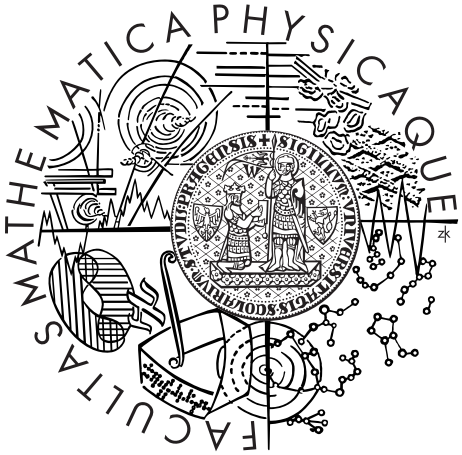
Charles University in Prague
Faculty of Mathematics and Physics

MASTER THESIS



Tomáš Kypta

Search Strategies for Scheduling Problems

Department of Theoretical Computer Science and Mathematical Logic

Supervisor of the master thesis: doc. RNDr. Roman Barták, Ph.D.

Study programme: Informatics

Specialization: Theoretical Computer Science

Praha 2011

I would like to thank my supervisor doc. RNDr. Roman Barták, Ph.D. for his leading of my master thesis, for his many valuable advices, suggestions, and for providing materials for the thesis.

I declare that I carried out this master thesis independently, and only with the cited sources, literature and other professional sources.

I understand that my work relates to the rights and obligations under the Act No. 121/2000 Coll., the Copyright Act, as amended, in particular the fact that the Charles University in Prague has the right to conclude a license agreement on the use of this work as a school work pursuant to Section 60 paragraph 1 of the Copyright Act.

In ..... date ..... signature

Název práce: Prohledávací strategie v rozvrhovacích problémech

Autor: Tomáš Kypta

Katedra: Katedra teoretické informatiky a matematické logiky

Vedoucí diplomové práce: doc. RNDr. Roman Barták, Ph.D., Katedra teoretické informatiky a matematické logiky

Abstrakt: V předložené práci porovnávám prohledávací strategie pro řešení rozvrhovacích problémů z pohledu programování s omezujícími podmínkami. Těžiště práce je věnováno rozvrhovacím problémům obsahujícím alternativní úlohy. V práci jsou jednak rozebrány různé již publikované způsoby modelování těchto problémů, dále pak jsou popsány a experimentálně porovnány prohledávací strategie pracující s těmito modely. Porovnáván je zejména vliv strategií na rychlost práce řešiče v závislosti na typu a velikosti dat. Jako vedlejší efekt práce studuje možnosti řešení rozvrhovacích problémů obsahujících alternativní úlohy pomocí řešiče Choco, který byl pro implementaci experimentů použit.

Klíčová slova: prohledávací strategie, rozvrhovací problémy, omezující podmínky, alternativní aktivity

Title: Search Strategies for Scheduling Problems

Author: Tomáš Kypta

Department: Department of Theoretical Computer Science and Mathematical Logic

Supervisor: doc. RNDr. Roman Barták, Ph.D., Department of Theoretical Computer Science and Mathematical Logic

Abstract: In the present work I compare the search strategies for solving scheduling problems from the view of constraint programming. The thesis is focused on scheduling problems containing alternative activities. An analysis of previously published various ways of modelling the problems is provided, next description and experimental comparison of search strategies targetting these models is provided. The influence of strategies on the speed of the solver is studied primarily. As a sideeffect the work studies the ways how Choco solver, which was utililized for implementation of the experiments, can be used to solve the scheduling problems with alternative activities.

Keywords: search strategies, scheduling problems, constraints, alternative activities

Contents

Introduction **4**

1 Constraint Satisfaction **5**
1.1 Constraint Optimisation 5
1.2 Search techniques 5
1.2.1 Complete Search Algorithms 6
1.2.2 Incomplete Search Algorithms 7
1.2.3 Search Heuristics 7
1.2.4 Branching Schemes 8
1.2.5 Branch and Bound Method 8

2 Scheduling Problems **9**
2.1 Basic Definitions 9
2.1.1 Resources 10
2.1.2 Precedences and Temporal Constraints 10
2.2 Scheduling Problems with Alternatives 11
2.2.1 PEX Temporal Networks 11
2.2.2 Temporal Networks with Alternatives 12
2.2.3 Custom Extension to PEX Temporal Networks 15
2.2.4 x-RCPSP 17
2.2.5 Alternative Resource Scheduling Problems 18

3 Implementation **20**
3.1 Used Choco Constraint Model 20
3.1.1 Activity variables 20
3.1.2 Constraints 21
3.1.3 PEX 22
3.1.4 Nested Temporal Networks with Alternatives 23
3.1.5 x-RCPSP Constraints 25
3.1.6 Optimality Criterion 25
3.2 Temporal Filtering in Temporal Networks with Alternatives . . . 26
3.2.1 Parallel Branching 26
3.2.2 Alternative Branching 26
3.3 Precedence Constraint Posting in Choco Solver 27
3.3.1 Modelling PCP with Alternatives 27

4 Search Strategies **29**
4.1 Classical Constraint Satisfaction Search Strategies 29
4.2 Basic Strategies for Scheduling Problems 30
4.2.1 Set Start Time 30
4.2.2 Precedence Constraint Posting 31
4.3 Traditional Approach to Scheduling with Alternatives 33
4.4 PEX 34
4.4.1 PEX Propagation 34
4.4.2 PEX and Heuristics 35

4.4.3 SumHeightPEX . . . 35
4.4.4 CBASlackPEX . . . 37
4.4.5 LJRandPEX . . . 38
4.5 Two Level Branching Strategy for Alternatives . . . 38
4.5.1 OptActTwoLevel Strategy . . . 39
4.6 Proposed Search Strategies for Alternatives . . . 40
4.6.1 OptActTwoLevelPair Strategy . . . 40
4.6.2 CBASlackNoPEX Strategy . . . 42
4.6.3 LJRandNoPEX Strategy . . . 42

5 Empirical Evaluation **43**
5.1 Experiments . . . 43
5.1.1 Experiments on n-TNA Data . . . 43
5.1.2 Experiments on PEXTNA Data . . . 44
5.1.3 Experiments on x-RCPSP Data . . . 44
5.1.4 Test Environment Configuration . . . 45
5.1.5 Performance Measurement . . . 45
5.2 Tested Hypothesis . . . 46
5.3 Results . . . 47
5.3.1 Results for PEXTNA Data . . . 47
5.3.2 Results for x-RCPSP Data . . . 48

Conclusion **49**

Bibliography **51**

List of Abbreviations **55**

Attachments **56**

A Test Program Documentation **57**
A.1 Test Program Documentation . . . 57
A.1.1 Test Program User Documentation . . . 57
A.2 Test Framework Documentation . . . 58
A.2.1 Project Structure . . . 58

B Graphs **61**
B.1 Graphs of n-TNA Results . . . 61
B.1.1 Performace of the Strategies . . . 61
B.1.2 Comparison of the Strategies . . . 63
B.2 Graphs of PEXTNA Results . . . 65
B.2.1 Performace of the Strategies . . . 65
B.2.2 Comparison of the Strategies . . . 69
B.3 Graphs of x-RCPSP Results . . . 71
B.3.1 Performace of the Strategies . . . 72
B.3.2 Comparison of Strategies by Data Configuration . . . 73

C Tables **75**
C.1 Tables with n-TNA Results 75
C.2 Tables with PEXTNA Results 78
C.3 Tables with x-RCPSP Results 82

D Compact Disc Content **84**

Introduction

The thesis deals with the problem of scheduling. Scheduling problems contain a huge scope of problem types. Some are easy to solve meanwhile there is a substantial portion of the difficult problems. Many practically important problems are hard to solve - they are typically NP-complete. Because of the significant practical impact of such problems there is intensive search for efficient solving methods. The unfavourable complexity situation caused that various approaches to enhancing scheduling abilities have been developed during last decades.

Constraint satisfaction is one of the most successful technologies used to solve scheduling and many other problems. This thesis presents an overview and an experimental comparison of solving techniques based on the constraint satisfaction technology. Since, the scheduling problems containing optional activities came recently to the spotlight due to their significant practical importance, the emphasis of the thesis is put on the scheduling problems with optional activities. Search strategies and heuristics targetting these problems are studied as the core goal of the thesis. Finally, some search strategies are proposed for solving the scheduling problems with alternatives and they are compared to other methods.

The experimental comparison is performed in Choco solver, which is a Java library constraint solver. Hence, part of the thesis is then focused on usage of Choco Solver for scheduling. Especially, the implementation of the presented scheduling strategies is mentioned and strengths and weaknesses of the scheduling capabilities of the solver are discussed.

Document Structure

The constraint satisfaction technology is first shortly described in chapter 1. In chapter 2 the scheduling problems which the thesis targets are described. Chapter 2 is devoted to the definitions of the scheduling problems that are studied in the thesis. Also, the notation of scheduling problems is unified here. In section 2.2 the scheduling problems with alternatives, which are studied primarily, are described.

Chapter 3 presents modelling of the scheduling problem presented in section 2.2 as constraint satisfaction problems. The modelling is described from the perspective of Choco solver.

Chapter 4 describes search strategies that are studied int the thesis. First some basic search strategies for scheduling problems are mentioned and then the strategies developed for scheduling problems with alternative activities are presented.

In chapter 5 the evaluation of the search strategies is provided. Section 5.3 presents and discusses the results obtained by the test program.

Appendix A provides documentation to the test program that was developed to evaluate the strategies. In appendix B the plotted graphs are situated and in appendix C the tables with results are places. In appendix D contents of the enclosed compact disc is listed.

The enclosed compact disc contains the electronic version of the thesis, the test program and test data with the generated output.

1. Constraint Satisfaction

First of all constraint satisfaction, as the underlining technology the thesis deals with, has to be explained in general.

The constraint satisfaction is a technology originating in artificial intelligence and initially designed for effective solving of combinatorial problems. The idea is to declaratively describe the problem and then use generic solving techniques. An overview of constraint satisfaction and its application for planning and scheduling can be found in [4, 5, 10].

Definition. Constraint satisfaction problem (CSP) *is a triple* $\Theta = (V, D, C)$, *where* $V = \{v_1, v_2, \ldots, v_n\}$ *is a finite set of variables,* $D = \{D_1, D_2, \ldots, D_n\}$ *is a set of domains for variables* $v_1, v_2, \ldots, v_n$, D_i *is a domain of variable* v_i, *and* $C = \{c_1, c_2, \ldots, c_m\}$ *is a finite set of constraints restricting the values that can be assigned to the variables at the same time. Constraint* c_i *is a relation defined on a subset* $\{v_{i_1}, v_{i_2}, \ldots, v_{i_{k_i}}\} \subseteq V$, $c_i \subseteq \{D_{i_1} \times D_{i_2} \times \ldots \times D_{i_{k_i}}\}$. *A* solution s *to a CSP is a complete assignment of the variables from* V *that satisfies all the constraints.*

In case the domains are finite discrete sets, we talk about *(pure) constraint satisfaction*. The constraint programming branch where the constraints are defined over infinite or more complex domains is called *constraint solving*. Constraint solving is not targeted in the thesis.

If a domain D is common for all the variables and contains exactly two elements, we talk about *Boolean constraint satisfaction problems*. It's worth noting that a pure CSP can be converted to an equivalent Boolean CSP using SAT encoding.

The solving techniques for constraint satisfaction involve several methods. A short summary of the solving techniques can be found in [4, 5].

