). The most important property is 'Leveling order' that determines how will scheduling algorithm treat with tasks and in which order.

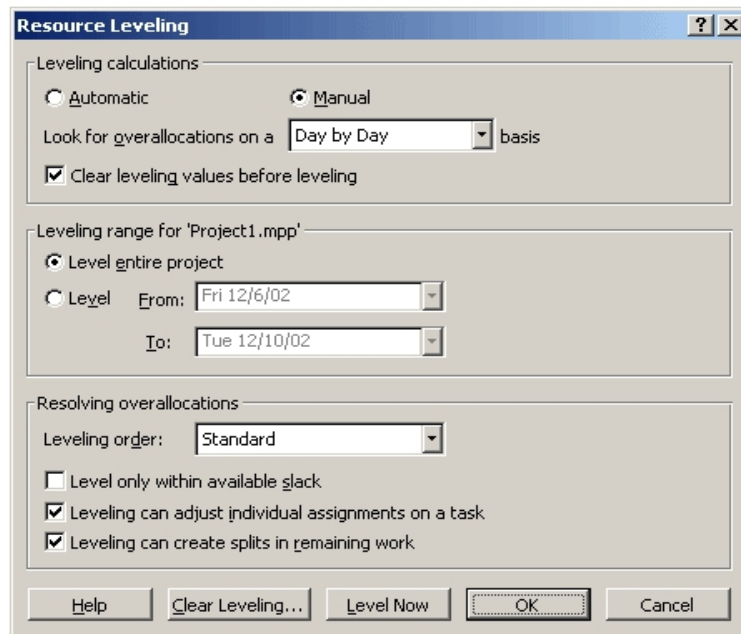


Figure 1.4: Microsoft Project 2007

There are 3 possibilities:

- ID Only - algorithm delays tasks with higher Task ID first before looking at lower IDs.
- Standard - algorithm examines following criteria in this order: predecessor relationship, slack, dates, priorities, constraints.

⁵Resource leveling is a process of resolving resources over-allocations in the project.

- Standard and Priority - same as Standard, but uses priorities as primary factor.

Even though, Microsoft Project is very popular and prestige software, it is unable to bind two tasks with forward and backward edge all at once (see Figure 1.5).

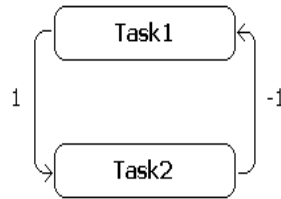


Figure 1.5: Microsoft Project, two-task problem example

Sciforma Corporation PS8 (<http://www.sciforma.com/>) is proprietary project management software. It allows to set resource leveling properties, and to specify both project and task priorities for use in resource leveling. It also allows to define margins for each resource that determine what conditions constitute an overload.

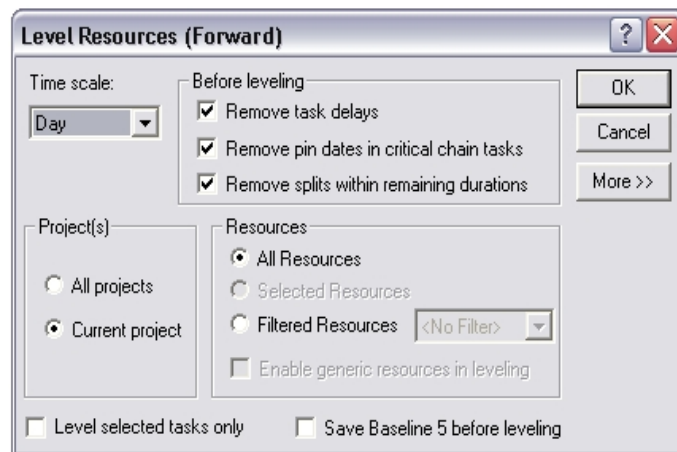


Figure 1.6: Sciforma Corporation PS8

Open Workbench (<http://www.openworkbench.org/>) is an open-source project management software. The main difference between Microsoft Project and Open Workbench is that Microsoft Project leveling resource algorithm is only shifting overloaded tasks on the schedule to find the next contiguous window in which the resource can complete the entire task. In Open Workbench, the resource leveling algorithm calculates the whole schedule from the beginning. Although, it is an open-source software, scheduling algorithms are currently not open sourced.

1.4 Contribution

BaB or ILP algorithms find always an optimal solution of RCPSP/max. However, they are not able to find an optimal solution for a large scale scheduling problems. Therefore heuristic algorithms are used. Their computational times are very often less than BaB or ILP algorithms but on the other side, they often leads to loss of finding the best solution. Heuristic mostly find a near optimal solution in polynomial time.

The thesis primary object is to implement An Effective Algorithm for Project Scheduling with Arbitrary Temporal Constraints which is heuristic based on Squeaky Wheel Optimization and resolution method called Bulldozing. Selected heuristic will be implemented into Mathworks Matlab environment and into an open-source project, GanttProject. GanttProject is the project manager software working under GPL⁶. It will be extended for a new function called Resource Leveling which will improve application usability closer to commercial software like MS Project. All source-codes will be uploaded to GanttProject CVS⁷ server, hosted at SourceForge(<http://sourceforge.net/>), and will be a part of the future version.

Outline of this thesis. The thesis is divided into six chapters. To give a concise overview, let us summarize the contents of the individual chapters:

CHAPTER 1: Provides a brief introduction to project scheduling, problem statement and related works.

CHAPTER 2: This chapter gives a detailed introduction to RCPSP/max. It contains problem description and also introduces basic scheduling theory notions like task, processing time, release date, C_{max} , etc.

CHAPTER 3: Describes closely Squeaky Wheel Optimization approach.

CHAPTER 4: Here, individual parts of selected algorithm are analyzed. It is divided into two sections. The first one introduces main algorithm parts: Squeaky Wheel optimization as a part of an implemented algorithm, Bulldozing and Refilling methods. The second one concerns about own algorithm modifications that lead to enhancement its

⁶General Public License

⁷Concurrent Versions System

performance.

CHAPTER 5: This chapter gives an information about GanttProject structure and about own implementation of the scheduling algorithm as a Resource Leveling method.

CHAPTER 6: Chapter six presents an experimental results of the implemented algorithm and its modification.

Chapter 2

Resource-Constrained Project Scheduling Problem with Arbitrary Temporal Constraints (RCPSP/max)

2.1 Introduction to scheduling

Before defining RCPSP/max more closely, let's introduce some basic scheduling theory terms that are used throughout the thesis.

One of the most important term is *Task* (or sometimes called *Activity*). Graphic representation of task and its parameters is shown in Figure 2.1.

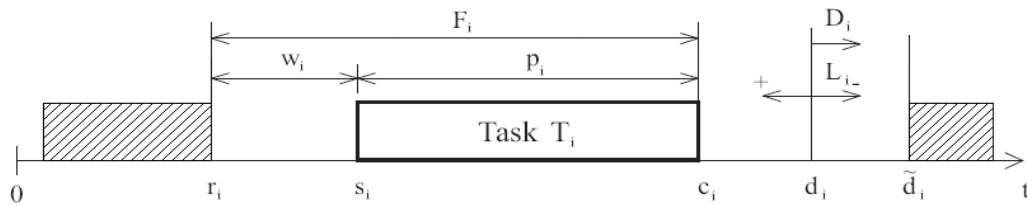


Figure 2.1: Graphic representation of task parameters

Tasks are characterized by following notions:

- Task *processing times* $\mathbf{p} = (p_1, p_2, \dots, p_i, \dots, p_n)$. Concern about how long are individual tasks processed without interruption.

- *Ready times* $\mathbf{r} = (r_1, r_2, \dots, r_i, \dots, r_n)$. Express time where each task is ready to be processed.
- *Due dates* $\mathbf{d} = (d_1, d_2, \dots, d_i, \dots, d_n)$. Due dates specify times when each task **should** be finished. If task is finished later than its due date, some penalization is imposed (e.g. increase production costs).
- *Deadlines* $\tilde{\mathbf{d}} = (\tilde{d}_1, \tilde{d}_2, \dots, \tilde{d}_i, \dots, \tilde{d}_n)$. In comparison to due dates, deadlines specify times when each tasks **must** be finished otherwise the scheduling is assumed to be failed.
- *Priorities* $\mathbf{w} = (w_1, w_2, \dots, w_i, \dots, w_n)$. Concern about task priority with respect to other tasks (e.g. one task must be scheduled before another).

For each task, there are additional parameters, used like a criterion for resulting schedule classification:

- *Completion time* C_i . Time when task is truly finished.
- *Flow time* $F_i = C_i - r_i$. Represents total task processing time, including task's waiting.
- *Tardiness* $D_i = \max(L_i, 0)$. Non-zero value indicates overrun required due date.
- *Earliness* $E_i = \max(-L_i, 0)$. Non-zero value indicates that task has been finished earlier than required due date.

We have introduced terms like task and its parameters. Now let's introduce some criteria along which resulting schedules are optimized:

- *Makespan* (total schedule length):

$$C_{max} = \max\{C_i\}$$

- *Maximum lateness*:

$$L_{max} = \max\{L_i\}$$

- *Mean flow time:*

$$\bar{F} = \frac{1}{n} \sum_{i=1}^n F_i$$

- *Mean weighted flow time:*

$$\bar{F} = \frac{\sum_{i=1}^n F_i}{\sum_{i=1}^n w_i}$$

- *Mean earliness:*

$$\bar{E} = \frac{1}{n} \sum_{i=1}^n E_i$$

2.2 RCPSP/max problem

RCPSP/max problem contains a set $\{T_1, \dots, T_n\}$ of tasks and a set $\{R_1, \dots, R_m\}$ of resources. Each task T_i has its start time S_i and a processing time p_i . Each resource has its maximum capacity c_k and execution of each task T_i requires an amount r_{ik} of resource k for each time unit of execution. There are two types of constraints:

Precedence constraints mean that the start time of each task is related to the start time of another task. There are two types of generalized precedence constraints:

1. Positive time lags w_{ij} , from task T_i to task T_j , indicates that S_i must be at least w_{ij} time units after S_j .
2. Negative time lags w_{ji} , from task T_j to task T_i , indicates that S_j must be no more than w_{ji} time units after S_i .

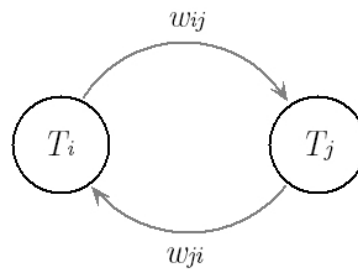


Figure 2.2: Two tasks with generalized precedence constraints

Precedence constraints can be summarized as follows:

$$\min_{i,j} w \leq S_j - S_i \leq \max_{i,j} w$$

Resource constraints represent maximum capacity of each resource.

Resource constraints can be summarized as follows:

$$\sum_{\{i|S_i \leq t \leq S_i + p_i\}} r_{ik} \leq c_k, \forall t, \forall k$$

A schedule can be considered as an assignment of start time S_i to each task T_i . It is *time-feasible* if it satisfies all precedence constraints and is *resource-feasible* if it satisfies all resource constraints. A schedule satisfying both time-feasible and resource-feasible constraints is *feasible*. The goal of an RCPSP/max problem is to find a feasible schedule minimizing $C_{max} = \max(S_i + p_i)$. Minimizing the C_{max} is not the only one criterion while solving the RCPSP/max problem. C_{max} is always bound with tasks and their start times. Sometimes there is a need for criteria, where the objective function does not depend on how the individual activities are scheduled, but on how they are combined in order to use the resources. For this purpose, the *Resource Leveling Problem* (RLP) criterion is used.

RLP minimizes:

- The total squared utilization cost of all resources.
- The total overload cost (loading the resource R_k over its capacity c_k incurs the total overload cost).

RCPSP/max is referred to the NP-hard class problems. It means that no polynomial time algorithm exists for this problem.

Chapter 3

Squeaky Wheel Optimization

3.1 Squeaky Wheel Optimization

Squeaky Wheel Optimization (SWO) is an iterative search technique for solving optimization problems. A solution is constructed by a greedy algorithm with priority scheme. When constructing a solution, greedy algorithm is making its decision based on priorities assigned to all elements of the problem¹ (elements with higher priority are handled earlier). Constructed solution is then analyzed and elements priorities are changed. The result of this analysis is a new priority order, used by the greedy algorithm to construct the next solution. This Construct/Analyze/Prioritize cycle (see Figure 3.1) continues until some accepted or required solution is found.

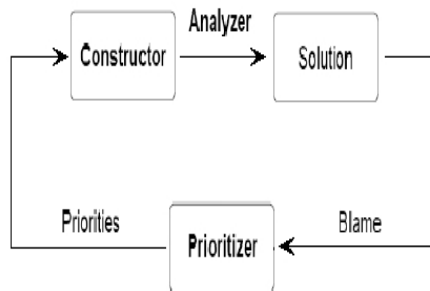


Figure 3.1: Construct/Analyze/Prioritize cycle

¹In scheduling domain, those elements are referred to tasks.

Squeaky wheel optimization can be split into three parts:

Constructor

The constructor adds elements, one at a time, into schedule based on priority order. Priorities are arranged in increasing order which means that elements with higher priority are handled by constructor earlier.