1.1 Constraint Optimisation

Practical applications of CSP often require some high-quality solution. It is not sufficient to find any solution. The quality of solution, usually referred to as an *objective function* or, in the context of scheduling (see section 2), a *cost function*, can be defined by many different criteria. The goal is then to find a solution that minimizes (or maximizes) the given objective function. This class of problems is referred to as *Constraint Optimisation Problems*.

1.2 Search techniques

There exist several approaches to solving CSP. It can't be said that one technique outperformes all the other. Usually some techniques perform better for some problems while there might be different problems that the techniques fail to solve efficiently or even solve at all. Also a small change of a problem instance might cause a serious change in the performance of a solving technique.

A huge number of publications cover various search techniques. Since solving techniques for general CSPs are not a target of the thesis only a short summary of these techniques is presented here. It is based on the publications of Barták [4, 5, 10].

A CSP is usually solved by a systematic search through the solution space. As mentioned earlier in the pure constraint satisfaction there is a finite set of variables, a finite set of constraints and the domain of each variable is finite and discrete therefore the solution space is also finite. Although being finite the solution space is usually huge and search through it is computationally demanding. To reduce the solution space several sophisticated techniques have to be used. Some consistency inference (constraint propagation) techniques are being used to prune values before or during the search.

1.2.1 Complete Search Algorithms

A systematic search through all the variables and through all the assignments of values to the variables guarantee to find a solution or to prove that there isn't any solution for the problem. A systematic search of such algorithm can be represented by a search tree. The aim is to reduce the size of the search tree and the number of steps performed by the algorithm during search.

The most simple algorithm is *backtracking search*. In each step the algorithm takes the set of consistently instantiated variables and tries to extend the set with a new variable; a variable is taken and a value is tried being assinged for the variable. If the new value is consistent with all the constraints and the values of the previous variables, the process continues with the next unassigned variable. Otherwise another value is tried for the variable. If there is no other value for the variable a backtracking step is performed and the algorithm returns to the previously assigned variable. At the beginning the set of variables is empty. When the algorithm stops a complete assignement for all the variables is found or the algorithm returns with an empty set, the same one it started with. If an empty set was returned the search was unsuccessful and the problem has no solution, it is *inconsistent*.

The presented algorithm is probably the most simple systematic search algorithm for solving CSPs, it is called *chronological backtracking*. The algorithm can be simply improved. A *look-back algorithm* tries to use information about previous assignments. Because the previous assignments can impose inconsistencies in the currently processed variables, consistency checks between the current variable and the past variables are added. Therefore look-back algorithm can prune search through a sub-tree containing no solution. There are several examples of look-back algorithms, for example backjumping [21], conflict-directed backjumping [31], backmarking [20], learning [19].

Look-forward algorithms look forward to check consistency. Shortly the key is to recognize that a tested value for a variable can't be part of the solution because there exists a variable that doesn't have a compatible value. An example of look-forward algorithm is forward-checking [23].

1.2.2 Incomplete Search Algorithms

Although the complete search algorithms possess lots of desirable features, they still might be computationally expensive. Therefore another types of search algorithms vere developed - incomplete search algorithms. Incomplete search methods don't search the whole search space. An incomplete search algorithm might be also non-systematic, it might visit a state in the search space more than once.

Incomplete search algorithms can't guarantee finding all the solutions nor even to find a solution to a problem that is solvable. They also can't detect insonsistent problems. Their advantage lies on the speed of the computation. These algorithms are especially appropriate when searching for any solution to a problem.

These search methods are often referred to as metaheuristics. Lots of resolution paradigms can be covered here: evolutionary algorithms, local search techniques, etc. The best performing algorithms are based on constructive methods or on iterative repair methods. The constructive methods gradually extend a partial solution to a complete one. The iterative repair methods start with a solution, it frequently don't have to be a feasible solution, and incrementally modify it to acquire a better solution. Examples of such algorithms can be found for example in [29, 30].

1.2.3 Search Heuristics

The search heuristics help to fasten the search algorithm. During the search the critical point is the decision about which branch of the current search state to choose. The decision basically consists of two things - the variable and the value selection.

Variable Selection

In the backtracking-based search the variable selector selects a variable to be processed in the search step. It corresponds with a node in the search tree. The order in which the variables are processed can greatly influence the speed of the search. It changes the shape of the search tree and therefore it can change the number and "quality" of leaves of the search tree.

For the variable selection a general rule called *fail-first* is applied [23]. It says that the critical decisions are better to be taken first. The point is that the subtree originating from this decission should have more inconsistent values and therefore more values are pruned. If there is a solution in the subtree then it's not necessary to search through too much values and to backtrack too much. Otherwise if there is no solution in the subtree then the search would probably prune lots of values and the complete search through the subtree is supposed to be fast.

Some examples of variable selectors are shown in chapter 4 in the context of the search strategy they belong to. For example in the works of Caseau and Laburthe in [17] the most constrained activities (activities with the smallest domains) are selected. Generally it's better to select a variable with the smallest domain.

Value Selection

Value selection is the decision which value to choose after the variable selection was performed. A general rule called *succeed-first* is applied [33]. It says that the value leaving more possibilities should be selected. The point is that if there is a solution in the subtree then it's better to select a value that allows to find the solution faster. Also in the case of constraint optimisation, the obtained solution might be used by the solver to prune subtrees with worse overall cost function. More about algorithms for constraint optimisation is in section 1.2.5.

Some examples of value selectors are shown in chapter 4 in the context of the search strategy they belong to.

1.2.4 Branching Schemes

In complete search algorithms a *branching scheme* is an important factor of the efficiency of the search strategy. A branching scheme decides about the type of branching in search steps of the strategy. It decides about the number of options the value selector can choose from. It can be seen as resolving a disjunction. Typical branching scheme in classical strategies is *enumeration*. The scheme is enumerating all the values for each variable. Let a variable x has a domain $D = \{v_1, \ldots, v_n\}$. The selection can be assumed as an additional constraint $x = v_1 \vee \ldots \vee x = v_n$. Each choice in the search step corresponds with a value for the variable. Value selector then chooses a value from the domain D.

Another way of branching is *dichotomic branching* $x = v_1 \vee x \neq v_1$. In this case the branching factor remains small, but the number of search decissions greatly increases.

The last frequently used general branching scheme is *binary branching* which splits the domain in half $x \in \{v_1, \ldots, v_{\frac{n}{2}}\} \vee x \in \{v_{\frac{n}{2}+1}, \ldots, v_n\}$. This kind of branching assigns a value to a variable only in several search steps. Other search decissions only prunes the domain and possibly helps to detect a dead ends sooner.

There are many other types of branching schemes. They are typically part of a complex search strategy. Some examples of branching schemes are shown in chapter 4 in the context of the search strategy they belong to.

1.2.5 Branch and Bound Method

Branch and bound method is a search algorithm often used for constraint optimisation problems. It needs a heuristic function that maps a partial assignment of variables to a numerical value. The value is an estimate of the objective function for the best complete assignment that can be obtained from the partial assignment. Branch and bound is a backtracking based algorithm, after a search step of the algorithm (assigning a value to a variable), the heuristic function for the new assignment is computed. If the value exceeds the bound, the current subtree under the current partial assignment is pruned immediately. If the objective function is being minimized and the estimate computed by the heuristic function is smaller than the bound, the solver continues in processing the subtree.

2. Scheduling Problems

A *scheduling problem* is basically a problem of allocating *activities* to one or more time intervals and to one or more *resources*. There exists a wide range of classes of scheduling problems. An overview of classes of scheduling problems with their solving algorithms can be found in Brucker's book [15]. The book is organized according to well known three-field classification[1] coming from [22].

Two different terminologies can be seen in literature - *project scheduling* terminology and *machine scheduling* terminology. The thesis uses project scheduling terminology and notation that has its roots in [14]. Machine scheduling terminology comes from [18, 32]. In machine scheduling activities are called *operations* and resources are called *machines*, furthermore operations can be groupped in *jobs*. In our case we'll be considering situations where one operation equals one job.

2.1 Basic Definitions

Let's go back to the definition of the scheduling problems. For our purpose we can further narrow it and not consider preemption. A preemptive activity can be released from the resource it is allocated to at any time of it's execution and be restarted later. Activities we are considering has to be finished once started.

Now we define the scheduling problems we are dealing with more formally, let's have a set of resources $\mathcal{R} = \{R_1, \ldots, R_m\}$ and a set of activities $\mathcal{A} = \{A_1, \ldots, A_n\}$, every activity A_i has a processing time $p_{A_i} \in \mathbb{N}_0$. Each activity A_i is associated with a set of resources $Q_{A_i} \subseteq \mathcal{R}$, activity A_i can be allocated to any of the resources in Q_{A_i}. Unless otherwise stated, further in the text we'll be considering only cases where $|Q_i| = 1, \forall i \in \{1, \ldots, n\}$. Moreover, we'll be considering cases where activity A can have more than one set Q_A associated. Each activity can have $q_A \in \mathbb{N}_0$ such sets associated, we'll denote the sets $Q_A^j, j \in \{1, \ldots, q_A\}$. Such cases cover scheduling problems where the activity has to be allocated to all those q_A resources at the same time.

The scheduling problems require some kind of optimality criteria to be able to measure the quality of a schedule. The third field in the Graham classification [22] specifies the optimality criterion which is measured by a *cost function*. A cost function $f_A(t)$ is defined to measure the cost of completing activity[2] A at time t, $f_A : \mathbb{N} \to \mathbb{R}$. If the *finishing time* of an activity A is denoted by C_A, then a cost function called *makespan* is denoted by $C_{max} = \max\{C_A | A \in \mathcal{A}\}$. To define other cost functions more terms have to be introduced. A *release date* r_A is associated with activity A to define the time when the activity becomes available for processing. *Due date* d_A for activity A defines the time when the activity has to be finished. *Lateness* is defined as $L_A = C_A - d_A$, *earliness* $E_A = \max\{0, d_A - C_A\}$ and *tardiness* $T_A = \max\{0, C_A - d_A\}$. Activities might have different importance in processing, therefore a *weight* w_A of an activity A can be defined. Also activities

[1] The notation of the classification is $\alpha|\beta|\gamma$ where α specifies the resource environment, β specifies the job characteristics and γ denotes the optimality criterion.

[2] A cost function is normally defined to measure the cost of a job, but in our case jobs and activities correspond.

might have different importance of early or late completion, therefore *late cost* and *early cost* for each activity can be defined. In general, all data p_A, r_A, d_A, w_A, late cost and early cost are assumed to be integer. All these terms can be used in the definitions of various cost functions. Besides makespan a cost function called *maximum lateness* is the most important. It is defined as $L_{max} = \max_{A \in \mathcal{A}}\{L_A\}$. Other frequently used cost functions could be $\sum T_A$, $\sum w_A \cdot T_A$, $\sum E_A$ and many other. They can be found in for example in Brucker's book [15]. In further text, only makespan is referred to as the optimality criterion.

In classic definition, a schedule is a function $s : \mathcal{A} \to \mathcal{R} \times \mathbb{N}_0$ giving the target resource and the start times of operations. In our case we'll consider schedule as a function $s : \mathcal{A} \to \mathbb{N}_0$ since all the resource requirements $Q_A^i, i \in \{1, \ldots, q_A\}$ are compulsory.

In general, the aim of the scheduling problem is to find a *feasible schedule*. A schedule is called *feasible* if no two time intervals overlap on the same resource, all the constraints are satisfied, and the resource meets other problem-specific characteristics. A feasible schedule is *optimal* if it minimizes a given optimality criterion.