Analyzer

The analyzer analyzes the solution created by the constructor and tries to find trouble elements (elements that are hard to schedule or don't respect task constraints). After finding trouble elements, the "blames" are assigned. The analyzer calculates a lower bound on the minimum possible cost (minimum worthiness that task extends actual makespan) that each task could contribute to schedule. The blame assigned to each task is the difference between its actual cost and a minimum possible cost. In other words, priorities of trouble elements are increased much more than non-trouble elements. In the next algorithm iteration, trouble elements (or hardly scheduled tasks) are handled by constructor earlier.

Prioritizer

Once the blame has been assigned, the prioritizer modifies the previous sequence of elements by moving tasks with higher blame into the front of sequence. The higher the blame, the further the element is moved.

Example 3.1: Squeaky wheel optimization

Let's have a look at simple example for better understanding Construct/Analyze/Prioritize cycle. The problem is represented by task-on-node graph shown in Figure 3.2.

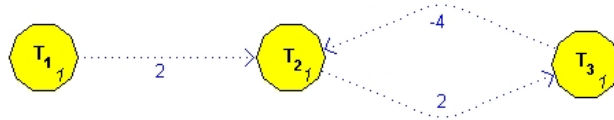


Figure 3.2: Example task-on-node graph

task processing times: $p_1 = p_2 = p_3 = 1$

initial task priorities: $\text{Priorities}(T_1) = 1$, $\text{Priorities}(T_2) = 2$, $\text{Priorities}(T_3) = 3$

precedence constraints:

T_1 must start at least 2 time units before T_2

T_2 must start at least 2 time units before T_3

The first SWO iteration:

Construction

Construction task order (based on task priorities) is (T_3, T_2, T_1) .

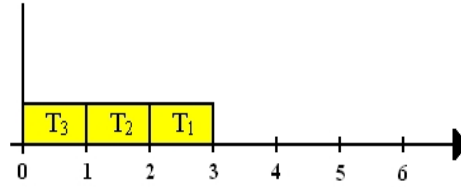


Figure 3.3: The first SWO iteration schedule

Analyze:

Schedule is not feasible.

Based on precedence constraints, T_1 must be placed at least 2 time units before T_2 ($S_1' = -1$) and T_2 must be placed at least 2 time units before T_3 ($S_2' = -2$)

$$\text{Blames}(T_1) = S_1 - S_1' = 2 - (-1) = 3$$

$$\text{Blames}(T_2) = S_2 - S_2' = 1 - (-2) = 3$$

$$\text{Blames}(T_3) = S_3 - S_3' = 0$$

$$\text{newPriorities}(T_1) = \text{Priorities}(T_1) + \text{Blames}(T_1) = 1 + 3 = 4$$

$$\text{newPriorities}(T_2) = \text{Priorities}(T_2) + \text{Blames}(T_2) = 3 + 2 = 5$$

$$\text{newPriorities}(T_3) = \text{Priorities}(T_3) + \text{Blames}(T_3) = 3 + 0 = 3$$

Prioritize

New construction order = (T_2, T_1, T_3) .

The second SWO iteration:

Construction:

Construction task order is (T_2, T_1, T_3)

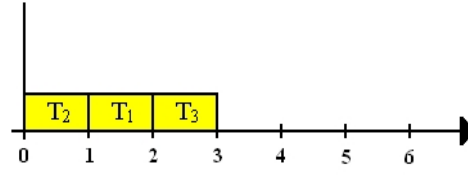


Figure 3.4: The second SWO iteration schedule

Analyze:

Schedule is not feasible.

Based on precedence constraints, T_1 must be placed at least 2 time units before T_2 ($S_1' = -2$).

$$\text{Blames}(T_1) = S_1 - S_1' = 1 - (-2) = 3$$

$$\text{newPriorities}(T_1) = \text{Priorities}(T_1) + \text{Blames}(T_1) = 4 + 3 = 7$$

$$\text{newPriorities}(T_2) = \text{Priorities}(T_2) = 5$$

$$\text{newPriorities}(T_3) = \text{Priorities}(T_3) = 3$$

Prioritize:

New construction order = (T_1, T_2, T_3)

The third SWO iteration:

Construction:

Task order is (T_1, T_2, T_3)

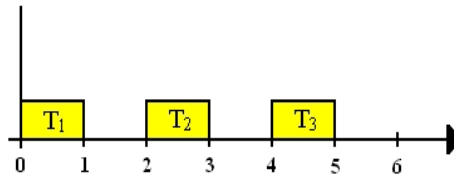


Figure 3.5: The third SWO iteration schedule

Analyze:

The schedule is feasible.

Chapter 4

An Effective Algorithm For Project Scheduling With Arbitrary Temporal Constraints

In the Chapter 2, RCPSP/max problem has been introduced. Now, let's have a look how to solve it. There are many approaches to solve RCPSP/max. We have decided for An Effective Algorithm For Project Scheduling With Arbitrary Temporal Constraints (Smith and Pyle, 2004). This algorithm has been chosen because of its low computation time and good solution quality. The chapter is divided into two sections. The first one deals with the algorithm fundamentals. The second one describes the algorithm implementation into Mathworks Matlab environment. The next chapter gives an introduction to project management and concerns about the implementation of the algorithm into an open-source project management software, GanttProject.

The algorithm represents a heuristic approach using an iterative method called Squeaky wheel optimization (SWO) to aim RCPSP/max problem. At the core of SWO is a greedy schedule constructor. The constructor is priority-based, which means that the tasks are placed one at time into the schedule according to their priorities (higher the priority, earlier the task is handled by constructor). Each solution is then analyzed and the priorities are changed in order to find a feasible solution. This cycle continues until some acceptable solution is found.

Since RCPSP/max belongs to NP-hard problems, it can happen that SWO will be ineffective and not able to find a feasible solution. Therefore, the algorithm is extended for a new conflict resolution mechanism called *Bulldozing*. This method is trying to main-

tain a partial-constructed schedule feasibility by moving the set of tasks to the later start times. After finding a feasible solution, similar methods are used to move certain sets of tasks back to earlier start times, in order to preserve schedule feasibility.

Before describing algorithm implementation, let's briefly remind some RCPSP/max problem fundamentals introduced in Chapter 2:

The problem contains a set $\{T_1, \dots, T_n\}$ of tasks and a set $\{R_1, \dots, R_m\}$ of resources. Each activity T_i has its start time S_i and a processing time p_i . Each resource has a maximum capacity c_k and execution of each task T_i requires an amount r_{ik} of resource k for each time unit of execution.

Precedence constrains:

$$S_i + w_{ij} \leq S_j$$

Resource constraints:

$$\sum_{\{i | S_i \leq t \leq S_i + p_i\}} r_{ik} \leq c_k, \forall t, \forall k$$

The main goal is to minimize $C_{max} = \max(S_i + p_i)$ representing the total schedule length. The resulting schedule has to be feasible which means that it has to satisfy all precedence and resource constraints.

4.1 Algorithm Implementation

At first, let's give an general overview and then explain certain algorithm features more closely.

As it was mentioned before, SWO is an iterative search technique for solving optimization problems. It means that solution is found after minimal Construct/Analyze/Prioritize cycles (iterations). In each iteration, the constructor (represented by the *ScheduleWith-Dozing*(T_i, X^1) method) selects a start time for each unscheduled task from a time window $[EST_i, LST_i]$ (EST_i - Earliest Task Start Time, LST_i - Latest Task Start Time). Each window represents all time points between the earliest and latest start time we

¹where X is a set of tasks that need to be moved by the Bulldozing method

wish to consider, and is initialized to $[0, horizon - p_i]$, where initial *horizon* is the goal makespan. When $[EST_i, LST_i]$ interval is selected then $PlaceTask(T_i, t)$ places task T_i into the *Schedule* at a time point t . If t is later than LST_i , *Bulldozing method* is invoked. If there has been still no feasible start time selected, then T_i is placed at EST_i and is marked as unfeasible. After exploring all tasks, resulting schedule is analyzed, blames are assigned and priorities are updated. When a feasible solution is found, *LeftShifting method* is trying to move all tasks to earlier start times in order to preserve feasibility. After finding a feasible solution, makespan is counted and compared to the previous solution. If the new solution is better than the previous one, the new solution is saved and kept by the algorithm.

An algorithm general overview can be summarized in the following pseudo code:

```

1:    $MK_{best} = \text{infinity}$ 
2:    $horizon = \text{goal makespan}$ 
3:   initialize all Priorities
4:   for 1 to  $MaxIter$ 
5:       for 1 to  $n$ 
6:            $T_i$  - get unscheduled task with the highest priority
7:            $X = \{ \}$ 
8:           if ScheduleWithDozing( $T_i, X$ ) fails
9:                $feasible = \text{FALSE}$ 
10:        end
11:        if feasible
12:            LeftShifting(Schedule)
13:            if ( $MK_{current}$  less than  $MK_{best}$ )
14:                 $MK_{best} = MK_{current}$ 
15:                decrease  $horizon$ 
16:        if 10 iteration without feasible solution
17:            increase  $horizon$ 
18:        update Priorities
19:    end

```

At the beginning of the algorithm, priorities need to be initialized. However, the al-

gorithm is able to find a feasible solution without priorities initialization, the proper initialization can help to find a feasible solution faster. Each $Priority(T_i)$ is initialized to $10 \times LST_i$ value, calculated by the temporal constraint propagation (Caseau, 1997). Before LST_i calculation, the problem is transformed into the task-on-node graph with general precedence constraints. This graph is then used as an input argument for the *Floyd-Warshall algorithm* (Demel, 2002) which returns the longest path between each two nodes (tasks) in a weighted, directed graph (task-on-node graph).

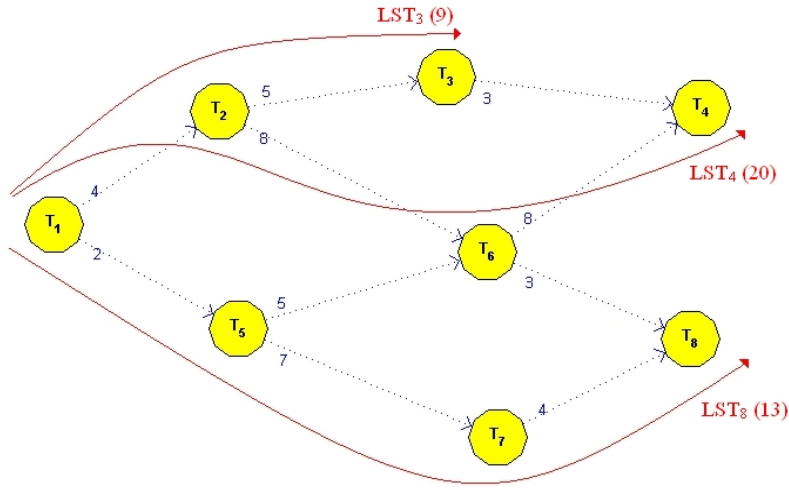


Figure 4.1: Initial priorities calculation

Now, let's describe individual methods more closely:

The ScheduleWithDozing method

Before describing the *ScheduleWithDozing method*, some basic scheduling terms will be remembered, and a Precedence matrix (with general precedence constraints) will be introduced:

- Task *processing times* $\mathbf{p} = (p_1, p_2, \dots, p_i, \dots, p_n)$. Concern about how long are individual tasks processed without interruption.
- Task *start times* $\mathbf{S} = (S_1, S_2, \dots, S_i, \dots, S_n)$. Specify the task processing start time.

- *Precedence matrix* with general precedence constraints W

$$\mathbf{W} = \begin{bmatrix} w_{11} & w_{12} & \dots & w_{1j} & \dots & w_{1n} \\ w_{12} & w_{22} & \dots & w_{2j} & \dots & w_{2n} \\ \vdots & \vdots & & \vdots & & \vdots \\ w_{i1} & w_{i2} & \dots & w_{ij} & \dots & w_{in} \\ \vdots & \vdots & & \vdots & & \vdots \\ w_{n1} & w_{n2} & \dots & w_{nj} & \dots & w_{nn} \end{bmatrix}$$

The matrix represents how the each task *start time* p_i is related to the *start time* of another tasks p_j (see Figure 4.2).

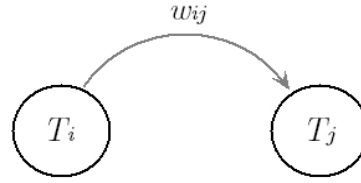


Figure 4.2: Two tasks with generalized precedence constraints

At the beginning, the function calculates a time window domain $[EST_i, LST_i]$ for the specified task T_i . This bounds (EST_i, LST_i) are calculated using the task start times vector \mathbf{p} and the precedence matrix \mathbf{W} . When the time window domain $[EST_i, LST_i]$ is determined, the algorithm is exploring all time slots in the schedule in an effort to find a free resource-feasible time slot for starting the task T_i . If none is found, *Bulldozing method* is invoked. Bulldozing allows the greedy constructor to move the sets of tasks X later in a partial schedule to maintain feasibility. After a successful bulldoze, the *LeftBulldozing method* is trying to move the same sets of tasks back to the earlier start times. If bulldozing fails and none resource feasible time slot has been found, resource constraints are neglected, and the task T_i is placed at the earliest start time EST_i . The task T_i is then marked as unfeasible.