2.1.1 Resources

So far, we have defined the resources vaguely. For each resource R we define a *capacity* $cap_R \in \mathbb{N}_0$ and requirement $req_{A,R} \in \mathbb{N}_0^+$ of activity A of the resource. At any given time the total combined requirement of all the activities allocated to the resource at the time cannot exceed the capacity of the resource.

When a resource R has capacity $cap_R = 1$, we call it *unary resource* or *disjunctive resource*. It this case no two distinct activities can overlap an the resource. Formally, the following formula holds for unary resources:

$$s(A) + p_A \leq s(B) \text{ or } s(B) + p_B \leq s(A), \text{ if } Q_A = Q_B = \{R\} \text{ and } A \neq B$$

If a resource R has capacity $cap_R > 1$ we call it *cumulative resource*. The resource limitation can be formally defined by the next formula:

$$\sum_{A \in \{B \in \mathcal{A} | Q_B = R \,\&\, s(B) \leq t \leq s(B) + p_B\}} req_{A,R} \leq cap_R \quad \forall R \in \mathcal{R}\ \forall t \in [0, D].$$

2.1.2 Precedences and Temporal Constraints

Typical scheduling problem often include temporal relationships between activities. Therefore precedence relation between activities was introduced. The relation is denoted by $\ll$ symbol[3]. Precedence $A \ll B$ denotes that activity A must be finished before the activity B. Formally:

$$s(A) + p_A \leq s(B)$$

The precedence relation can be further generalized into temporal constraint (temporal link) between a pair of activities. A temporal link between two activities A and B is described by minimum distance $minDist_{(A,B)} \in \mathbb{N}_0$ and maximum

[3]In literature, reader might encounter different symbols for precedence relation such as $\rightarrow$.

distance $maxDist_{(A,B)} \in \mathbb{N}_0$. Formally, it is:

$$minDist_{(A,B)} \leq s(B) - (s(A) + p_A) \leq maxDist_{(A,B)} \tag{2.1}$$

2.2 Scheduling Problems with Alternatives

In classic scheduling problems the aim is to assign all activities their start times and resources which the activities will be executed on. Each activity must be executed. In the previously defined scheduling problems the resource which the activity is executed on is specified in advance. The only problem is to find the start time for each activity.

Many real life problems require more variability in the problem definition, especially splitting into more alternatives, therefore problems with *alternatives* were introduced. Alternatives basically divide into two types, *alternative activities* and *alternative recources*. The thesis targets scheduling problems with alternative activities, therefore they are discussed in the next sections in further deatail. Scheduling problems with alternative resrouces are shortly mentioned in section 2.2.5.

In scheduling problems with alternative activities the problem is to decide which activities exist in the solution and to find the start times for those activities that exist in the solution.

Next an overview of works about alternatives is presented in the section. The works are mentioned in approximate chronological order of their publication.

2.2.1 PEX Temporal Networks

One of the oldest works about alternative activities are the ones of Beck and Fox in [12, 13]. They studied alternative activities from the view of constraint-based scheduling - they tried to expand the scope of the constraint-directed scheduling techniques.

Now, we'll present only their representation of scheduling problems with alternatives. The proposed constraint-directed solving methods are further mentioned in chapter 4.4.

In their work, the scheduling problems with alternative activities are modelled via temporal networks. In the temporal networks nodes represent activities. Alternatives are introduces through special nodes - so called *XorNodes* and *AndNodes*. XorNodes are used to model alternative branching and AndNodes are used to model parallel branching. Moreover, XorNodes and AndNodes represent dummy activities used only to represent the branching of alternative process plans. An example of temporal network can be seen in figure 2.1, XorNodes are marked XOR, AndNodes are marked AND.

AndNodes

AndNode is a dummy node with zero processing time in the temporal network that has special semantics. The semantic of AndNode defines that if the AndNode exists in a solution then all non-XorNodes directly linked to it also exist. Similarly, if one of the non-XorNodes linked to an AndNode exist in a solution then the AndNode exist in the solution as well.

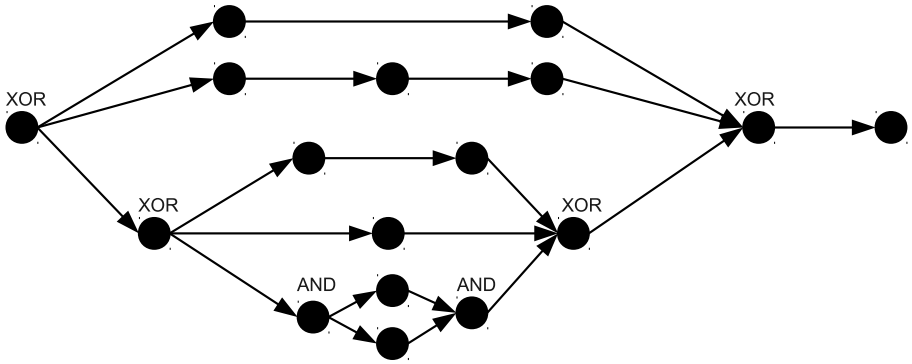


Figure 2.1: An example of a PEX temporal network.

Temporal perspective of an AndNode defines that the AndNode have to start after maximum earliest completion time of all the upstream connected nodes. Similarly, an AndNode have to end before minimum latest start time of all the downstream connected nodes.

XorNode

XorNode has, similarly to AndNode, zero length. The semantics of XorNode defines that if the XorNode exists in a solution then one and only one of the nodes directly connected upstream/downstream exists in the solution. If a directly connected node exists in a solution then the XorNode exists in the solution too.

Temporal perspective of a XorNode defines that the XorNode have to start after the minimum earliest completion time of all the nodes connected upstream and have to end before the maximum latest start time of all the nodes connected downstream.

Illegal Temporal Networks

We've seen an example of a PEX temporal network in figure 2.1. Although a wide range of temporal networks can be expressed using AndNodes and XorNodes, not all such networks are allowed. The limitation is posed to XorNodes. If the temporal network contains XorNode A with k upstream links, then it must contain corresponding XorNode B with k downstream links that is upstream to A. Similarly, if the temporal network contains XorNode C with l downstream links, then it must contain corresponding XorNode D with l upstream links that is downstream to C. An example of illegal PEX temporal network can be seen in figure 2.2.

The reason why such temporal networks are illegal lies in the context constraint satisfaction techniques proposed to solve PEX temporal networks, it is explained in section 4.4.

2.2.2 Temporal Networks with Alternatives

Another approach representing alternatives can be found in [6, 9]. The scheduling problems with alternatives are modeled via *temporal networks with alternatives*

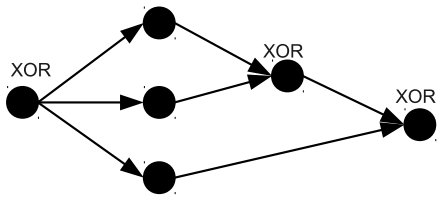


Figure 2.2: An example of illegal PEX temporal network.

(TNA) which are thoroughly studied in [7, 8, 9]. The whole approach of TNA is build on constraint-based scheduling as the underlying technology. The alternatives are defined through implicit constraints coming from the type of branching in the network. The work defines two kinds of branching in the network - *parallel branching* and *alternative branching*.

Compared with the PEX temporal networks (see previous section) parallel branching can be seen as an analogy to AndNode and alternative branching as an analogy to XorNode. But, in contrary to AndNodes and XorNodes, branching in TNA can be defined for each node and for each direction (upstream or downstream) separately. The concept of TNA is more general, besided the branching semantics, the PEX illegal network shown in figure 2.2 is legal in TNA.

Now, we formally define Temporal Networks with Alternatives. The definitions come from [7].

Definition. *Let G be an acyclic graph, a subgraph of G is called a* fan-out subgraph *if it consists of nodes $x, y_1, \ldots, y_k$ (for some $k \in \mathbb{N}$ such that each $(x, y_i), 1 \leq i \leq k$ is an arc in G. Similarly, a subgraph of G is called a* fan-in subgraph *if it consists of nodes $x, y_1, \ldots, y_k$ (for some $k \in \mathbb{N}$) such that each $(y_i, k), 1 \leq i \leq k$, is an arc in G. In both cases x is called a* principal node *and all $y_1, \ldots, y_k$ are called* branching nodes*. If $y_1, \ldots, y_k$ are all and the only successors of x in G then we call the fan-out subgraph complete (similarly, is defined complete fan-in subgraph).*

Definition. *A directed acyclic graph G together with its pair wise edge-disjoint decomposition into complete fan-out and fan-in subgrahps, where each subgraph in the decomposition is marked eighter as a* parallel *subgraph or an* alternative *subgraph, is called* P/A graph*.*

Definition. Temporal Network with Alternatives *is a P/A graph where each arc (x, y) is annotated by a pair of numbers $[a, b]$ (a* temporal annotation *where a describes the minimal distance between x and y and b describes the maximal distance, formally, $a \leq t_y - t_x \leq b$ where t_x denotes the position of node x in time.*

Alternatives are described by an *assignment* of 0/1 values to nodes of given P/A graph. If the node is assigned value 1, it means the node is selected. The value 0 means the node is not selected. The assignment is called *feasible* if the semantics restrictions of parallel and alternative subgraphs is fullfilled.

The semantics of parallel subgraph is similar to the one of AndNode, it defines that all nodes (both the principal node and all branching nodes) are either selected

(have value 1) or not (have value 0). The semantics of alternative subgraph is similar to XorNode, it defines that all nodes (both the principal node and all branching nodes) are not selected (have value 0) or the principal node and exactly one branching node are selected (have value 1) and all other branching nodes are not selected.

Scheduling Problems Represented by TNA

TNA with its alternative branching implicitly represent alternative processing plans. Nodes represent activities, temporal annotations on arcs represent temporal constraints between activities, where a corresponds with $minDist$ and b corresponds with $maxDist$. The only difference here is that activities can have non-zero duration. Temporal annotation is then described by (2.1).

To denote branching of the fan-out/fan-in subgraphs, let A be a principal activity (an activity representing principal node) in a fan-out subgraph, then $outBranch_A \in \{PAR, ALT\}$ represents type of branching of the subgraph. Similarly, $inBranch_A \in \{PAR, ALT\}$ represents type of branching of the fan-in subgraph with a principal activity A. The value PAR denotes parallel branching and the value ALT denotes alternative branching.

Also the sets of branching nodes should be defined for further usage. Let A be a principal activity of a complete fan-out subgraphs and let $B_1, \ldots, B_k$ be the activities representing all the corresponding branching nodes. Then $\mathcal{B}_A^{out} = \{B_1, \ldots, B_k\}$ denotes the set of all the activities representing all the branching nodes in a complete fan-out subgraph with principal node representing activity A. Similarly, $\mathcal{B}_A^{in} = \{B_1, \ldots, B_k\}$ denotes the set of all the activities representing all the branching nodes in a complete fan-in subgraph with principal node representing activity A.

An example of temporal network with alternatives can be seen in figure 2.3. The alternative branching is denoted with ALT, the parallel branching is denoted with an arch crossing arcs branching/joining in the node. The temporal constraints are written on the arcs.

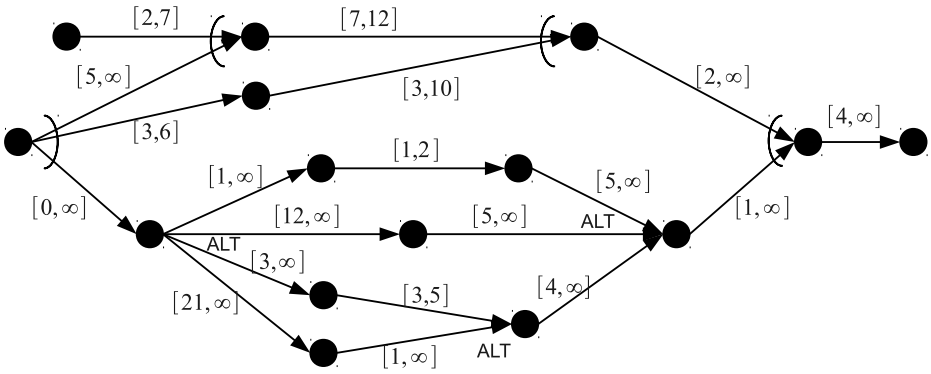


Figure 2.3: An example of a temporal network with alternatives.

P/A graph assignment problem

Definition. P/A graph assignment problem *is given by a P/A graph G and a*

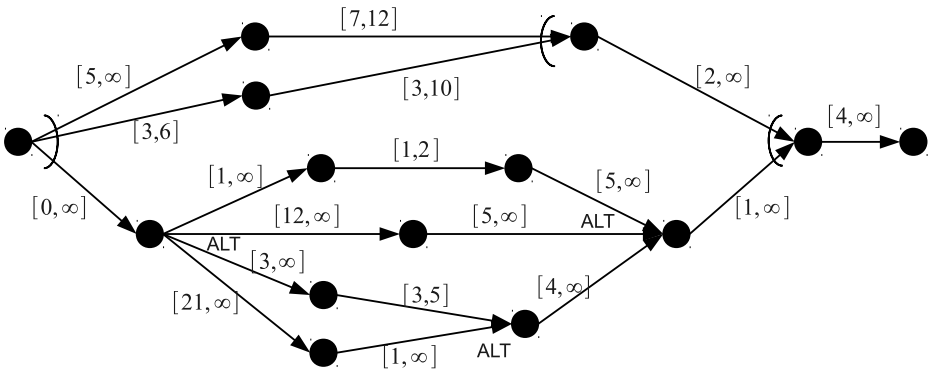


Figure 2.4: An example of a nested temporal network with alternatives.

list of nodes of G which are assigned value 1. The question is whether there exist a feasible assignment of 0/1 values to all nodes of G which extends the prescribed partial assignment.

In [7] Bartak and Čepek shows that the P/A graph assignment problem is NP-complete.

Nested Temporal Networks with Alternatives

A restriction from TNA called *Nested Temporal Networks with Alternatives* (n-TNA) was proposed in [8]. The restriction is motivated by real-life manufacturing scheduling problems. The main advantage of n-TNA is tractability (solvability in polynomial time) of the assignment problem.

Definition. *A directed graph* $G = (\{s, e\}, \{(s, e)\})$ *is a* (base) nested graph. *Let* $G = (V, E)$ *be a graph,* $(x, y) \in E$ *be its arc, and* $z_1, \ldots, z_k (k \in \mathbb{N})$ *be nodes such that neither* z_i *is in* V*. If* G *is a nested graph (and* $I = \{1, \ldots, k\}$ *then graph* $\hat{G} = (V \cup \{z_i | i \in I\}, E \cup \{(x, z_i), (z_i, y) | i \in I\} - \{(x, y)\})$ *is also a* nested graph.

In [8] Bartak and Čepek have shown that the nested P/A graph assignment problem is tractable.

An example of nested temporal network with alternatives can be seen in figure 2.4.

In the empirical evaluation of the search strategies in chapter 5 one of the data types used in experiments are Nested Temporal Networks with Alternatives.

2.2.3 Custom Extension to PEX Temporal Networks

The representation proposed by Beck and Fox is rather limited. It defines that AndNodes and XorNodes are dummy nodes with zero duration that doesn't represent any activity. As an extension, we redefine the concept of AndNodes and XorNodes by utilizing branching properties of TNA from section 2.2.2. Let's call the extension PEXTNA.

First of all, there's no reason why AndNodes and XorNodes can't represent regular activities with non-zero duration, therefore the extension allows AndNodes and XorNodes to represent real activities with non-zero duration. Still, if necessary zero-length dummy nodes are allowed in the network.

The properties of AndNodes and XorNodes are defined for both directions - upstream and downstream. By contrast, the definitions of Temporal Networks with Alternatives in the works of Barták and Čepek is more general, it allows a node to have different branching upstream and downstream. It's possible to replace AndNodes and XorNodes with branching properties of TNA and still fulfil requiments for corresponding XorNodes. Instead of an AndNode a parallel branching for fan-out/fan-in subgraphs will be used. An AndNode in the original definition would be then replaced by a node representing activity A having $outBranch_A = inBranch_A = PAR$. A XorNode will be replaced by an alternative branching for fan-out/fan-in subgraphs. Similarly, a XorNode in the original definition would be replaced by a node representing activity A having $outBranch_A = inBranch_A = ALT$.

With the extension in place, it's possible to have a node having parallel branching in one direction and alternative branching in the second direction. In PEX temporal network two separate nodes - one AndNode and one XorNode - would be necessary to represent such situation. Moreover, without the need for dummy XorNodes and AndNodes PEXTNA representation requires fewer nodes. An example of corresponding PEX and PEXTNA temporal networks can be seen in figure 2.5, it's clear from the picture, that some of the XorNodes and AndNodes can be removed and their branching properties transfered to regular nodes (nodes A and B in the example).

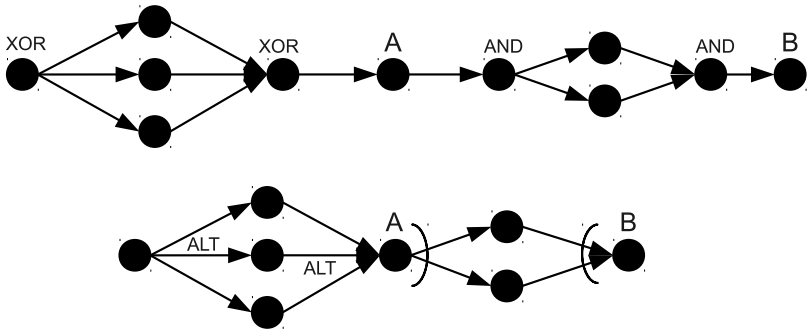


Figure 2.5: An example of corresponding PEX and PEXTNA temporal networks.

Despite the extension, the requirement for corresponding XorNodes, must be kept. Formally, defined with branching properties, let A be an activity and $outBranch_A = ALT$ and $|\mathcal{B}_A^{out}| = k$ then a corresponding activity B having $inBranch_B = ALT$ and $|\mathcal{B}_A^{in}| = k$ have to be present in the problem instance downstream to the activity A. And similarly, let A be an activity and $inBranch_A = ALT$ and $|\mathcal{B}_A^{in}| = k$ then a corresponding activity B having $outBranch_B = ALT$ and $|\mathcal{B}_A^{out}| = k$ have to be present in the problem instance upstream to the activity A. Explanation for the corresponding XorNodes requirement is provided in section 4.4.

With the extension in place, PEXTNA can be seen as a subclass of TNA (and n-TNA). In the empirical evaluation of the search strategies in chapter 5 the extension of PEX temporal networks is used.

2.2.4 Extended Resource-Constrained Project Scheduling Problem

A different approach to alternatives was presented in [25, 26]. The papers introduce an extension of *Resource-Constrained Project Scheduling Problem* (RCPSP) introduced in [16]. We'll now shortly present the concept of RCPSP to understand the extension.

Resouce-Constrained Project Scheduling Problem

RCPSP introduced two kinds of resrouces to scheduling problems: *renewable* and *nonrenewable* resources. Renewable resource were already presented, they are standard resources as defined in section 2.1.1. Nonrenewable resource are different, the problem instance specifies the numbers of units of resources of this kind that are available for the entire planning time. Once an activity consumes a unit of nonrenewable resource, this unit doesn't appear again after the end of execution of the activity. Although nonrenewable resources is an important feature of scheduling problems, problems with nonrenewable resources are not targeted in the thesis.

Alternative activities were introduced in RCPSP. A subclass of RCPSP called multi-mode resource-constrained project scheduling problems (MRCPSP) [16, 24] provides a way to define alternatives for activities. Each activity can be performed in one or more processing alternatives - modes. For an activity A, a set $\mathcal{M}_A$ of its modes is defined. An activity can have different processing times and resource requirements in different modes. Once an activity is started in one of its modes, it may not be interrupted, and the mode can't be changed. The support for alternatives is still very limited in MRCPSP, there is no way to define alternative execution paths consisting of several activities. The further extensions of RCPSP were then defined to target the problem.

Extended Resouce-Constrained Project Scheduling Problem

Papers by Kuster, Jannach and Friedrich [25, 26] further extends the concept of RCPSP into *extended resource-constrained project scheduling problem* (x-RCPSP). Instead of using modes for activities like in MRCPSP, the alternative process plans are implemented through the distinction between *active* and *inactive* activities, where only the active activities are considered in the resulting schedule. Activities can be dynamically activated and deactivated according to the predefined rules.

Formally, an instance of x-RCPSP scheduling problem is defined by a set of *potential* activities $\mathcal{A}^+ = \{A_0, A_1, \ldots, A_n, A_{n+1}\}$, where activity $A_0 \in \mathcal{A}^+$ and $A_{n+1} \in \mathcal{A}^+$ represent *mandatory* abstract start and end activities with a duration of 0 and no associated resource requirements. The *active activities* form a subset $\mathcal{A}' \subseteq \mathcal{A}^+$, the corresponding difference $\mathcal{A}^+ \setminus \mathcal{A}'$ contains the *inactive activities*. The set of renewable resources is denoted by $\mathcal{R} = \{R_1, \ldots, R_k\}$, for each resource R a constant amount of units cap_R is available. The set of all potentially relevant precedence constraints is denoted by $\mathcal{P}^+$, existence of an element $p_{A,B} \in \mathcal{P}^+$ states that activity $B \in \mathcal{A}^+$ cannot start before the end of activity $A \in \mathcal{A}^+$. The set of all potential resource requirements is denoted by the set $\mathcal{Q}^+$, an element

$q_{A,R} \in \mathcal{Q}^+$ is a constant denoting the required amount of resource $R \in \mathcal{R}$ by activity $A \in \mathcal{A}^+$. The sets $\mathcal{P}'$ and $\mathcal{Q}'$ are defined as restrictions on active activities of sets $\mathcal{P}^+$ and $\mathcal{Q}^+$ respectively.

The activity relations and dependencies are modeled by *activity substitutions* and *activity dependencies*. The set of potential activity substitutions is denoted by $\mathcal{X}^+$, existence of $x_{A,B} \in \mathcal{X}^+$ states that the process variation where activity $A \in \mathcal{A}'$ is replaced with activity $B \in \mathcal{A}^+ \setminus \mathcal{A}'$ is allowed. The set $\mathcal{M}^+$ denotes the mandatory implications of a deliberate activity state modification. The existance of $m^{\oplus}_{A,B}(m^{\ominus}_{A,B}) \in \mathcal{M}^+$ indicates that activity $B \in \mathcal{A}^+$ has to be (de)activated upon the (de)activation of activity $A \in \mathcal{A}^+$, the existence of $m^{\sqsubset}_{A,B} \in \mathcal{M}^+$ indicates that activity $B \in \mathcal{A}^+$ has to be deactivated upon the activation of activity $A \in \mathcal{A}^+$, and the existence of $m^{\sqsupset}_{A,B} \in \mathcal{M}^+$ indicates that activity $B \in \mathcal{A}^+$ has to be activated upon the deactivation of activity $A \in \mathcal{A}^+$.