The *ScheduleWithDozing*(T_i, X) method pseudo code is shown below:

```

1:       $t =$  earliest resource-feasible time for  $T_i$  in  $[EST_i, LST_{i(orig)}]$ 
2:      if ( $t \leq LST_i$ )
3:          PlaceTask( $T_i, t$ )
4:      else if ( $t$  more than  $LST_i$ ) % Bulldozing method is invoked
5:          PlaceTask( $T_i, t$ )
6:           $T_j =$  tasks that need to be moved to preserve schedule feasibility
7:           $X = X \cup \{T_j\}$ 
8:           $Y = X$ 
9:          while ( $Y$  is NOT empty) do
10:              $T_k =$  randomly selected task from the set  $X$ 
11:             UnplaceTask( $T_k$ )
12:              $X = X - \{T_k\}$ 
13:             ScheduleWithDozing( $T_k, X$ )
14:          end
15:          LeftBulldozing( $Y$ )
16:      else
17:          PlaceTask( $T_i, EST_i$ )
18:          Undo All Bulldozing
19:          feasible = FALSE
20:      end

```

EST_i calculation

For task T_i , all predecessors T_j are checked whether they are scheduled or not.

If yes, then:

$$EST_i = \max (EST_i, S(T_j) + w(T_j, T_i)).$$

The predecessors (in the matrix W) are examined in the order shown in Figure 4.3.

$$W = \begin{bmatrix} w_{11} & w_{12} & \dots & w_{1j} & \dots & w_{1n} \\ w_{12} & w_{22} & \dots & w_{2j} & \dots & w_{2n} \\ \vdots & \vdots & & \vdots & & \vdots \\ w_{i1} & w_{i2} & \dots & w_{ij} & \dots & w_{in} \\ \vdots & \vdots & & \vdots & & \vdots \\ w_{n1} & w_{n2} & \dots & w_{nj} & \dots & w_{nn} \end{bmatrix}$$

W [T_j, T_i]




Figure 4.3: EST_i calculation, predecessors order **LST_i calculation**

For task T_i , all successors T_j are checked whether they are scheduled or not.

If yes, then:

$$LST_i = \min (LST_i, S(T_j) - w(T_i, T_j)).$$

The predecessors (in the matrix W) are examined in the order shown in Figure 4.4.

$$W = \begin{bmatrix} w_{11} & w_{12} & \dots & w_{1j} & \dots & w_{1n} \\ w_{12} & w_{22} & \dots & w_{2j} & \dots & w_{2n} \\ \vdots & \vdots & & \vdots & & \vdots \\ w_{i1} & w_{i2} & \dots & w_{ij} & \dots & w_{in} \\ \vdots & \vdots & & \vdots & & \vdots \\ w_{n1} & w_{n2} & \dots & w_{nj} & \dots & w_{nn} \end{bmatrix}$$

W [T_i, T_j]




Figure 4.4: LST_i calculation, successors order**The Bulldozing method**

For highly constrained problems the SWO greedy algorithm is unable to find a feasible schedule. This problem is mostly occurred because of existence of the maximum time lags (the negative time lags between two nodes). Without maximum time lags, a greedy algorithm can find always a feasible schedule, because each task T_i can be postponed as

long as it is necessary to conserve resource availability. Using the maximum time lags, the task T_i can not be postponed long enough, if the tasks that constraints T_i have been scheduled too early. The main idea of bulldozing is to delay the set of tasks X constraining T_i , so that T_i can be scheduled in the postponed time.

The *Bulldozing method* is invoked when the task T_i can not be scheduled in the current $[EST_i, LST_i]$ time window domain. Bulldozing considers a larger time window domain to find a free time slot in the schedule for placing the task T_i . It extends the $[EST_i, LST_i]$ time window domain to $[EST_i, LST_{i(orig)}]$, where $LST_{i(orig)}$ is the latest start time of the task T_i when no other activities are scheduled. If bulldozing finds a free time slot (at the time t) for task T_i outside the $[EST_i, LST_i]$ window but within the $[EST_i, LST_{i(orig)}]$, the task T_i is placed at the time t . After placing the task T_i , algorithm finds all tasks T_j that constrain task T_i and places them into the set X (which is the set of all tasks that need to be moved). Each task T_k that is randomly selected from the set X , is taken out from this set, unplaced and the algorithm is trying to find a new feasible start time for the task T_k using the recursive call of *ScheduleWithDozing*(T_k, X). If all tasks from set X are scheduled feasible then bulldozing success. Otherwise, all delayed tasks T_k from the set X revert to their previous position (position before invoking bulldozing).

One of the main advantages of the *Bulldozing method* is its recursive nature, which means that more tasks can be moved than the original set X forced by task T_i . Each task T_k from the set X can also force another set of tasks X' . From a broader point of view, we can say that each bulldozed task T_i forces sub-sets of highly constrained tasks. These sub-sets will be moved to later start times until all sub-sets precedence constraints are satisfied. This approach is very interesting because it is able to divide problem into the sub-problems. Each sub-problem is then solved separately and these partial solutions are then combined into the overall solution.

Example 4.1: The Bulldozing method

The purpose of this example is to better explain the Bulldozing method. Let the example consist of four tasks with the following precedence constraints:

- Task T_4 can start at least 5 time units after starting the task T_1 .
- Task T_4 must start no more then 3 time units after starting task T_2 .
- Task T_4 must start no more then 2 time units after starting task T_3 .

- Between all other tasks there are no precedence constraints.

The problem could be also presented by task-on-node graph shown in Figure 4.5:

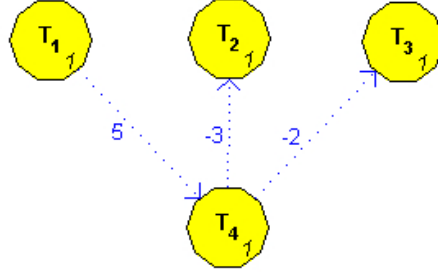


Figure 4.5: Bulldozing method example, task-on-node graph

Consider that tasks T_1, T_2 and T_3 have been scheduled within the following time domain windows:

$$EST_{1(orig)} = EST_1 = 0, LST_1 = 10$$

$$EST_{2(orig)} = EST_2 = 1, LST_2 = 10$$

$$EST_{3(orig)} = EST_3 = 2, LST_3 = 10$$

Now, we are trying to schedule T_4 inside a time window domain $EST_4 = 5, LST_4 = 4$. The constructor is unable to find a resource feasible time slot in the schedule. Therefore the LST_4 is updated to $LST_{4(orig)} = 10$ and the constructor is trying to examine the feasible schedule once again. Now, the constructor has found a free time slot at time $t = 5$. The task is scheduled at $t = 5$ apart from maximum time lags between tasks T_2 and T_3 (see Figure 4.6).

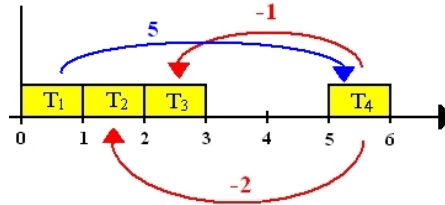


Figure 4.6: Bulldozing method example

Bulldozing method finds all tasks T_j causing unsatisfied (maximum time lags) constraints and updates the set X .

Constraining tasks are: T_2, T_3

Update the set X: $X = \{T_2, T_3\}$

After unscheduling the tasks (one at time) $\{T_2, T_3\}$, the time domains for each task T_2 and T_3 is updated:

$EST_2=3, LST_2=10$

$EST_3=4, LST_3=10$

After the successful bulldoze (moving all tasks from the set X to the later start times), the resulting schedule in Figure 4.7 is feasible and all general precedence constraints are satisfied.

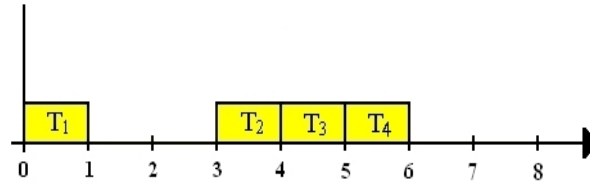


Figure 4.7: Bulldozing method example

The LeftBulldozing method

However, bulldozing finds almost always a feasible solution (if exists), this solution is often not optimal. The *LeftBulldozing method* is trying to reduce the schedule C_{max} and helps to find an optimal solution. After a successful bulldoze, the same set of tasks X , that have been moved by bulldozing, is tried to be moved by leftBulldozing back to the earlier start times. This approach is also Bulldozing, because we are trying to move the sub-sets of tasks that are highly constrained among themselves.

The leftBulldozing is the same as scheduleWithDozing except for these three differences:

- On line 1, the range tried is $[EST_{i(orig)}, LST_i]$ (ie we try placing the task earlier than it can currently go, not later). We assume that $EST_{i(orig)}$ is a counted EST_i before running bulldozing.
- On line 2, we instead use $t \geq EST_i$.
- On line 17, we place the activity where it was already, rather than at EST_i .

The *LeftBulldozing*(X) method pseudo code is shown below:

```

1:       $T_i$  = the first task from the set  $X$ 
2:       $X = X - \{T_i\}$ 
3:       $t$  = earliest resource-feasible time for  $T_i$  in  $[EST_{i(orig)}, LST_i]$ 
4:      if ( $t \leq EST_i$ )
5:          PlaceTask( $T_i, t$ )
6:           $T_j$  = tasks that need to be moved to earlier start time
7:           $Y = Y \cup \{T_j\}$ 
8:          while ( $X$  is NOT empty) do
9:               $T_k$  = randomly selected task from the set  $Y$ 
10:             UnplaceTask( $T_k$ )
11:              $Y = Y - \{T_k\}$ 
12:             LeftBulldozing( $T_k, Y$ )
13:         end
14:     if ( $t > LST_i$ )
15:         Undo All LeftBulldozing
16:         feasible = FALSE
17:     end

```

Example 4.2: The LeftBulldozing method

Let's have a look at the previous example and try to perform the LeftBulldozing method to the resulting schedule. Even though, the purpose of the LeftBulldozing method is to reduce the schedule makespan, this approach rarely works. Because of this, the example mainly shows how to count the $EST_{i(orig)}$ for tasks that are left bulldozed.

After a successful bulldoze of the tasks T_2 and T_3 , leftBulldozing is trying to move the same set of tasks back to the earlier start times. At first, the EST_i time domains are updated to:

$$EST_2 = EST_{2(orig)} = 1, LST_2 = 10$$

$$EST_3 = EST_{3(orig)} = 2, LST_3 = 10$$

Applying the LeftBulldozing method, the resulting schedule (in Figure 4.8) hasn't change compared to the resulting schedule produced by the Bulldozing method.

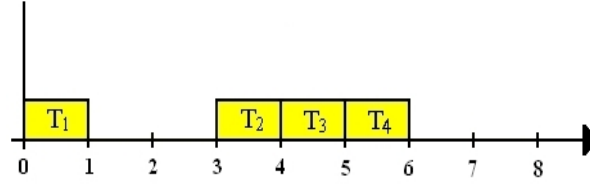


Figure 4.8: LeftBulldozing method example

The LeftShifting method

After finding a feasible solution (schedule), this method is trying to reduce the schedule makespan. The main idea of this method is to left-shift tasks that can take an advantage of resources vacated by bulldozed activities. Simply said, this approach is trying to move each task A_i , for which holds start time $S_i > EST_i$, to the left as much as possible.

The algorithm explores start times S_i of all tasks (in order they were scheduled) and counts appropriate EST_i . If $S_i > EST_i$ then the task is moved to the earlier start time as much as possible.

The *LeftShifting(Schedule)* method is represented by the following pseudo code:

```

1:    for 1 to  $n$ 
1:         $t =$  earliest resource-feasible time for  $T_i$  in  $[EST_i, LST_i]$ 
2:        if ( $t < S_i$ )
3:            PlaceTask( $T_i, t$ )
4:    end

```


The Priorities update

The priorities are updated at the end of each algorithm iteration. This is a very important phase, because it strongly affects the construction task order in the next algorithm iteration. The main idea of priorities updating is to increase priorities to all not-scheduled tasks by a constant amount. All other tasks priorities are increased by a smaller amount with a small probability. Doing these two steps, we ensure that in each iteration, at least one task will have its priority changed.

In our implementation, following priorities-increase constants are chosen:

- An amount, how much more the task priority is increased, when the task has been scheduled = 2
- An amount, how much more the task priority is increased, when the task could not be scheduled = 4
- A probability, with which the task priority is increased, when the task has been scheduled = 0.01

4.2 Own Algorithm Modifications

Even though, the implemented algorithm shows a very good benchmark results (shown in the Chapter 6), there are some modifications that can be done to increase its performance. The main impact of these changes is to decrease algorithm computational time on the assumption, that the resulting makespans of the original and modified algorithm are nearby equal. This can be done in two ways:

- By modifying the prioritization phase or more precisely, to change the idea of updating the priorities.
- By exchanging the certain algorithms for the simpler ones with nearby the same result.

Now, let's have a look at it more closely:

4.2.1 Prioritization phase modifications

The main purpose of the algorithms solving the RCPSP/max is to find an appropriate construction order in which the tasks are scheduled feasibly. The schedule, constructed in this order, should also minimize the C_{max} criterion. Therefore, the prioritization phase is very important because it strongly influences the next iteration constructor task order. Compared to the original implementation, the main change is that priorities are not updated in the end of each iteration, but they are update continuously in the bulldozing part of the *ScheduleWithDozing method*. The main idea is that task T_i , that needs to be bulldozed, forces another tasks T_k (all tasks in the set X) to be moved to the later start times. All tasks T_k from X are then feasibly rescheduled; videlicet these tasks, for conserving the schedule feasibility, must be in the construction order later than T_i . Therefore, the best what way to update the priorities is to make sure, that in the construction order, all tasks $T_k \in X$ are later than T_i . Notice, that we do not do the difference between the tasks in the set X . We can afford it, because the tasks T_k that are moved by bulldozing are selected randomly (from the priorities point of view, these tasks are equal). This is done by assigning the priority values to all tasks T_k less than the priority of task T_i . We decided to decrease this value by a constant amount weighted by 2:

$$\text{Priorities}(T_k) = \text{Priorities}(T_i) - 2$$

An important observation is that the correct construction order (obtained from the appropriate priority values), which achieves a feasible schedule, is found only in the one algorithm iteration.