A *schedule* s of an instance of x-RCPSP is a function $s : \mathcal{A}' \to \mathbb{N}_0$ assigning active activities their starting times. The activation state $\mathcal{A}'$ is considered valid, if and only if it can be derived from an original valid activation state through the application of the substitutions defined in $\mathcal{X}^+$, satisfying all the constraints defined in $\mathcal{M}^+$.

To acquire a meaningful model of a process, some extra constraints have to be defined. Namely the dependency consistency must hold - the model must forbid simultaneous activation and deactivation of an activity. Next, the precedence consistency - there must be no cyclic dependencies in the set of active precedence constraints $\mathcal{P}'$. And also, the requirement consistency as expressed in section 2.1.1 must hold.

Modeling of a problem in x-RCPSP provides better flexibility than MRCPSP does. The mode alternation[4] as it is provided in MRCPSP can be easily modeled in x-RCPSP. Moreover, activity insertion/removal, order change, and serialization/parallelization can be modeled in x-RCPSP. More details about defining these alternatives and the whole concept of x-RCPSP can be found in [25, 26]. This approach provides a pretty general way to model alternative execution paths and is independent from the underlying scheduling technology.

2.2.5 Alternative Resource Scheduling Problems

For the sake of completeness of the overview of scheduling problems with activities, alternative resources are shortly mentioned here, although they are not the target of the thesis. Back in section 2.1 we defined that an activity A is associated with a set of resources $Q_A \subseteq \mathcal{R}$, problems where $|Q_A| > 1$ are called *alternative resource scheduling problems*.

Modelling with Alternative Activities

The alternative resource scheduling problems can be easily modeled using alternative activities. For an activity A that requires some amount of an alternative resource, a bunch of alternative activities $A^{(1)}, \ldots, A^{(k)}$, where $k = |Q_A|$ can be created to model the resource alternatives. For each alternative activity $A^{(i)}$,

[4]Mode alternation provides a way to define different activity duration and resource requirements.

where $1 \leq i \leq k$ it holds that $|Q_{A^{(i)}}| = 1$. Resource alternatives are then treated exactly the same way like alternative activities.

Modelling with Cumulative Resources

The alternative resource scheduling problems can be, with some limitations, modelled also with *cumulative resources* (for definition see section 2.1.1). Assume that $|Q_A| > 1$ then we can define one resource $\tilde{R}$ with capacity $cap_{\tilde{R}} = |Q_A|$ (presuming that all the activities B requiring any of the resources in Q_A have requirements $req_{B,R} = 1$, where $R \in Q_A$).

3. Implementation

In chapter 1 general concepts of constraint satisfaction technology were presented and in chapter 2 we defined targetted scheduling problems. Scheduling problems can be naturally represented as constraint satisfaction problems. Before proceeding towards the core of the thesis - search strategies - we have to describe how the mentioned scheduling problems can be (and are in the test program) modelled as constraint satisfaction problems. This chapter describes modelling of scheduling problems with alternatives as CSPs. All the modelling is viewed from the capabilities of Choco solver [37], which is used for empirical evaluation of the strategies (more about experiments in chapter 5).

3.1 Used Choco Constraint Model

Modelling scheduling problems as constraint satisfaction problems usually resembles in its basic features across different constraint solvers. However, there might be slight differences in capabilities of some solvers and distinct sets of available constraints might be available in each solver.

The thesis targets scheduling problems with alternative activities, therefore we'll concentrate on modelling x-RCPSP and Temporal Networks with Alternatives (specifically n-TNA and PEX temporal networks) in Choco solver.

3.1.1 Activity variables

Traditional representation of scheduling problems involves encoding activities with several variables. Let $s_A \in \mathbb{N}_0^+$ be minimum start time and $p_A \in \mathbb{N}_0^+$ processing time (duration) of activity A and $D \in \mathbb{N}_0^+$ be an overall *due date* of the instance of a scheduling problem with alternatives. For each activity A the following variables are defined:

- Start time: $start_A \in \{s_A, \ldots, D\}$
- Completion time: $end_A \in \{s_A, \ldots, D\}$
- Duration: $dur_A = p_A$
- Validity: $val_A \in \{0, 1\}$
- Task variable: $task_A = (start_A, dur_A, end_A)$

Processing time of an activity A is usually constant, in the model we represent it with integer constant[1] dur_A.

Validity variables are present in the model only if the scheduling problem contains alternative activities. If the scheduling problem contains alternative activities and an activity A is known to be always valid then it's modelled with

[1] In Choco an integer constant is only a special type of integer variable. It's a constraint variable that's instantiated from the very beginning, it's domain is of size 1 and can't be emptied.

a boolean[2] constant $val_A = 1$. In temporal networks an activity A is known to be always valid if it has no upstream activities ($\mathcal{B}_A^{in} = \emptyset$) or no downstream activities ($\mathcal{B}_A^{out} = \emptyset$) linked to it. In x-RCPSP an activity A is always valid if it has no alternatives ($\{x_{A,B} | x_{A,B} \in \mathcal{X}^+\} = \emptyset$) and if it isn't dependent on any other activity ($\{m_{B,A} | m_{B,A} \in \mathcal{M}^+\} = \emptyset$).

Task variable is a construct of Choco solver, it's only a tuple of variables describing position of the activity in time, it consists of start, duration and end variable. The task variables are required by some constraints, example of such constarint is *disjunctive* constraint described later in section 3.1.2.

Other Variables

Some strategies utilize other helper variables. A pair validity variables might be, in some cases, added to the model. The denote the validity of a pair of activities. Let A and B be the activities, then the boolean variable $pairval_{AB}$ is defined:

$$pairval_{AB} = val_A \cdot val_B$$

3.1.2 Constraints

All the scheduling constraints defined in chapter 2 can be represented by constraints in constraint solvers, although some scheduling constraints require more complex constraint constructions utilizing some artifical variables.

Resource Constraints

Scheduling problems we are targetting contain only fixed resource requirements ($|Q_A^i| = 1, \forall A \in \mathcal{A}, 1 \leq i \leq q_A$). The only variability here is the possibility that an activity isn't present in the final schedule. The constraint we use to represent resource requirements must account for validity of activities.

Choco solver offers two kinds of constraints to represent resource requirements - *disjunctive* constraint for unary resources and *cumulative* constraint for cumulative resources. Both these constraint are capable to utilize information about the validity of the activities. Each such constraint binds together all the activities requiring the resource.

The experiments target scheduling problems containing only disjunctive resource. The model then contains for each resource R one disjunctive constraint in the form:

$$disjunctive_R\{(task_A, val_A) | (A, R) \in \mathcal{Q}\}$$

The disjunctive constraint used in the experiments utilizes these filtering algorithms[3]:

- Edge-finding
- Not-first/not-last

[2] In Choco boolean variables are special cases of integer variables defined with domains $\{0, 1\}$.

[3] Choco can also utilize *overload checking*, but not in one constraint together with the mentioned three filtering algorithms. It's necessary to add second disjunctive constraint with different parameters.

- Detectable precedences

The filtering algorithms are capable to deduce significant pruning of domains of the task variables. Since they are not the core of the thesis they are not further described (their description can be found in [35]).

Precedences and Temporal Links

Precedences between activities modelled in the solver, have to include information about the activities' validity. Each precedence $A \ll B$ between a pair of activities $A, B \in \mathcal{A}$ is modelled:

$$end_A \cdot val_A \cdot val_B \leq start_B$$

Minimum and maximum distance of a temporal link between two activities are modelled similarly to precedence constraints. The minimum distance constraint between a pair of activities $A, B \in \mathcal{A}$ is modelled:

$$(end_A + minDist_{AB}) \cdot val_A \cdot val_B \leq start_B$$

Similarly, maximum distance is modelled:

$$(start_B - maxDist_{AB}) \cdot val_A \cdot val_B \leq end_A$$

3.1.3 PEX

So far, we've only described properties of PEX temporal networks. Work of Beck and Fox [12, 13] is much more complex. It involves not only description of special kind of temporal network, but also an extension to constraint satisfaction technology. The extension consists of a new kind of variable, constraints for the variable and special kind of propagation.

In this section, only PEX related parts of the model are discussed. There are other constraints used in the model, those constraints are being explained in the context of nested temporal networks with alternatives in section 3.1.4.

PEX variables

The special variable proposed by Beck and Fox is called PEX variable, it was developed to represent that an activity may not exist in the final solution. The *probability of existence* (PEX) of an activity is the estimated probability at a search state that an activity will exist in a solution. The PEX value of an activity is a real value in the range $[0, 1]$. The PEX value 1 indicates that the activity will certainly exist and value 0 indicates that it will certainly not.

PEX variable is not a proper domain variable. To be able to maintain consistent probability of existence of an activity during the search, the PEX variable, until it is instantiated, can be repeatedly assigned any real value from the interval $(0, 1)$ in any search step. The variable can only be instantiated to values 1 or 0. Once a PEX variable is instantiated in some search step S, its value cannot be changed in any search step below S. The domain of the PEX variable cannot be emptied.

Since PEX variable isn't a true domain variable, it usually isn't implemented in constraint solvers. Choco, which was used for empirical evaluation, doesn't

implement it too. An extension to Choco solver was implemented as part of the thesis program to provide means to test and evaluate strategies utilizing PEX variables. The extension implement PEX variables and other PEX related changes to the constraint solver.

PEX variables are used only in search strategies proposed by Beck and Fox [12]. In that case the model contains a PEX variable for each activity.

- pex_A - probability of existence (PEX) of activity A

Besides PEX variables, the validity variables are still used in the model, they are necessary in some constraitns.

PEX Constraints

PEX temporal network uses special type of nodes - AndNodes and XorNodes. However for testing purposes, the custom extension of PEX (see 2.2.3) is used instead. All nodes in the network represent activities.

Parallel branching in the network can be modelled using equality constraints. Formally, let A be an activity, $outBranch_A = PAR$, then pex_A is connected with all $pex_B \in \mathcal{B}_A^{out}$ by these constraints:

$$pex_A = pex_B \; \forall B \in \mathcal{B}_\mathcal{A}$$

Similarly, let A be an activity $inBranch_A = PAR$, then pex_A is connected with all $pex_B \in \mathcal{B}_A^{in}$ by the same constraints.

Alternative branching requires different constraint. Let A be an activity and $outBranch_A = ALT$, then the constraint between pex_A and all $pex_B \in \mathcal{B}_A^{out}$ is:

$$xorNode(pex_A, \{pex_B | B \in \mathcal{B}_A^{out}\})$$

The same constraint is used to model the relationship between pex_A and $pex_B \in \mathcal{B}_A^{in}$ in the opposite branching ($inBranch_A = ALT$).

Moreover, PEX variables are bound to validity variables. A PEX value is propagated to the validity variable only when the PEX variable is instantiated (the PEX value is 0 or 1).

$$pex_A = val_A \; \forall A \in \mathcal{A}$$

3.1.4 Nested Temporal Networks with Alternatives

In the empirical evaluation, one type of scheduling problems targetted are those represented with nested temporal networks. Therefore, we're discussing modelling of n-TNA (and PEX temporal networks as subclass of n-TNA). The constraint are utilized for n-TNA models and models of PEX temporal networks as well.

Branching

Branching constraints for n-TNA are very similar to those used for PEX variables in PEX temporal networks.

Let A be an activity $outBranch_A = PAR$, then the constraint describing parallel branching in the fan-out subgraph is:

$$val_A = val_B \; \forall B \in \mathcal{B}_A^{out}$$

Similarly, if A is an activity with $inBranch_A = PAR$, then the constraint describing parallel branching in the fan-in subgraph is:

$$val_A = val_B \; \forall B \in \mathcal{B}_A^{in}$$

Now, let activity A be a principal principal of a fan-out subgraph with alternative branching ($outBranch_A = ALT$), the following constraint is then used to model the situation:

$$val_A = \sum_{B \in \mathcal{B}_A^{out}} val_B$$

Alternative branching in a fan-in subgraph with A ($inBranch_A = ALT$) as a principal activity is modelled similarly.

Improving Propagation over Alternative Branching

To improve propagation over alternative branching, pairs of corresponding principal activities in alternative branching are bound together with constraints. Let activity A be the out-branching activity such that $outBranch_A = ALT$ and B the in-branching activity such that $inBranch_B = ALT$, then their validity variables are constrainted:

$$val_A = val_B$$

and an explicit precedence constraint is added:

$$end_A \cdot val_A \cdot val_B \leq start_B$$

Unfortunately, not all principal nodes having alternative out/in-branching can be addressed by the constraints. Only pairs of corresponding nodes can be addressed. For example in figure 3.1 it can be applied only to bind together nodes A and D. Nodes C and D doesn't have any corresponding counterpart. Due to characteristics of PEX temporal networks, all alternative branching there have corresponding counterparts.

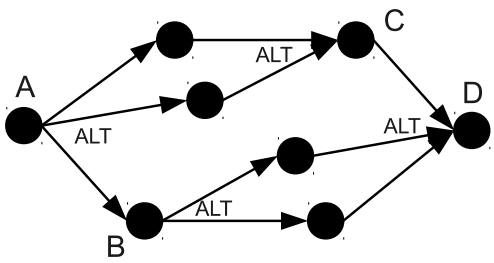


Figure 3.1: n-TNA example

3.1.5 x-RCPSP Constraints

Extended resource-constraint project scheduling problems define special activity relations: *activity substitutions* and *activity dependencies*. The relations can be modelled using standard solver constraints.

Activity Substitutions

Alternatives in x-RCPSPS are modelled by activity substitutions. We're targetting only cases where the activity substitutions are symmetric (if $x_{A,B} \in \mathcal{X}^+$ then also $x_{B,A} \in \mathcal{X}^+$). For each activity A_0 having k alternatives $A_1, \ldots, A_k$ (formally $x_{A_i,A_j}, x_{A_j,A_i} \in \mathcal{X}^+, 0 \leq i, j \leq k, i \neq j$), the following constraint is then added to model the situation:

$$1 = \sum_{i=0}^{k} val_{A_i}$$

Activity Dependencies

To model activity dependencies constraints on valilidy variables has to be added.

Let activity B to be activated upon activation of activity A, formally $m^{\oplus}_{A,B} \in \mathcal{M}^+$, then the constraint is:

$$val_A \leq val_B$$

Let activity B to be deactivated upon deactivation of activity A, formally $m^{\ominus}_{A,B} \in \mathcal{M}^+$, then the following constraint is used:

$$val_A \geq val_B$$

Let activity B to be activated upon deactivation of activity A, formally $m^{\urcorner}_{A,B} \in \mathcal{M}^+$, the constraint is:

$$val_A + val_B \geq 1$$

And let activity B to be deactivated upon activation of activity A, formally $m^{\llcorner}_{A,B} \in \mathcal{M}^+$, the constraint is:

$$val_A + val_B \leq 1$$

3.1.6 Optimality Criterion

In all the tests, one cost function is used - *makespan*. Choco solver natively supports makespan as its cost function, but the built-in implementation of makespan doesn't reflect validity of the variables. Therefore custom model for the makespan is used. The makespan is modelled with respect to optional activities with the following constraint:

$$makespan = max_{A \in \mathcal{A}}\{end_A \cdot val_A\}$$

3.2 Temporal Filtering in Temporal Networks with Alternatives

As part of the stategy tests temporal filtering as described in [8] is implemented to evaluate its effectivity. Choco solver doesn't implement the filtering by default. In Choco solver filtering is bound to constraints, therefore new constraints were implemented to provide the temporal filtering functionality.

Now, a short description of the filtering algorithm is provided. In the following text, let $d(x)$ denote domain of variable x. The model assumes that $d(x)$ is an interval for any variable x that is taking part in the temporal filtering, thus the interval arithmetic can be used, formally $\langle l, u\rangle + \langle a, b\rangle = \langle l + a, u + b\rangle$ and $\langle l, u\rangle - \langle a, b\rangle = \langle l - b, u - a\rangle$.

3.2.1 Parallel Branching

Let activities A and B be linked together in parallel branching, let the temporal link be described by $minDist_{AB}$ and $maxDist_{AB}$. Then we define

$$UP = d(start_B) \cap (d(end_A) + \langle minDist_{AB}, maxDist_{AB}\rangle).$$

Whenever $d(end_A)$ changes, the following filtering rule is applied:

$$\begin{aligned} d(start_B) &\leftarrow UP && \text{if } UP \neq \emptyset \\ d(val_B) &\leftarrow d(val_B) \cap \{0\} && \text{if } UP = \emptyset \end{aligned}$$

The temporal filtering works similarly in the opposite direction. Let

$$DOWN = d(end_A) \cap (d(start_B) - \langle minDist_{AB}, maxDist_{AB}\rangle)$$

Whenever $d(start_A)$ changes, the following filtering rule is applied:

$$\begin{aligned} d(end_A) &\leftarrow DOWN && \text{if } DOWN \neq \emptyset \\ d(val_A) &\leftarrow d(val_A) \cap \{0\} && \text{if } DOWN = \emptyset \end{aligned}$$

3.2.2 Alternative Branching

Let activity A be a principal node of fan-in subgraph.Let

$$UP = d(start_A) \cap \cup_{B \in \mathcal{B}_A^{in}} \{(d(end_B) + \langle minDist_{AB}, maxDist_{AB}\rangle) | d(val_B) \neq \{0\}\}$$

Now, whenever $d(end_B)$ or $d(val_B)$ of any $B \in \mathcal{B}_A^{in}$ is changed, the following filtering rule is applied:

$$\begin{aligned} d(start_A) &\leftarrow UP && \text{if } UP \neq \emptyset \\ d(val_A) &\leftarrow d(val_A) \cap \{0\} && \text{if } UP = \emptyset \end{aligned}$$

The temporal filtering in the opposite direction is slightly different. Let

$$DOWN_B = d(end_B) \cap (d(start_A) - \langle minDist_{AB}, maxDist_{AB}\rangle)$$

for activity $B \in \mathcal{B}_A^{in}$. Whenever $d(end_B)$ is changed, the following rule is applied:

$$\begin{aligned} d(end_B) &\leftarrow DOWN_B && \text{if } DOWN_B \neq \emptyset \\ d(val_B) &\leftarrow d(val_B) \cap \{0\} && \text{if } DOWN_B = \emptyset \end{aligned}$$

3.3 Precedence Constraint Posting in Choco Solver

There's one last part of the model, that has to be explained, even though it jumps ahead. It's the implementation of *precedence constraint posting* (PCP) in Choco.

In section 1.2.1 complete search algorithm was mentioned in general. Backtracking search assigning values to variables was mentioned as the basis. Nevertheless, a different approach to backtracking search exists, in a search step, instead of assigning values to variables, new constraints are posted to the model. Upon backtracking the constraints are revoked from the model. Such approach to backtracking is called constriant posting.

Constraint posting is not implemented by all constraint solvers, usually it's being simulated. The simulation is done by adding reified constraints and reification variables to the model during modelling. The constraints are then permanent, they are present in the model from the very beginning and they are not being revoked. The search algorithm then assigns values to the reification variables instead and acts the standard way. Precedence constraint posting (PCP) is constraint posting of precedence constraints.

Choco solver offers constraint posting directly, however authors of the solver advise to use reification instead mainly because of the performance.

3.3.1 Modelling PCP with Alternatives

PCP is used by some strategies to post precedences between activities requiring the same resource. Reified precedences and reification variables are added for each unordered pair of activities requiring the same resource.

To enable optional activities in PCP the model has to be more complex. For each pair of activities a pair of boolean precedence decision variables is created:

- $prec_{AB} \in \{0, 1\}$
- $prec_{BA} \in \{0, 1\}$

The decision variables are used by the search strategy to "post" the constraint. The variables are bound together such that only one of them can be true:

$$prec_{AB} + prec_{BA} = 1$$

The precedence decision variables are not the real reification variables. There is another pair of boolean variables - the reification variables:

- $precImpl_{AB} \in \{0, 1\}$
- $precImpl_{BA} \in \{0, 1\}$

These variables are bound to the precedence decision variables:

$$prec_{AB} \cdot val_A \cdot val_B = precImpl_{AB}$$

and

$$prec_{BA} \cdot val_A \cdot val_B = precImpl_{BA}$$

The binding is done to handle properly all the cases when some of the activities are invalid.

Precedences are then modelled using *precedenceImplied* built-in constraint in Choco solver. The *precedenceImplied* constraint is defined:

$$precedenceImplied(task_A, task_B, 1) \Rightarrow A \ll B$$
$$precedenceImplied(task_A, task_B, 0) \Rightarrow true$$

The *precedenceImplied* constraints are then used to model the reified precedences as follows:

$$precedenceImplied(task_A, task_B, precImpl_{AB})$$
$$precedenceImplied(task_B, task_A, precImpl_{BA})$$

It might happen that $prec_{AB}$ and $prec_{BA}$ variables are not instantiated at the end of the search - such situation happens when at least one of the activities A or B is invalid. Nevertheless, it is not a problem, the $prec_{AB}$ and $prec_{BA}$ are used only for decision making and only non-invalid activities are considered in implementation of the search strategies.

4. Search Strategies

In section 1.2.1 complete search algorithms were described. This chapter is devoted to description of backtracking based search strategies for scheduling problems in the framework of constraint satisfaction. First standard strategies for scheduling problems without alternatives are presented and then strategies for scheduling problems with alternatives.

A search strategy typically consists of three parts: variable selection (see 1.2.3), branching scheme (see 1.2.4) and value selection (see 1.2.3). In the following section the strategies are described as conjunction of these parts and optionally some other aspects.

4.1 Classical Constraint Satisfaction Search Strategies

The term "classical" search strategy means a standard strategy for any CSP that doesn't use any scheduling specific information. The problem is modeled using the mentioned constraints and variables and then the solver uses its built-in solving procedures to find the values for the modeled variables. The strategy doesn't distinguish kinds of variables (e.g. start variable, validity variable), they are all treated as being all the same. The strategy simply selects variables and values for the variables according to general search heuristics.

The classical search strategy basically consists of two parts: the *variable selector* and the *value selector*. Examples of standard variable selectors can be found in table 4.1. All the variable selectors, except Random variable selector, follow the *fail-first* principle (see section 1.2.3).