This is not an only one difference. Another modification, but not as important as previous one, is that at the beginning of each iteration, a random constant amount is added to each priority. This is done because it can happen, that two tasks will have the same priority and it needs to be decided which one to schedule first.

$$\text{Priorities}(T_i) = \text{Priorities}(T_i) + \text{random}(0..1)/1000$$

4.2.2 Reducing the code

The main idea is to exchange certain algorithms, that take a long time to be processed or their effect to the resulting schedule is not equal to their processing time. These algorithms are substituted for the simpler ones that provide nearby the same result, but using the smaller amount of CPU time. We decided for letting out the *LeftBulldozing method*. There are two reasons to do it:

1. It is a recursive method that is invoked every time when bulldozing success.
2. If the correct construction task order is known, we are able to schedule all tasks once again from the beginning. Using this construction order, we will always find a feasible solution where majority part of tasks is placed to their earliest start times, which achieve the minimal makespan. This can be done without using the *LeftBulldozing method*.

The leftBulldozing is then replaced by a new algorithm that originates by modifying the *LeftShifting method*. The new method, in comparison to the original one, finds not only tasks that can be moved to the earlier start times, but constructs the whole schedule from the beginning using the feasible construction order. This all is done not after a successful buldoze but at the end of each algorithm iteration.

The comparisons of the original and modified algorithm are presented in the Chapter 6.

Chapter 5

Project Management Software, GanttProject

5.1 Project Management

The project management is the discipline of organizing and managing the resources, with a view to achieving a specific goal (e.g. minimizing the total project cost or total project length). A project might be a building construction, implementation of new technologies into a factory or a political campaign. The project management in the modern sense began in the early 1960s, but it was also known by ancient Egyptians engineers while they were building great pyramids, that are seen still today. They had to plan how many people would be involved in specified works, how much material they would need, how much it would cost, how long it would take, etc. Nowadays, it is very similar, but the use of the project management is wider. People involved in business realized, that the properly planed projects lead to saving costs, and to the better prevention before problems that could happened during the project realization. The project requirements are often summarized in a triangle (see Figure 5.1), where each vertex represents one of the following constraints: time, cost and scope.

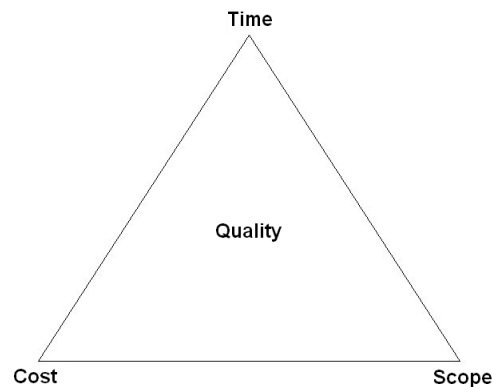


Figure 5.1: The Project Management Triangle

Each of these constraints has its own meaning:

- *Time*, represents the total project length.
- *Cost*, is equal to the total project cost.
- *Scope*, refers to what must be done to produce the project's end result.

The main idea is that the one side of the triangle cannot be changed without impacting the others. E.g. increasing the scope typically leads to increase of time and increase of the cost. The deal of the project management is to organize the project team in the way that will meet these constraints.

We have introduced the basics of the project management. Now becomes a question, how to plan the project in the easiest and most efficient way? The answer is, use the project management software.

5.2 Project Management Software

The project management software is an important tool for each project manager. It allows user to easily create, modify, share projects between managers, show Gantt charts¹, etc. Each project consists of the set of resources, of the set of tasks and of the set precedence

¹Gantt chart is a type of bar chart illustrating a project schedule (start and finish dates of the tasks, project length, etc.). For more, see information (<http://www.ganttchart.com/>)

constraints between tasks (notice, that this is the same as RCPSP/max problem). During the project planning, these tasks are created and assigned to the appropriate resources. One of the purposes of the project management software is to continuously inform the manager about the project state (e.g. gives information about total project length, task start times, task end times, critical path² length, resource allocation, etc.). These indices are mostly shown through Gantt charts. However, there are more important project state indices; we will take interest about resources allocation and its prevention.

Resource allocation is the process of assigning the tasks to the appropriate resources. During the project planning, it can happen that more tasks are assigned to resource then the resource is able to handle. This is called *resource overloading* and it could lead to the problems during the project realization. Most of the project management software can detect and alert this problem, but in many cases it is the role of the user to maintain it. Now, you can say: “It is not so bad, just move problem tasks to the later start times..”. You are right, but what if there are more overloaded resources and more then one hundred tasks and precedence constraints among themselves? It is still so easy to reschedule them, considering the precedence constraints and conserving the minimal project length? The answer is “No” and therefore the good-class project management software allow user to maintain this problem using scheduling algorithms instead of manual correction.

Example 5.1: Building the house

Assume the Building the house example presented in the Chapter 1. Let’s modify it slightly:

- Suppose that there are two workers groups. The group A works on the project all the time (is able to work on all tasks). The group B is smaller then the group A and specializes only for building the rooms (e.g. living room).
- Consider, that after finishing the cross buntions, we are able to build the kitchen, bedroom, bathroom and living room simultaneously.

The table below shows a simple project plan for building a house (note that task durations are not real).

²Critical path is the series of events (tasks) depending on each other and whose durations directly determine the length of the whole project.

Task number (assigned group)	Task	Duration	General precedence constraint
1 (group A)	Foundations	15 days	
2 (group A)	Walls	10 days	Starts at least 5 days after finishing the foundations (task 1).
3 (group A)	Roof	12 days	Starts after finishing the walls (task 2).
4 (group A)	Doors and windows	2 days	Starts after finishing the roof (task 3).
5 (group A)	Cross buntions	2 days	Starts after finishing the doors and windows (task 4).
6 (group B)	Kitchen	2 days	Is able to start immediatelly after finishing the cross buntions (task 5).
7 (group A)	Bedroom	3 day	Is able to start immediatelly after finishing the cross buntions (task 5).
8 (group B)	Bathroom	2 days	Is able to start immediatelly after finishing the cross buntions (task 5).
9 (group A)	Living room	3 day	Is able to start immediatelly after finishing the cross buntions (task 5).

The project has been designed in Microsoft Project 2007. Figure 5.2 shows resource Group A allocation graph. Take notice of resource overallocation on 10th February.

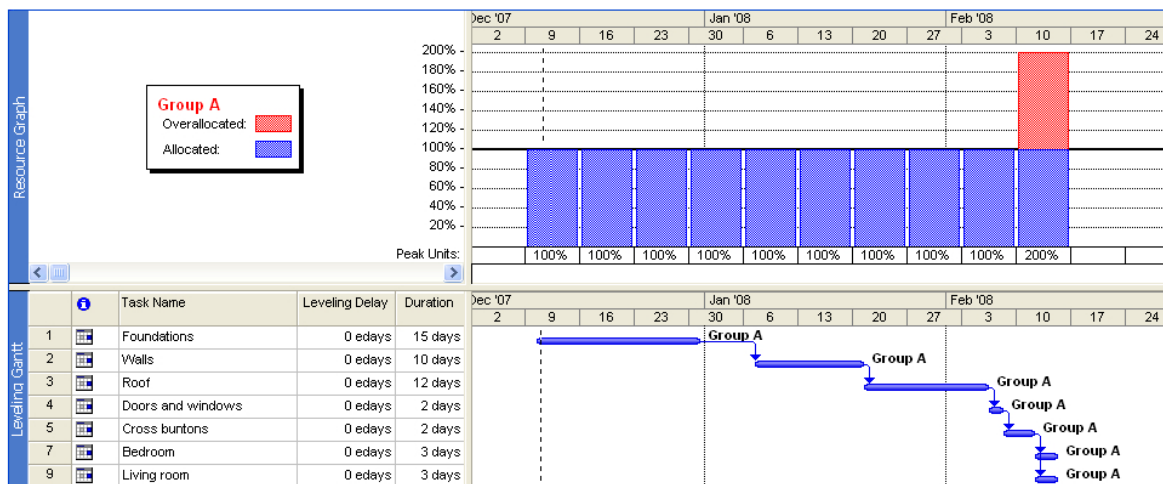


Figure 5.2: Example 5.1: resource Group A overallocation

The same situation, but for resource Group B, is shown in Figure 5.3. Remark that the resource is either overallocated.

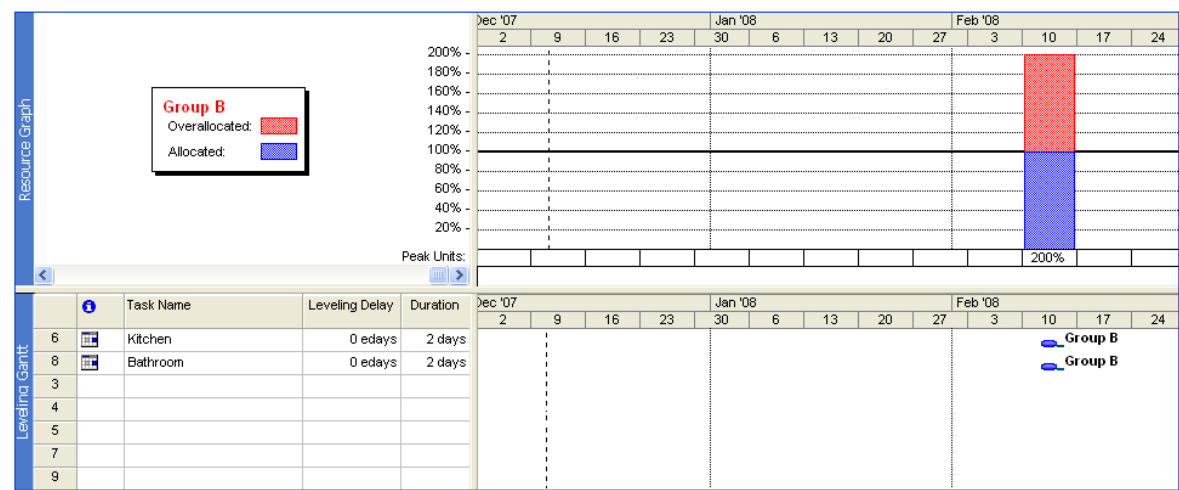


Figure 5.3: Example 5.1: resource Group B overallocation

Now, we will solve the overallocation problem using the function from the menu Tools → Resource Leveling. In the next two figures, we can see that the algorithm implemented in the Resource Leveling function has moved the problem tasks to the later start times in order to preserve the resources capacities.

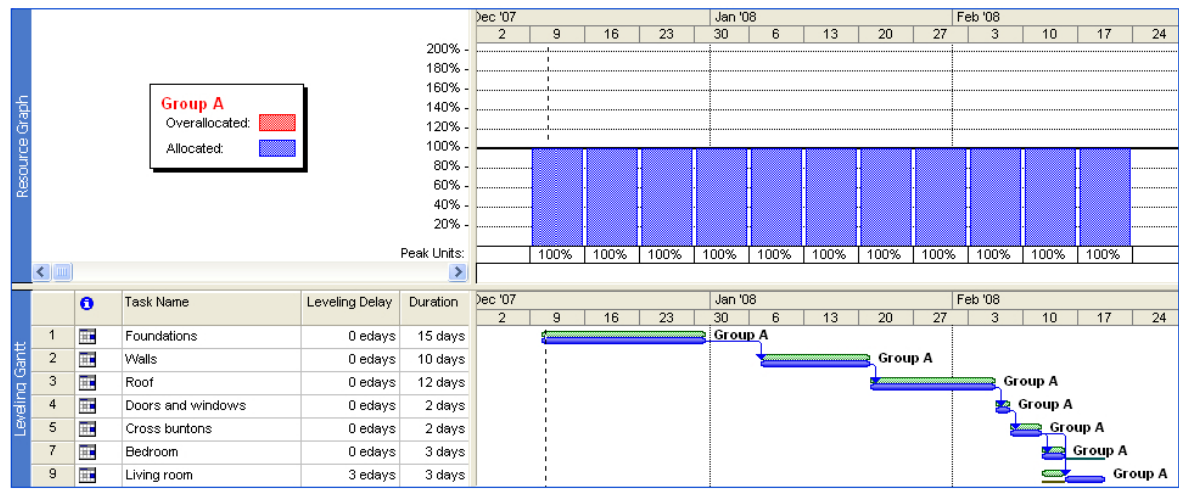


Figure 5.4: Example 5.1: Group A, Resource Leveling function result

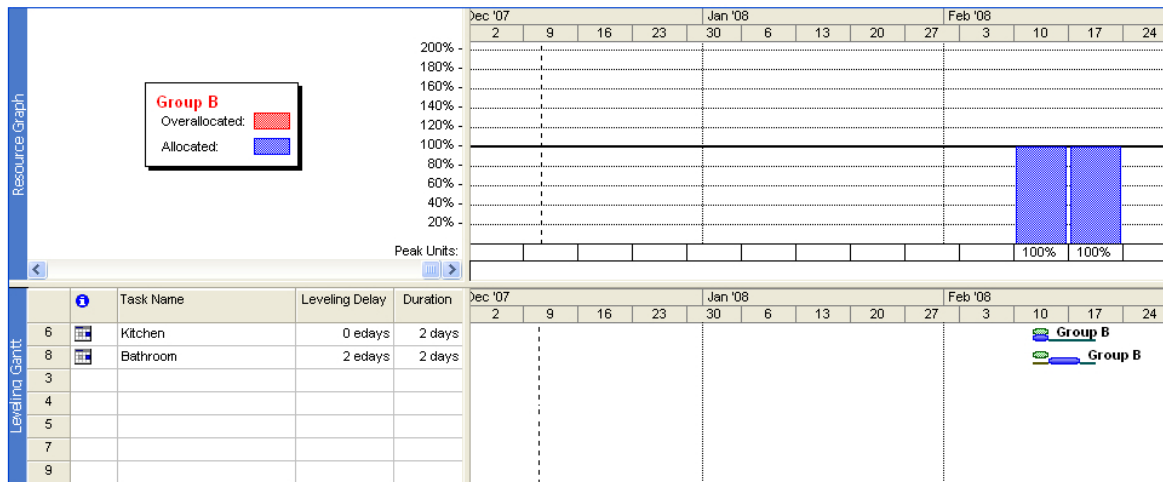


Figure 5.5: Example 5.1: Group B, Resource Leveling function result

Understanding the Example 5.1, we should have a clear idea about resource leveling as a solution of the overallocating problem. In the next section, we will concern about an open-source management software called GanttProject and we will extend it for the scheduling algorithm (introduced in the Chapter 4) as a Resource Leveling function.

5.3 GanttProject

As we have mentioned before, GanttProject is an open-source project management software developed under the GPL license. It allows to simple create the tasks, assign them to the human resources and easily establish the dependencies between the tasks. For project control, GanttProject renders the project using two charts: a Gantt chart, displaying all tasks, dates and dependencies between the tasks; and a Load chart for resource, where all resources and their loads are shown. Here we are able to see whether some resources are overallocated or not. GanttProject is also able to print charts, generate PDF and HTML reports, exchange data with Microsoft(R) Project(TM) and spreadsheet applications³.

Although, there are many various project management applications, GanttProject offers an open-source software that is easy to learn, and is still extending its set of the basic features. Another advantage resides in its cross platform. GanttProject is a Java based application that runs on Windows, Linux, MacOSX and other operating systems supporting Java.

³The typical spreadsheet application is MS Office

Figure 5.6 shows the interactions between the user and the basic GanttProject functions.

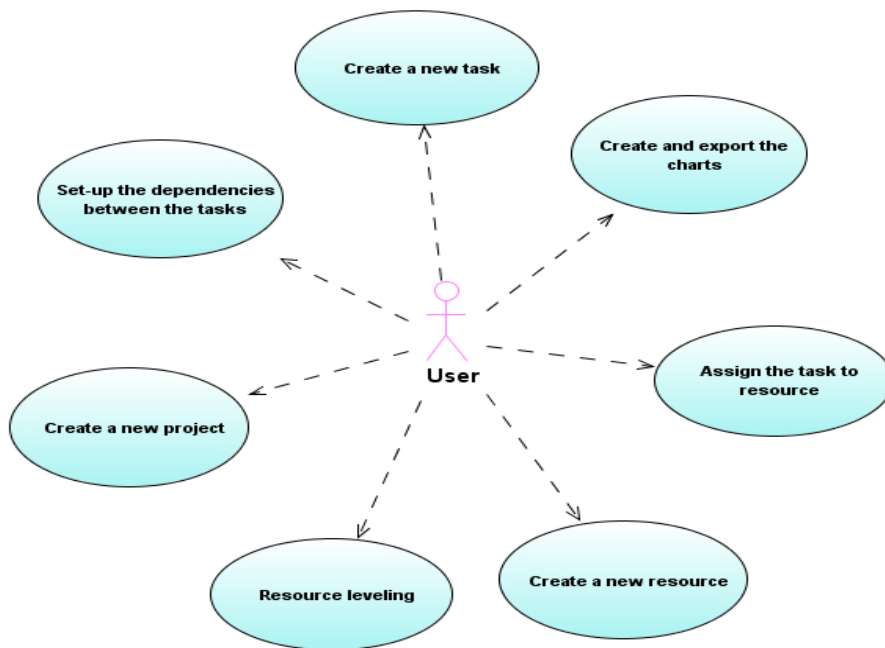


Figure 5.6: The basic GanttProject functions

5.3.1 GanttProject structure

GanttProject contains of 21 packages and 24 subpackages. The main project structure (without subpackages) is shown in Figure 5.7.

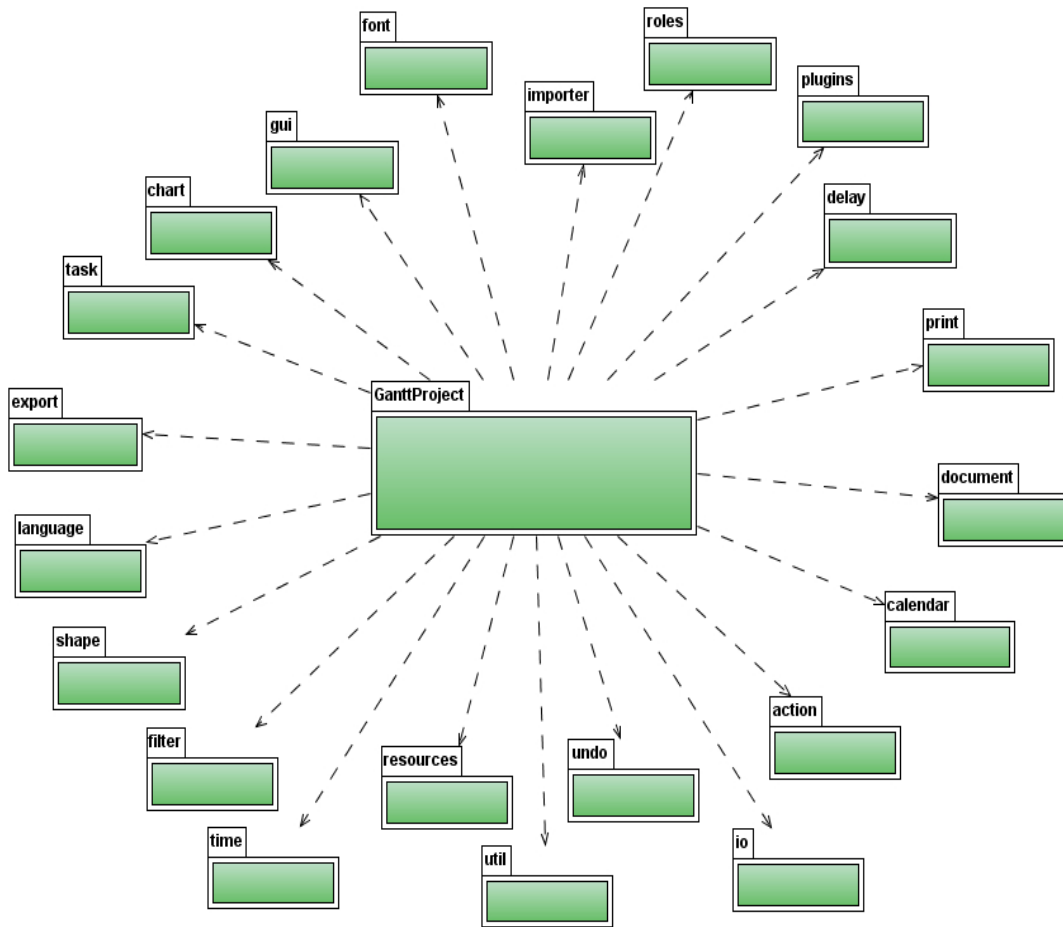


Figure 5.7: The GanttProject structure

In this section, we will explain only those packages and subpackages, that are in touch with our implementation (extension for the scheduling algorithm as a Resource Leveling function):

package net.sourceforge.ganttproject

Contains all general classes (e.g. classes for project variables initialization, graphical user interface (GUI), etc.).

package net.sourceforge.ganttproject.action

Includes all action listeners⁴ : in our case, for the GUI items (e.g. buttons).

package net.sourceforge.ganttproject.task

⁴Listener is the class that handles an event (e.g. after clicking the button, it calls the appropriate method)

This package holds the classes for the manipulation with the tasks (e.g. create or delete task, assigning the resources, task duration, task start and task end date, etc.).

package net.sourceforge.ganttproject.task.dependency

Contains the classes that concern about dependencies between the tasks (e.g. type of the relationship, the weight of the edge between themselves, the set of the successors, the set of the predecessors, etc.).

package net.sourceforge.ganttproject.task.algorithm;

Covers the classes for handling the tasks events (e.g. moving tasks to another start times, critical path computation, etc.).

package net.sourceforge.ganttproject.resource

This package holds the classes for the manipulation with the resources (e.g. create or delete resource, resource load etc.).

5.3.2 Scheduling algorithm implementation

We decided to implement the scheduling algorithm, described in the Chapter 4, as the Resource Leveling function. Therefore, there is no need to explain the algorithm idea (and how it works) again. We will rather concern about mounting the algorithm in to Ganttproject as a Java environment. There are three main issues to do so:

1. Create a menu item to invoke the Resource Leveling function.
2. Create a Java class to handle the event when the Resource Leveling function is called.
3. Obtain the input data for the scheduling algorithm from the Ganttproject structure.
4. Create a Java class implementing the scheduling algorithm function.
5. Involve the newly created classes to the overall GanttProject structure.

5.3.2.1 Create a menu item to invoke the Resource Leveling function

Creating the menu item for invoking the Resource Leveling action is not a big deal. It is just adding some lines of code into the main GanttProject package, into the *GanttProject.java* file. We have created a new item and assigned it to the appropriate menu. The next step was to “tell” the button what to do, when the user clicks on it. For this purpose, we used “event action handler” which is the class containing the instructions that are invoked after clicking the menu item. The last think to do was to add the “Resource Leveling” item into the *i18n.properties* file, which is the language database file containing all english translations for each menu item.

5.3.2.2 Create a Java class to handle the event when the Resource Leveling function is invoked

Creating the event action handler class for the Resource Leveling menu item, *ResourceLevelingAction.java*, involves addressing following three subissues:

- Obtain the input data for the scheduling algorithm from the Ganttproject structure.
- Call the scheduling algorithm class with the specified input parameters.
- Use the results, returned by the algorithm, to move the tasks to start times where the resources are not overallocated.

For acquiring the data from the project, we had to decide what kind of data we would need. If we look back to the Chapter 4, we can see that the input parameters for the algorithm are: taskprocessing times \mathbf{p} , the precedence constraints that are represented by the matrix \mathbf{W} and the assigned resources \mathbf{R} . For this purpose, following methods were implemented (note, that the objective of this section is not to describe these method in detail, rather the design of the classes that need to be implemented to perform the Resource Leveling function).

- short[] getTaskProcessingTimes() - returns the one-dimensional array of the task processing times \mathbf{p}
- float[][] getW() - returns the two-dimensional array of the task processing times \mathbf{W}
- float[][] getTaskResources() - returns the two-dimensional array of the assigned resources \mathbf{R}

The next issue was to design how to call the scheduling algorithm. We decided to create the new class, *ResourceLevelingAlgorithm.java*, that implements the algorithm function. The *ResourceLevelingAction* class then creates a new instance of the *ResourceLevelingAlgorithm* class with the \mathbf{p} , \mathbf{W} and \mathbf{R} as the input parameters. The output parameter of the algorithm specifies the task start time for each task, and the resource where each task should be processed.

Figure 5.8 shows the relationship between the *ResourceLevelingAction* class and the *ResourceLevelingAlgorithm* class.