Variable Selector	Description
Random	A heuristic selecting randomly a non-instantiated variable.
Minimal domain	A heuristic selecting the variable with the smallest domain.
Most constrained variable	A heuristic selecting the variable with the maximum degree.
Domain over degree	A heuristic selecting the variable with the smallest ration $\frac{domainsize}{degree}$, the degree is the number of constraints linked to it.
Domain over dynamic degree	A heuristic selecting the variable with smallest degree, the degree is the number of constraints linked to it that are not completely instantiated.

Table 4.1: Examples of standard variable selectors in constraint solvers.

Examples of basic value selectors can be found in table 4.2.

Value Selector	Description
Random	A heuristic selecting randomly a value in the domain.
Min value	A heuristic selecting the lowest value in the domain.
Mid value	A heuristic selecting the middle value in the domain.
Max value	A heuristic selecting the highest value in the domain.

Table 4.2: Examples of basic value selectors in constraint solvers.

4.2 Basic Strategies for Scheduling Problems

In the previous section, we presented classic search strategies that do not utilize any information specific to scheduling problem. The strategies are capable to solve the problems, but their performance can be very slow. Therefore search strategies that utilize information about the scheduling problem were developed. These strategies, presented in this section, perform usually much better in solving scheduling problems.

4.2.1 Set Start Time

The *set start times* (SST) scheduling strategy is based on search through all possible start times of the activities. The strategy selects an activity and then decides its start time.

If we refer to the previous section 4.1 and variable and value selectors. The variable selector decides which activity is processed and the value selector decides the value of the start time of the activity. Typical branching scheme here is enumeration or binary choice.

There are many ways how to choose an activity, the start time of which, is to be set in the current step of the search strategy. Some notation has to be added now. For an activity A the earliest start time is denoted by est_A, the latest start time by lst_A and simirarly the earliest completion time is denoted by ect_A, the latest completion time by lct_A. Then the selected activity is normally determined according to these values. Typically, an activity A with the minimal est_A is chosen, and then the minimal value from the domain of $start_A$. Problem of this strategy is that wrong decisions are often recognized very late in the search. Therefore, early wrong decission still causes much of the search tree to be explored, and lots of backtracking steps are performed.

Despite these problems set start times strategy is frequently used. Lots of local search strategies are based on this strategy. The scenario is following: first an initial solution is computed, second the local strategy search incrementally improves the solution. The initial solution is usually computed with a SST strategy. Then these local search strategies usually work in two steps: relaxation and subsequent incremental recomputing. The recomputing step can be also based on a SST strategy. Examples of this approach are [**?**], and also iterative flattening [29, 30].

The work of Godard, Laborie and Nuijten [**?**] uses SST to find an initial solution and also for the recomputing step. They use minimal *est* for activity selection, ties are broken with minimal *lct*, and then the eariest start time is chosen as the value for the *start* variable.

The work of Oddi, Cesta, Policella and Smith [29, 30] assume the initial solution given. The initial solution can be for example computed with some greedy search (they optimize a MCJSSP with makespan as the optimality criterion). Then, again, the local search consists of two steps: relaxation step and flattening step. The flattening step can be based on SST or PCP (see 4.2.2). The flattening step based on SST selects the activity with the minimal *lct* (ties are broken by *est*).

In the following section some search streategies for scheduling problems with

alternatives are presented, some of them only decides about the validity the activities and order of activities on the resources they require. Then a variant of SST to assign the start times to the activities is used. In this case, the SST strategy is very fast, it assigns values to the start times of the activities that are already ordered, the value is assigned according to *est*.

4.2.2 Precedence Constraint Posting

A short introduction to PCP was mentioned in section 3.3, now the strategy is presented with more detail.

The *precedence constraint posting* (PCP) search strategy consists of two stages. The first stage decides for each resource the order of all activities requiring the resource. Because the first stage does not assign the start variables values (and the constraint propagation can rarely prune all other values) there is a necessity for another scheduling step. Therefore the second stage consists of a time allocation strategy, typically some variant of SST. But in this case, the search space can be significantly pruned due to propagation of precedence constraints. That's why the two stage PCP search strategy can perform much better that the SST alone.

The PCP stage of this strategy adds one or more precedence constraints between the activities in every node of the search tree. As mentioned in section 3.3 reified constraints are being used instaed of adding constraints during the search. The specific PCP strategy is also dependent on the branching strategy, which can be miscellaneous.

Binary Ordering of Activities

Many works implement some version of PCP. For example Smith and Cheng in [34] proposed a PCP strategy based on *slack*. Their strategy is designed for solving job shop scheduling problems. Job shop scheduling problems (JSSP) are defined on unary resources, thus cumulative resources are not considered in the current section. The (temporal) slack for ordering two activities $A \ll B$ is defined

$$slack(A \ll B) = lct_B - est_A - (p_A + p_B).$$

(Temporal) slack of two activities A and B is defined

$$slack(A, B) = \max\{slack(A \ll B), slack(B \ll A)\}.$$

And overal minimum (temporal) slack is defined

$$\min\{slack(A, B) | A, B \in \mathcal{A},\ A, B \text{ unordered}\}.$$

In more detail, their strategy distinguish four dominance conditions:

1. If $lct_A - est_B < p_A + p_B \leq lct_B - est_A$ then $A \ll B$.
2. If $lct_B - est_A < p_A + p_B \leq lct_A - est_B$ then $B \ll A$.
3. If $p_A + p_B > lct_B - est_A$ and $p_A + p_B > lct_A - est_B$ then there is no feasible schedule.

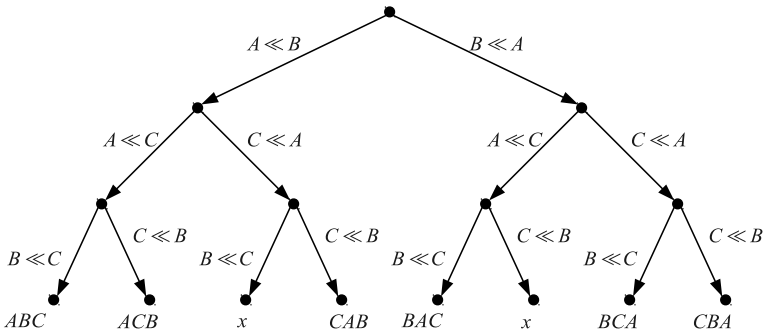


Figure 4.1: An example of a search tree for binary ordering of activities.

4. If $p_A + p_B \leq lct_B - est_A$ and $p_A + p_B \leq lct_A - est_B$ then either ordering is possible.

Given a particular resource, a search step of their strategy first decides whether any from the first three conditions can be used. If so, the rule is applied, and newly posted constraints are propagated. Otherwise the fourth rule is used. The fourth rule directly lead to binary branching. And in this fourth condition the slack is used in variable and value ordering to distinguish the best branch. For variable ordering they prefer to use overall minimum temporal slack (according to fail-first principle). And for value ordering, they choose the precedence with maximal slack (according to suceed-first principle).

Today this strategy is used in a different manner, only the fourth condition is applied. The first three conditions are part of constraint propagation during the search. The branching scheme they use is disjunctive constraint $(A \ll B) \vee (B \ll A)$. An example of the search tree of this strategy can be found in figure 4.1. There are three activities to be scheduled on a resource and only the fourth condition of their strategy is applied. Branching then leads to two inconsistent activity orderings (they are marked with x).

First Activity Selection

Another version of PCP was proposed by Baptiste, Le Pape and Nuijten in [1]. Instead of choosing a pair of activities and posting the precedence posting between them, they choose first (or last) activity among the set of unordered activities. Their strategy is also fitted to JSSP. Their algorithm first chooses a resource that is required by some unordered activities. Then selects an activity A that will be ordered first or last among all unordered activities requiring the particular resource (lets denote Ω the activities requiring the resource). Then the precedence constraints $A \ll B, \quad \forall B \in \Omega, \quad A \neq B$ are posted and propagated. In case of inconsistency other activities (activities from $\Omega \setminus \{A\}$) are tried as the first activity. The selection of activity runs until all the activities requiring the particular resource are ordered. Then the strategy runs untill all activities on all resources are ordered.

The work also proposed variable selections for both stages of this algorithms - the resource selection and the activity selection. The resource with the minimal resource slack is recommened (according to the fail-first principle). The resource

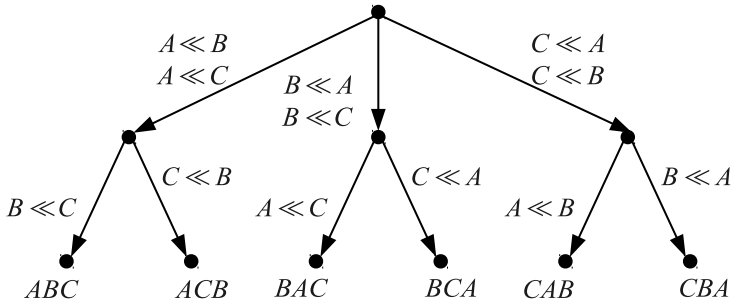


Figure 4.2: An example of a search tree for the first activity selection.

slack is defined as

$$\max\{\max_{A\in\Omega} lct_A - \min_{A\in\Omega} est_A - \sum_{A\in\Omega} p_A\}.$$

The activity selection consists of two steps. First, decide whether the activity will be ordered first or last, and better is to choose the variant with the smaller number of possibilities (the fail-first principle). Second, if the activity is being ordered to be first, select the activity according to smallest *est*, ties are broken by *lst*. A symmetric rule is used when ordering the activity as last. An example of this strategy can be found in figure 4.2.

The branching scheme they use is fairly complex. Let $\Omega = \{A_1, \ldots, A_n\}$, the branching scheme then dissolves disjunctions $(A_1 \ll (\Omega \setminus \{A_1\})) \vee (A_2 \ll (\Omega \setminus \{A_2\})) \vee \ldots \vee (A_n \ll (\Omega \setminus \{A_n\}))$ or $((\Omega \setminus \{A_1\}) \ll A_1) \vee ((\Omega \setminus \{A_2\}) \ll A_2) \vee \ldots \vee ((\Omega \setminus \{A_n\}) \ll A_n)$ in each search tree node.

4.3 Traditional Approach to Scheduling with Alternatives

Search strategies for scheduling problems with alternatives have to decide not only about time allocation of the activities but also about validity of the activities. Traditional approach to handle alternatives is, first, to decide which alternative to use, then to schedule the chosen activities on chosen resources. This approach is often referred to as "plan first, schedule next". The strategy could be following: first select a subset of activities satisfying the validity constraints, and second to schedule a standard scheduling problem that doesn't have any alternatives. This is classical strategy how to solve alternatives.

Modern approaches try to incorporate both steps to gain better performance. The activity selection and scheduling are strongly connected, integrating of these two stages can produce much better schedules. For example an activity can be swapped for its alternative when the domain of its start time is emptied. In traditional approach it would be recognized as a dead-end, the search would backtrack to the first stage, where a brand new set of valid activities would be selected.

Because of qualities of the approaches connecting both stages, many ways to work with alternatives more effectively than traditional "plan first, schedule next" approach were introduced.

4.4 PEX

The works about PEX [12, 13] were already mentioned in context of modelling scheduling problems with alternatives. Now, the search related proposition in their works are presented. Their works propose some adjustments to the search heuristics to incorporate PEX - *probability of existance* - as defined in 3.1.3.