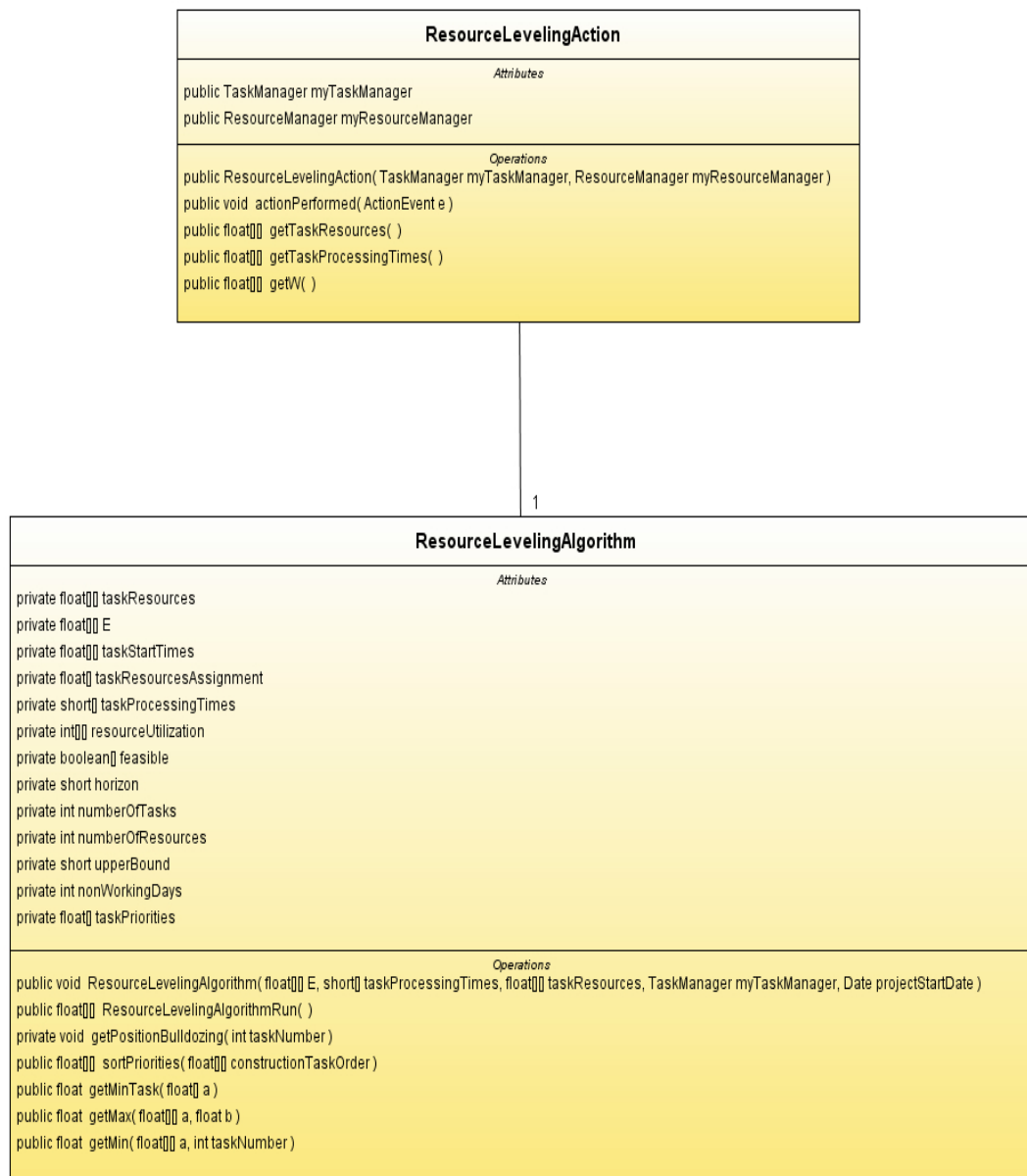


Figure 5.8: The relationship between the *ResourceLevelingAction* class and the *ResourceLevelingAlgorithm* class

We have introduced both classes. Now, becomes a question. How does it work together? Figure 5.9 shows the whole call sequence.

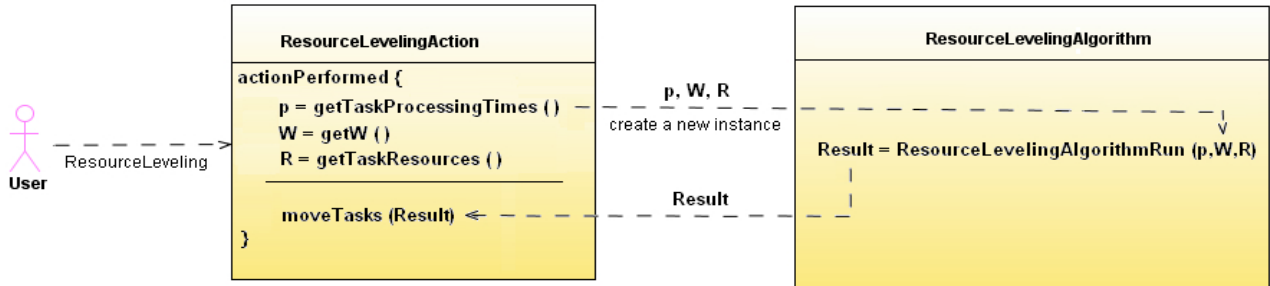


Figure 5.9: The call sequence between the *ResourceLevelingAction* class and the *ResourceLevelingAlgorithm* class

After clicking the Resource Leveling menu item, an instance of the *ResourceLevelingAction* class is created and the *actionPerformed* method invoked. The *getTaskProcessingTimes*, *getW* and *getTaskResources* methods are called to acquire the appropriate data form the project. After this, the new instance of the *ResourceLevelingAlgorithm* class is created and the *ResourceLevelingAlgorithmRun* method, with *p*, *W* and *R* as the input arguments, is called. The result of the *ResourceLevelingAlgorithmRun* method is then handed by the *actionPerformed* method of the *ResourceLevelingAction* class. The method moves the tasks according to the output argument (Result) of the *ResourceLevelingAlgorithmRun* method, in order to prevent the resources oveallocation.

5.3.2.3 Create a Java class implementing the scheduling algorithm function

As it was mentioned before, the implemented algorithm is the same as it was closely described in the Chapter 4. While the algorithm performs the same function and uses exactly the same idea, we decided that it is not necessary to describe it once again. For those ones, who have not heard about the implemented algorithm, see the Chapter 4.

5.3.2.4 Involve the newly created classes to the overall GanttProject structure

As it was mentioned before, GanttProject contains more than twenty Java packages and subpackages. These packages are logically named and ordered by the purpose of the classes involved in each package. We decided to place the *ResourceLevelingAction* class into the package *net.sourceforge.ganttproject.action.resource*. The *ResourceLevelingAlgorithm* class is placed in the package *net.sourceforge.ganttproject.resource.algorithm*.

Figure 5.10 shows the updated part of the GanttProject structure.

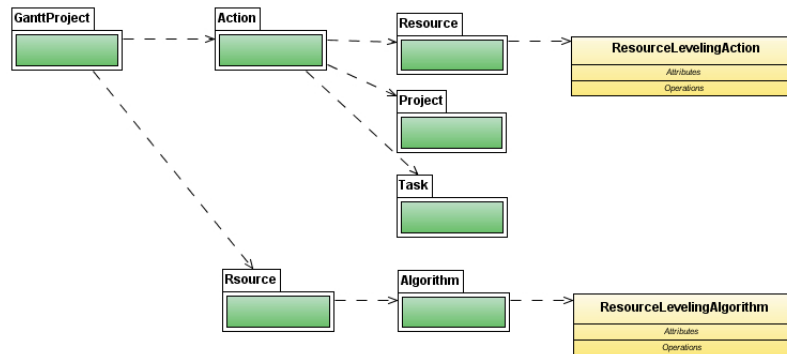


Figure 5.10: Involving the new classes into GanttProject structure

5.3.3 Compiling and running GanttProject

This subsection gives a step-by-step tutorial how set the project to be compilable in the easiest way. During my first project compilation, I met some troubles that I want to prevent other users (developers).

Before introducing the individual steps how to compile it, we will briefly summarize some requirements that need to be met:

- The official Java Development Kit (JDK) is Sun's **JDK 1.4.2**. GanttProject is compilable only with this version and probably is not compilable by uncertified Java compilers.
- The official development tool is **Eclipse 3.1**. It is also compilable with the newer versions or with Sun's Netbeans, but Eclipse 3.1 is the easiest way how to set the project in order to compile it. If you would like to use another development tool (e.g. the one, that you are familiar to), then you have to use ANT⁵ (<http://ant.apache.org/>) for building the project. Nevertheless, if you decided to use another development tool then setup the ANT in the following way:

In the source code distribution, there is a folder called *ganttproject-builder* containing build artifacts. There is a *build.xml* with default target *dist-bin*.

⁵ANT is a tool for building the Java applications.

Target dist-bin

Builds a ready to use binary distribution and puts it into dist-bin subfolder. No need to edit any files.

Target dist-src

Builds a source code distribution and puts it into dist-src subfolder. No need to edit any files.

- All codes are stored in CVS. GanttProject CVS server is hosted on SourceForge (<http://sourceforge.net/>). Most of the development tools have already integrated CVS clients to download a project from the repository. If your tool doesn't support it, then try to download an external CVS client (e.g. TortoiseCVS).

Compiling GanttProject

This step-by-step tutorial is only for Eclipse 3.1 over the Microsoft Windows XP operation system. It is possible that it will also suit for other Eclipse versions, but we do not give any guarantee.

1. step: Create a new project from CVS

Invoke File → New Project, select CVS → Checkout project from CVS, and create a new repository location using information shown below.

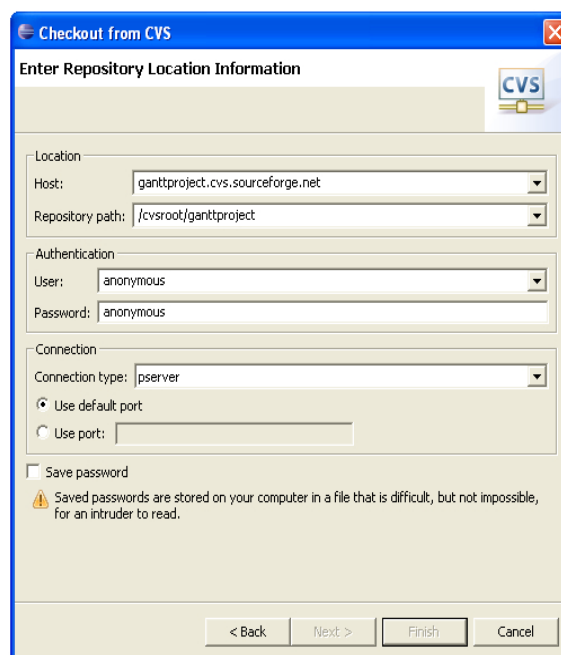


Figure 5.11: Create a new project from CVS

- Host : “ *ganttproject.cvs.sourceforge.net* ”
- Repository path : “ */cvsroot/ganttproject* ”
- User/Password : “ *anonymous/anonymous* ”
- Connection type : “ *pserver* ”

On the next page, you may type in the text field: Use specified module name, the value “ *ganttproject* ”. If you press “Next” button three times, you’ll see a page titled Select tag. Press the button “Refresh tags” and then choose node Branches → BRANCH_2_0_0 (see Figure 5.12).

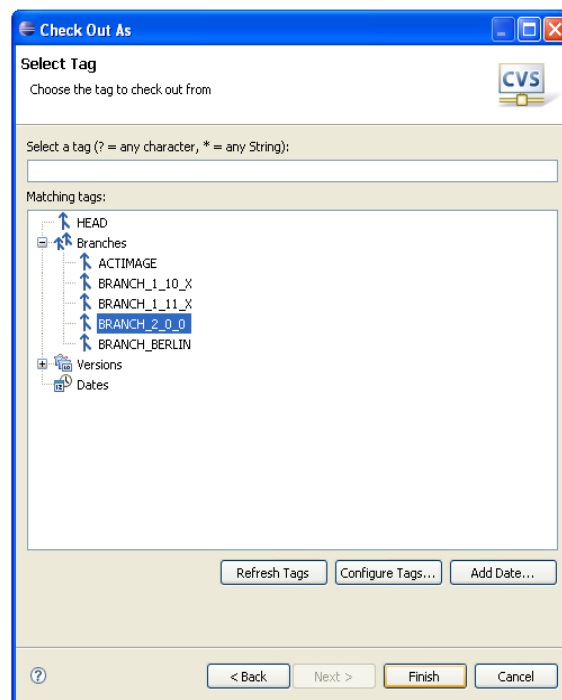


Figure 5.12: Checkout the project from CVS repository

2. step: You should change the compiler in your Eclipse enviroment to 1.4 version. Get back to Java (default) perspective and invoke Window → Preferences, page Java → Compiler. Uncheck Use default compliance settings and select 1.4 in all three combo boxes.

Running GanttProject as an Eclipse Application

1. step: Create new Eclipse application configuration. Menu Run → Run..., select Eclipse Application in the tree view and press button New. Then fill in accordance with Figure 5.12. Do not forget to change the Runtime JRE to Java 1.4.2 (in our case → j2re1.4.2_15), because as it was mentioned before, it works only with JDK 1.4.2.

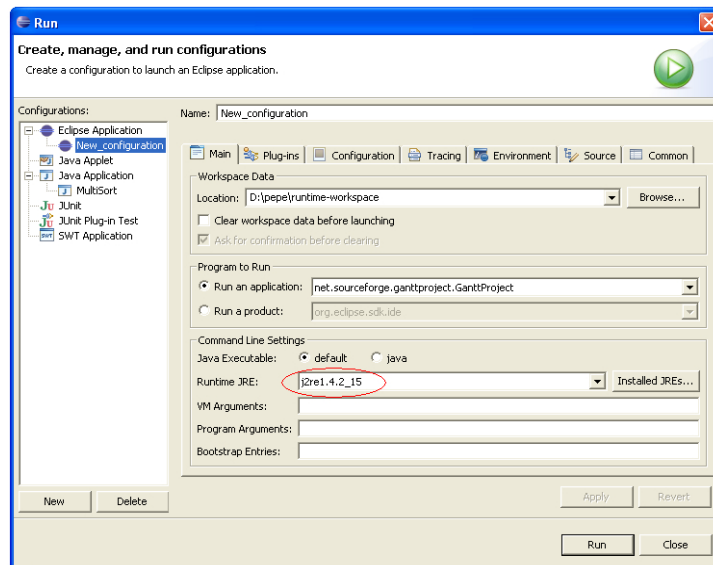


Figure 5.13: Running GanttProject

- Run an application : “ *net.sourceforge.ganttproject.GanttProject* ”
- Runtime JRE : “ *j2re1.4.2_15* ”

All other settings leave to default.

2. Run the project by pressing the Run button.

5.3.4 GanttProject vs. Microsoft Project 2007

There are many differences between GanttProject and Microsoft Project 2007. The essential difference lies in supported user functions and features. MS Project has a long time tradition and supports much more features then GanttProject. However, there are many other differences, this section concerns about the differences between Resource Leveling functions. There are two main differences:

1. GanttProject doesn't care about the assignment of the tasks to the resources before running the Resource Leveling function. The newly created schedule might assign to each task different resource. This is done because the algorithm is trying to find a first free resource to assign the task, in order to minimize total project length.
2. MS Project is unable to bind two tasks with forward and backward edge all at once (see Figure 5.14).

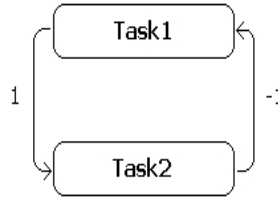
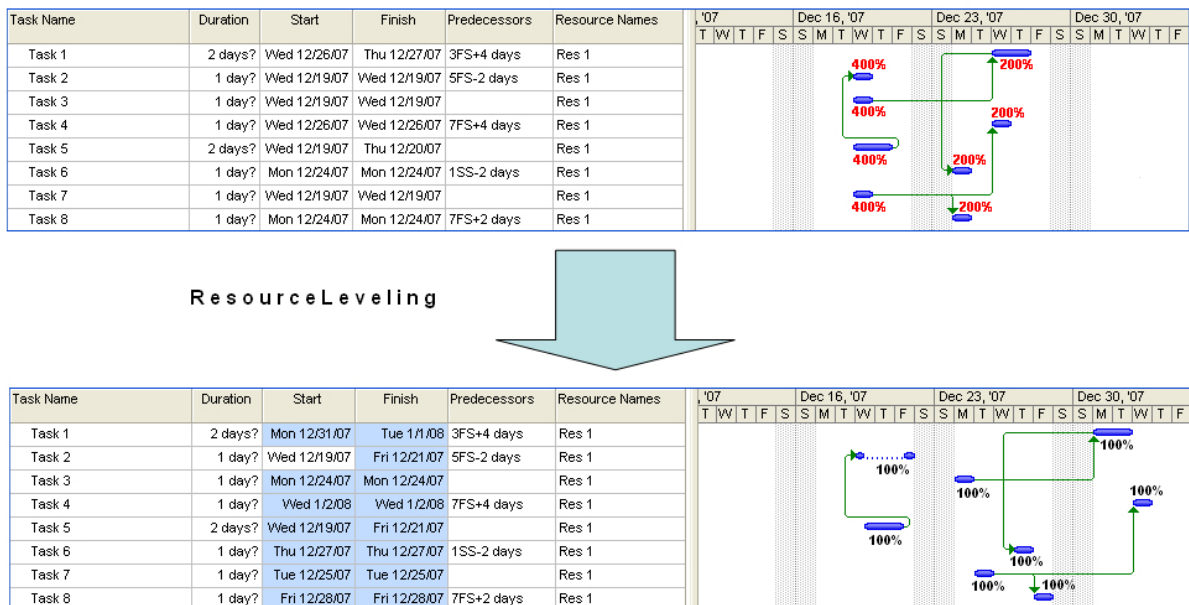


Figure 5.14: Microsoft Project, two-task problem example

Now, let's present the project that shows the differences between MS Project 2007 and GanttProject Resource Leveling algorithms⁶. Figures 5.15 and 5.16 show the same project before and after invoking the Resource Leveling algorithm, for both software products. We can see that the resulting schedule for GanttProject is one day shorter than for MS Project 2007.

MS Project 2007



⁶Algorithms that solve the RCPSP/max problem along the RLP criterion

Figure 5.15: MS Project 2007, Resource Leveling function example

GanttProject

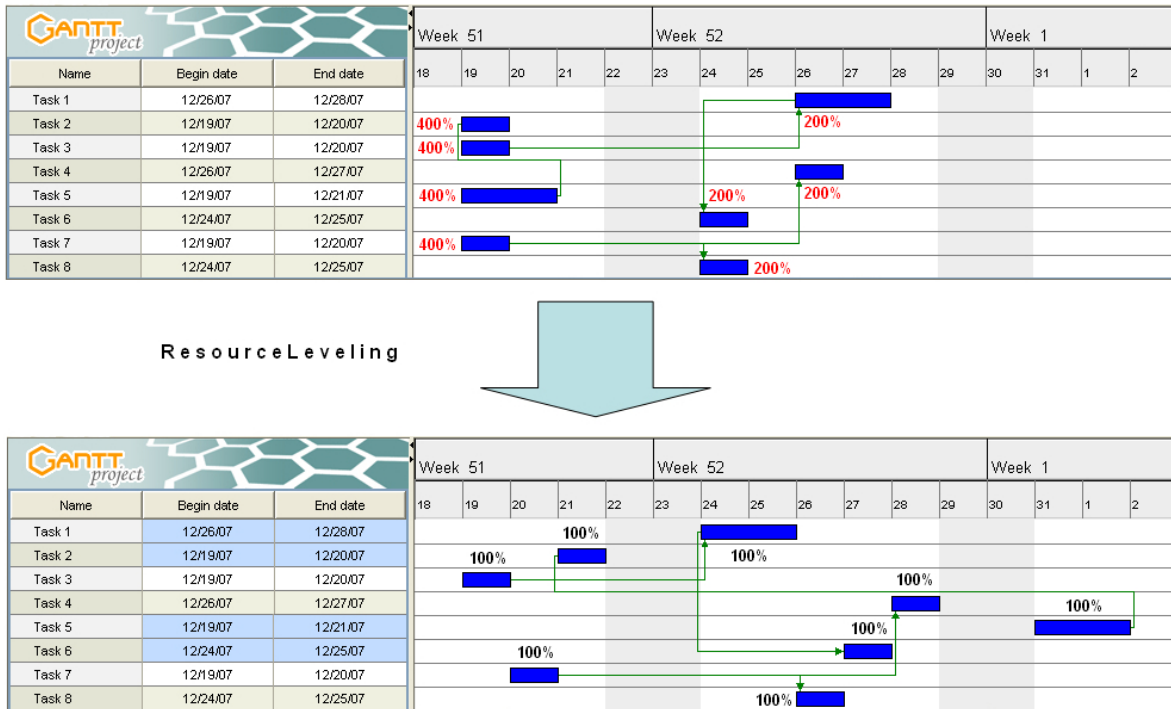


Figure 5.16: GanttProject, Resource Leveling function example

Chapter 6

Experimental Results

This chapter presents the experimental results of the implemented algorithm and its modification. The algorithms were tested in the Matlab environment along following two criteria:

1. C_{max} , the length of the resulting schedule.
2. *CPU time*, time required to find a resulting schedule.

The experiments were performed on the various sets of RCPSP/max instances generated by ProGen/Max (Schwindt, 1995). Each set contains 185 problem instances. The results are compared to the ILP algorithm (Sucha a Hanzalek, 2004) along both criteria. This chapter is divided into two parts. The first one concerns about setting up the ProGen/Max and the hardware performance where the algorithms were tested. The second one gives an information about the GUI for testing the instances and presents the experimental results.

6.0.1 Generating the instances

All the instances were created using the ProGen/Max (see Figure 6.1) which is the random instance generator for three different types of the resource-constrained project scheduling problems with minimal and maximal time lags: the resource-constrained project scheduling problem RCPSP/max, the resource-leveling problem RLP/max, and the resource investment problem RIP/max.

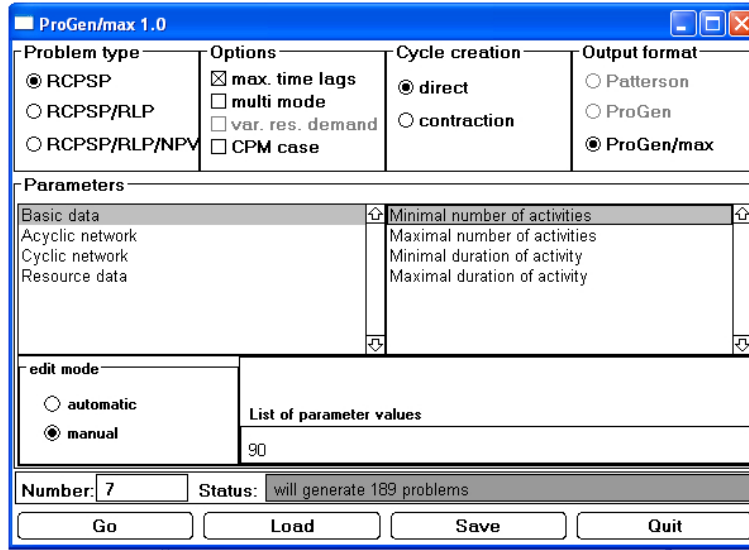


Figure 6.1: Progen/Max

The software package generates the problem instances, which are run through both algorithms. We have generated three different sets of instances, using the following settings:

1st set

- Minimal number of activities (n^{min}): 10
- Maximal number of activities (n^{max}): 80
- Number of resources (m): 1
- Minimal duration of activity (p_j^{min}): 1
- Maximal duration of activity (p_j^{max}): 15
- Minimal percentage of maximal time lags: 0.05
- Maximal percentage of maximal time lags: 0.15

2nd set

- Minimal number of activities (n^{min}): 80
- Maximal number of activities (n^{max}): 130
- Number of resources (m): 1

- Minimal duration of activity (p_j^{min}): 1
- Maximal duration of activity (p_j^{max}): 15
- Minimal percentage of maximal time lags: 0.05
- Maximal percentage of maximal time lags: 0.15

3rd set

- Minimal number of activities (n^{min}): 10
- Maximal number of activities (n^{max}): 80
- Number of resources (m): 1
- Minimal duration of activity (p_j^{min}): 1
- Maximal duration of activity (p_j^{max}): 15
- Minimal percentage of maximal time lags: 0.1
- Maximal percentage of maximal time lags: 0.3

6.0.2 Testing the instances

Both algorithms were implemented and tested in Matlab R2007a environment and were run on a 2000MHz Intel Pentium Core 2 Duo processor, with 2GB RAM and Microsoft Windows XP Professional installed.

6.0.2.1 Running the benchmarks

Each benchmark instance generated by ProGen/Max is saved in a separated file. The structure of these files is organized in a ProGen/Max format and is not so easy to be “understood” for our implemented algorithms. Therefore, there was a need to parse these files in order to obtain the data needed as an input for our algorithms. The input parameters for both algorithms include:

- Task processing times p .

- Task dependency matrix \mathbf{W} .

For acquiring the above mentioned data, the Scheduling Algorithm Tester for Matlab was developed (see Figure 6.2). It allows importing more than one Progen/Max output file at time. The imported files are then parsed and obtained data (\mathbf{p}, \mathbf{W}) are used as an input for the tested algorithms. This is not the only feature of the developed Scheduling Algorithm Tester. It also allows to choose the algorithms (from the checkbox) which the user would like to run the tests. After finishing all tests, the Scheduling Algorithm Tester saves data into “.mat” file and shows the benchmark graphs representing the above mentioned criteria.

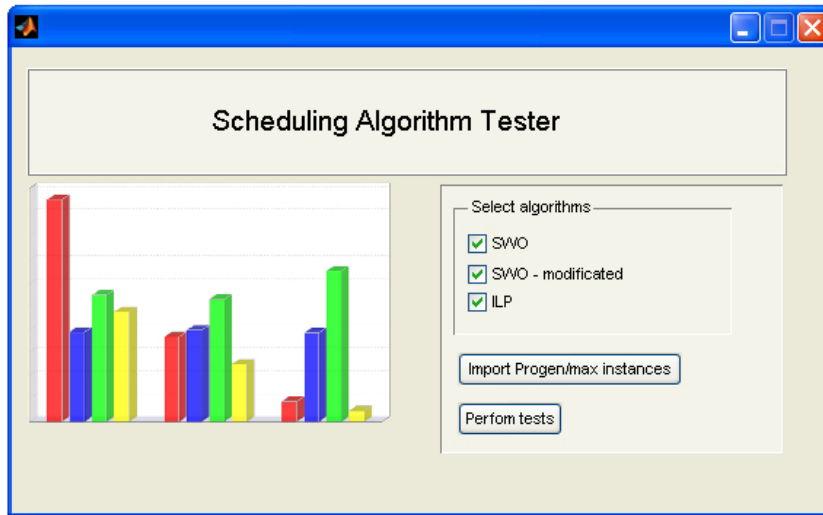


Figure 6.2: Scheduling Algorithm Tester

6.0.2.2 The benchmark results

The results are shown in the order already mentioned criteria and compared to ILP algorithm. For better text lucidity, we decided to denote the implemented algorithm as ALG. Here, we would like to remind that the main differences between the implemented algorithm and its modification rely on omitting certain parts of the code and on changes in the algorithm prioritization phase.

The C_{max} criterion

Figures bellow show the comparison results of the implemented algorithms to ILP and to themselves:

The comparison of ALG and modified ALG to ILP, tested set #1:

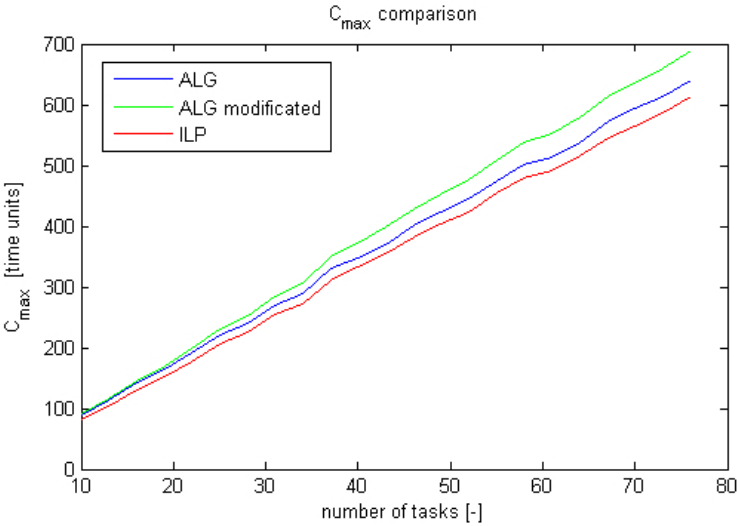


Figure 6.3: The C_{max} comparison

The comparison of ALG and modified ALG to ILP, tested set #2:

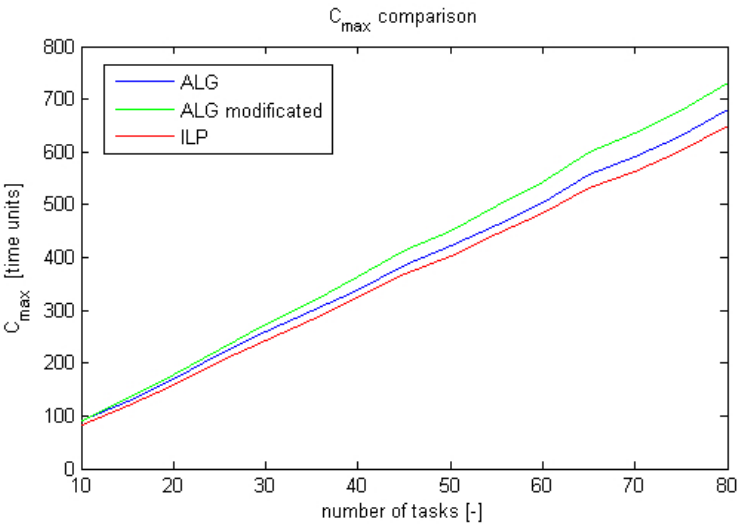


Figure 6.4: The C_{max} comparison with higher percentage of maximal time lags

The comparison of ALG to modified ALG, tested set #1:

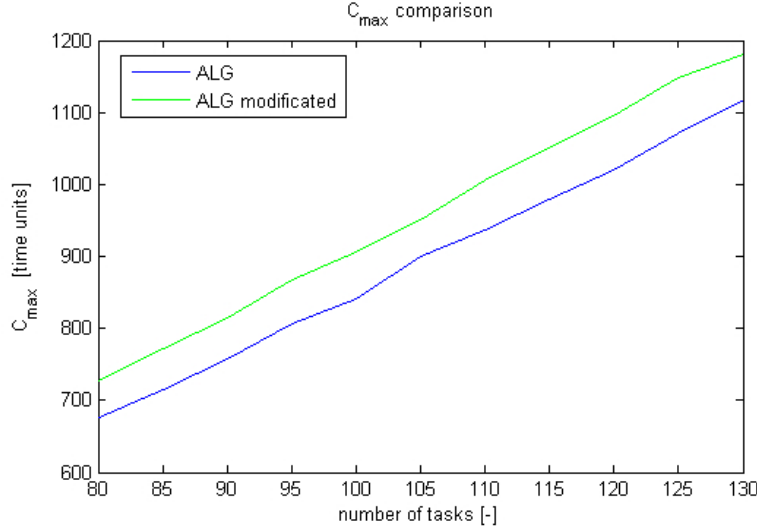


Figure 6.5: The C_{max} comparison of ALG and modified ALG for high scale problem instances

In Figure 6.3 and 6.4, it can be seen that the most effective is ILP, then ALG and at the end modified ALG. This is done, because ALG and modified ALG are heuristic and they are not always able to find the best solution. The modified ALG is even less effective than ALG (see Figure 6.5), because it omits some parts of code that are trying to reduce the total length of the feasible schedule at the price of higher *CPU time*.

The *CPU time* criterion

The comparisons are shown in figures below:

The comparison of ALG and modified ALG to ILP, tested set #1:

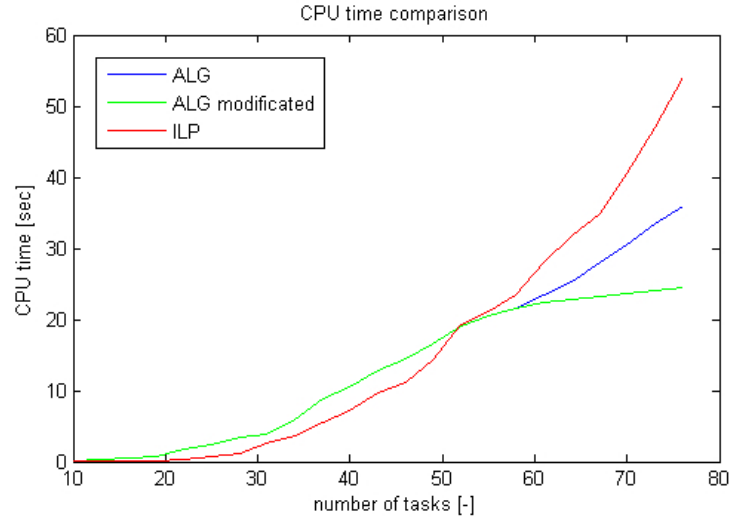


Figure 6.6: The *CPU time* comparison

The comparison of ALG and modified ALG to ILP, tested set #2:

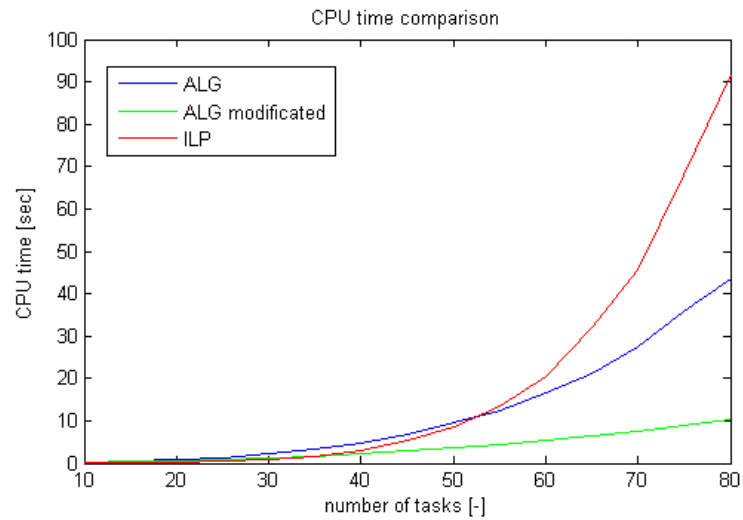


Figure 6.7: The *CPU time* comparison with higher percentage of maximal time lags

The comparison of ALG to modified ALG, tested set #1:

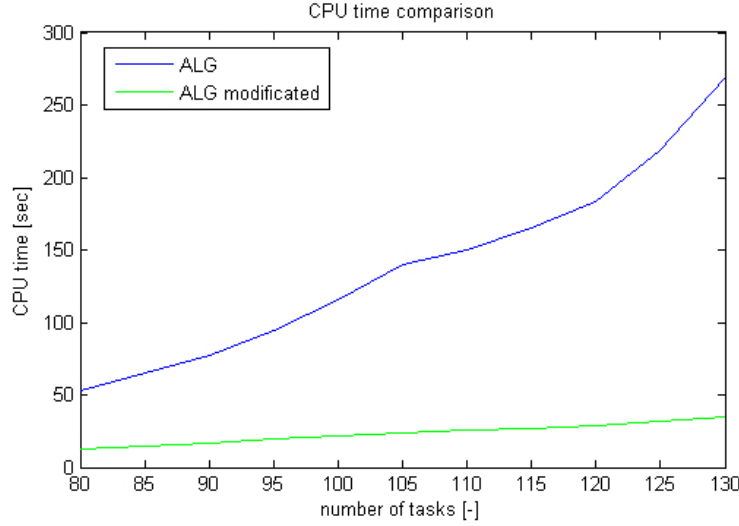


Figure 6.8: The *CPU time* comparison of ALG and modified ALG for high scale problem instances

The graphs in Figure 6.6 and 6.7 present that ILP is the most time-consuming, then the ALG and at the end again modified ALG. The reason is the same as in the previous criterion. The ALG and modified ALG are heuristics. In advance, the modified ALG omits the LeftBulldozing part of the code which is the recursive called function and increases the *CPU time*. The less *CPU time* consumption of the modified ALG, seen in Figure 6.8, is the big advantage, especially in large scale scheduling problems, where the C_{max} is not so important.

Chapter 7

Conclusions

The main purpose of this thesis was to find, analyze and implement a scheduling algorithm (heuristic) suitable as a Resource Leveling function for an open-source project management software GanttProject. After choosing the appropriate algorithm, it has been implemented and subsequently tested in Matlab R2007a environment. We also decided to modify the algorithm in order to increase its performance, concretely to decrease the *CPU time*, at the price of increasing the C_{max} . The testing sets were generated as a RCPSP/max problem instances using the ProGen/Max scheduling utility (Schwindt, 1995). Both algorithms were tested along the following criteria: C_{max} and *CPU time*; and compared to the results of ILP algorithm (Sucha a Hanzalek, 2004). The conclusion of these experiments was quite clear. Even if, the C_{max} of the implemented algorithms is greater than the C_{max} of ILP, the *CPU time* is much less for more then sixty tasks. At this place, we would like to note, that the modified algorithm, in comparison to the original one, consumes less *CPU time* (finds a feasible solution faster) at the price of the increased C_{max} . This is a big advantage in solving the large scale RCPSP/max problems, where the C_{max} is not so important.

The results satisfy the requirements, because we were looking for an algorithm that was able to solve the large scale RCPSP/max in the short time. For the problems containing a fewer tasks, we can still use the BaB or ILP algorithms. Respecting the experimental results, we decided to implement the modified algorithm into GanttProject. The rest of the thesis is concerned about the involving the algorithm implementation into the overall GanttProject structure. The algorithm has been implemented as the Resource Leveling function and compared to the Microsoft Project 2007. Even though, GanttProject is not as famous and sophisticated project management software as MS Project 2007 is, and GanttProject Resource Leveling function setting are not as extraordinary as MS Project

2007 are, our implemented algorithm behaves very well and in certain special cases was able to find a feasible solution while MS Project 2007 wasn't (forward and backward edge all at once). In cooperation with GanttProject administrators, all source-codes have been uploaded to SourceForge patch tracker, and are waiting to be approved as a part of the future version.

Future works

Extend the GanttProject Resource Leveling function options. This includes:

- To allow the user to level the resources based on the task ID or task priorities.
- To allow the user to level the resources only for a certain time period (from date to date).
- To allow the user to set the option - running the Resource Leveling function do not change the assignment of the tasks to the resources.
- Extend the algorithm for the preemption feature (e.g. to allow Resource Leveling function to maintain the tasks which duration is more than five days).

Bibliography

- Brucker, P., Hilbig, T. a Hurink, J. (1999), ‘A branch and bound algorithm for a single-machine scheduling problem with positive and negative time-lags’, *Discrete Applied Mathematics* **94**(1-3), 77–99.
- Caseau, Y. (1997), ‘Using constraint propagation for complex scheduling problems’, **1330/1997**, 163–166.
- Cesta, A., Oddi, A. a Smith, S. (2000), A constraint-based method for project scheduling with time windows, Technical Report CMU-RI-TR-00-34, Robotics Institute, Carnegie Mellon University, Pittsburgh, PA.
- Damay, J., Quilliot, A. a Sanlaville, E. (2007), ‘Linear programming based algorithms for preemptive and non-preemptive RCPSP’, *European Journal of Operational Research* **127**(3), 1012–1022.
- Demel, J. (2002), *Grafy*, Academia.
- Demeulemeester, E. L. a Herroelen, W. S. (1997), ‘A branch-and-bound procedure for the generalized resource-constrained project scheduling problem’, *Management Science* **45**(2), 202–212.
- Hartmann, S. (1999), ‘Self-adapting genetic algorithms with an application to project scheduling’.
- Hurink a Keuchel (2001), ‘Local search algorithms for a single-machine scheduling problem with positive and negative time-lags’, *DAMATH: Discrete Applied Mathematics and Combinatorial Operations Research and Computer Science* **112**.
- Joslin, D. E. a Clements, D. P. (1998), Squeaky wheel optimization, in ‘Proceedings of the Fifteenth National Conference on Artificial Intelligence (AAAI-98), Madison, WI’, pp. 340–346.

- Schwindt, C. (1995), ‘Progen/max: A new problem generator for different resource-constrained project scheduling problems with minimal and maximal time lags/’.
- Smith, T. B. a Pyle, J. M. (2004), ‘An effective algorithm for project scheduling with arbitrary temporal constraints’, pp. 544–549.
- Sucha, P. a Hanzalek, Z. (2004), ‘Scheduling with start time related deadlines’, *IEEE Conference on Computer Aided Control Systems Design* .
- Tormos, P. a Lova, A. (2001), ‘A competitive heuristic solution technique for resource-constrained project scheduling’, *Annals of Operations Research* **102**, 65–81.