4.4.1 PEX Propagation

It was already mentioned that the PEX variables aren't real CSP domain variables. All PEX values have to be either 0 or 1 in a solution, but they represent the current estimated probability of existence during the search. Propagation of PEX values in a temporal network is a vital part of the work of Beck and Fox. Initial PEX propagation begins with unassigned PEX values and consistently assings values to PEX variables. If an activity has to be present in the schedule (has either no upstream or downstream links) then its PEX value is set to 1. If an activity A has assigned PEX value pex_A then all activities B represented by non-XorNode nodes (and all activities C represented by XorNodes that has A as the only upstream/downstream neighbour) adjacent to A must have the same PEX value - $pex_A = pex_B$. If an activity A represented by XorNode has PEX value $pex_A = x$, there is k upstream and l downstream links, the PEX value of each directly connected upstream activity is set to x/k and the PEX value of each directly connected downstream activity is set to x/l.

Next, the incremental PEX propagation starts with a consistent network - each activity has a PEX value. Then a PEX commitment is introduced and PEX value of an activity is changed to 0 or 1. The new value is propagated into the network upstream and downstream from the selected activity. The PEX values of other activities are changed during the propagation, but only the activities with PEX value not equal to 0 or 1 are affected. PEX propagation runs in *cascades* - each cascade is limited by the surounding pair of XorNodes (one XorNode upstream and one XorNode downstream). Next cascade starts at the XorNodes where the previous cascade ended, it proceeds further upstream from the upstream XorNode and downstream from the downstream XorNodes. The propagation in a cascade runs first from the source node (or the source pair of XorNodes) upstream and downstream to the pair of surrounding XorNodes, and then from the surrounding XorNodes back into the nested network affecting only the remaining branches.

In section 2.2.1 it was mentioned that in PEX temporal networks each XorNode must have a corresponding XorNode. The reason is the inconvenient ambiguity for expected PEX values - in illegal PEX temporal network such as the one in figure 2.2 a value propagated upstream could be different to value propagated downstream.

Temporal propagation with PEX is different from normal propagation, the difference is caused by XorNodes. A XorNode must not start before at least one of its upstream neighbours ends, and similarly with the downstream neighbours. Also if it happens that a domain of an activity is emptied during the temporal propagation and there still exists a corresponding PEX value that is less than 1, it may not be a dead-end, it may only imply a PEX commitment - a particular PEX variable must be set to 0. If the node with the emptied domain has a PEX value

less than 1, PEX commitment is derived, otherwise a dead-end is recognized. Temporal propagation is performed after each PEX propagation cascade.

PEX Propagation in PEXTNA

In section 2.2.3 a PEX extension was defined. The extension uses alternative and parallel branching instead of XorNodes and AndNodes. PEX propagation in the extension remains the same. The initial PEX propagation begins with unassigned PEX values and consistently assigns values to PEX variables. An activity A with no upstream or downstream connections must be present in the schedule, thus $pex_A = 1$. If an activity A has PEX value pex_A then all activities B such that either $B \in \mathcal{B}_A^{out}, inBranch_B \neq ALT$ or $B \in \mathcal{B}_A^{in}, outBranch_B \neq ALT$, and all activities C such that either $\mathcal{B}_C^{out} = \{A\}, outBranch_C = ALT$ or $\mathcal{B}_C^{in} = \{A\}, inBranch_C = ALT$ are assigne the same value - $pex_B = pex_A$ and $pex_C = pex_A$. If an activity A is a principal node of alternative branching of a fan-out subgraph $outBranch_A = ALT$ and has PEX value $pex_A = x$, then all activities $B \in \mathcal{B}_C^{out}$ are assigned PEX value $pex_B = \frac{x}{|\mathcal{B}_C^{out}|}$. A symmetric rules is used if an activity A is a principal node of alternative branching of a fan-in subgraph $inBranch_A = ALT$.

The incremental PEX propagation runs in cascades as well. Each cascade is imited by surrounding pair of activities with alternative branching - an activity A upstream such that $outBranch_A = ALT$ and an activity B downstream such that $inBranch_B = ALT$.

Temporal propagation works the same, an activity A such that $inBranch_A = ALT$ must not start before at least one activity $B \in \mathcal{B}_A^{in}$ ends, similar rule holds if $outBranch_A = ALT$. Temporal propagation in PEX temporal network can be effectively implemented using temporal filtering algorithm as defined in 3.2.

4.4.2 PEX and Heuristics

The works about PEX [12, 13] propose some adjustments to the search heuristics to incorporate PEX. They extended three heuristics with PEX: SumHeight [11], CBASlack [34], LJRand [28]. SumHeightPEX directly use the value of PEX variable, the other two, CBASlackPEX and LJRandPEX, uses only three values: 0, 1 or neither-0-nor-1. Now the heuristics are described here, further details can be found in the original works [12, 13].

4.4.3 SumHeightPEX

To understand SumHeightPEX heuristic we first has to present the classical (non-PEX version) of the algorithm - *SumHeight* heuristic [11]. The SumHeight heuristic analyzes the constraint graph to identify critical points. Commitments are then made in these points to decrease the criticality.

First, some definitions are necessary. The *individual demand* $ID(A, R, t)$ is measured for activity A, resource R at time t:

$$ID(A, R, t) = \frac{\min(t, lst(A)) - \max(t - duration(A) + 1, est(A))}{|domain(start(A))|}.$$

First, the SumHeight heuristic identifies the most critical resource and time point.The most critical resource $\hat{R}$ and time point $\hat{t}$ is the one with the highest aggregated demand. The highest aggregated demand is computed by summing individual demands on each resource over all activities. After identifying $\hat{R}$ and $\hat{t}$, two activities A and B are identified - activities that contribute most to the individual demand on resource $\hat{R}$ at time $\hat{t}$ that are not already ordered. Then the consequences of each ordering of activities A and B ($A \ll B$ and $B \ll A$) are examined and the superior ordering is chosen.

SumHeightPEX heuristic incorporate PEX values into SumHeight heuristic. The individual demand is extended with PEX information:

$$ID_{PEX}(A, R, t) = PEX_A \cdot ID(A, R, t).$$

Again, the resource and time point with the highest aggregate demand are identified. Then, three activities are then found: an activity A with the highest individual demand and PEX value of $0 < pex_A < 1$, two activities B and C whose order is not already linkded by constraints, that have PEX values of 1 and have the highest individual demand on $\hat{R}$ at time $\hat{t}$. Let's assume $ID_{PEX}(B, \hat{R}, \hat{t}) >= ID_{PEX}(C, \hat{R}, \hat{t})$. If $ID_{PEX}(A, \hat{R}, \hat{t}) > ID_{PEX}(B, \hat{R}, \hat{t})$ then A is the most critical activity, the best commitment is then to set its PEX value to 0, upon backtracking its opposite (setting PEX value to 1) is used. If $ID_{PEX}(B, \hat{R}, \hat{t}) >= ID_{PEX}(A, \hat{R}, \hat{t})$ then activities B and C are sequenced, upon backtracking the reverse sequence is chosen.

Sequencing Critical Activities

The SumHeight (and SumHeightPex) heuristic uses complex value selector for sequencing the two most critical activities. It uses three value selectors: *Minimize-Max* sequencing heuristic, *Centroid* sequencing heuristic and *Random* sequencing heuristic. If MinimizeMax predicts that one sequence will be better, then it's selected. If not Centroid heuristic is tried. If Centroid heuristic predics that one sequence will be better, it is selected. Otherwise Random heuristic, which selects the sequencing randomly, is used.

MinimizeMax Sequencing Heuristic

Let $RD(R, t)$ denote the aggregated demand for resource R at time t. An estimate of the new aggregate demand at a single time point $AD'(A, A \ll B)$ is calculated. It is calculated on estimate of the new individual demand of activity A provided that precedence $A \ll B$ holds. It is calculated as follows:

1. Assuming that $A \ll B$, the new individual demand is calculated for activity A. The time point tp in the individual demand of activity A that has the maximum increase in height $\Delta height$ is identified.

2. Then the value $AD'(A, A \ll B)$ is defined as $RD(R, tp) + \Delta height$.

Next $AD'(B, A \ll B)$ is calculated similarly. Then maximum from those two values is calculated:

$$max_{AD'}(A, B) = max(AD'(A, A \ll B), AD'(B, A \ll B))$$

MinimizeMax heuristic then selects sequencing which satisfies:

$$MM = min(max_{AD'}(A, B), max_{AD'}(B, A))$$

If no sequencing commitment is better than the other, the Centroid heuristic is tried.

Centroid Sequencing Heuristic

Centroid sequencing heuristic is a variation of heuristic proposed in [27]. The centroid of the individual demand curve is the time point that equally divides the area under the curve. The centroid is calculated for each activity. The sequence that preserves the current ordering of centroids is chosen.

4.4.4 CBASlackPEX

The CBASlackPEX heuristic extends the CBASlack heuristic [34]. The CBASlack heuristic identifies the pair of activities on the same resource that have the *minimum biased slack*. Let's denote

$$S = \frac{\min\left(slack(A \ll B), slack(B \ll A)\right)}{slack(A, B)}$$

than *biased slack*

$$Bslack(A \ll B) = \frac{slack(A \ll B)}{f(S)}$$

where f is a monotonically increasing function, e.g. n-th root of S for $n \geq 2$. The *biased slack for a pair of activities* A, B is

$$Bslack(A, B) = \min\left(Bslack(A \ll B), Bslack(B \ll A)\right).$$

The activities are sequentced so as to preserve the most slack.

The CBASlackPEX heuristic calculate biased slack only for activities with PEX value greater than 0. The following three conditions then apply:

1. If both activities have a PEX value of 1, sequence them so as to have the maximum slack.
2. If activity A has a PEX value of 1 and activity B has PEX value less than 1, set the PEX value of B to 0.
3. Otherwise set the PEX value of the activity with the longest duration to 0.

CBASlackPEX Implementation

The implementation of CBASlackPEX finds the pair of activities according to the overall minimum biased slack across all the resources. If the problem instance contains a resource required by a single activity, then the CBASlackPEX heuristic isn't able to decide about PEX value of the activity. Therefore the CBASlackPEX heuristic is accompanied by a secondary heuristic, used only if there's no remaining pair of activities the CBASlackPEX heuristic can handle. The secondary heuristic selects activity A according to the longest duration (following the fail-first principle), the primary commitment is to assign $pex_A = 0$ (following the succeed-first principle), alternative commitment is to assigne $pex_A = 1$.

4.4.5 LJRandPEX

The LJRandPEX heuristic is an extension of the LJRand heuristic [28] that finds the smallest *ect* of all the unscheduled activities and identifies set Ω of uncheduled activities that can start before the smallest *ect*. Then an activity A is randomly selected from the set Ω and scheduled at est_A. When backtracking, the est_A is updated to the minimum *ect* of all other activities on that resource.

The PEX extension of LJRand finds the smallest *ect* of all unscheduled activities with PEX greater than 0. Then identifies set $\hat{\Omega}$ of uncheduled activities with PEX value greater than 0 that can start before the smallest *ect*. Then it randomly selects an activity A from the set $\hat{\Omega}$ and schedule A to est_A and assign $PEX_A = 1$. The alternative commitement, is to update the est_A to the minimum *ect* of all other activities with PEX value greater than 0 requiring the same resource, the alternative does not contain any PEX commitment.

LJRandPEX Implementation

Implementation of the LJRandPEX heuristic used in the tests is little different to the one proposed by Beck and Fox. First, it finds the smallest *ect* of all unscheduled activities with PEX greater than 0 requiring a resource, let C be the activity determining the *ect*. Then identifies set $\tilde{\Omega}$ of uncheduled activities with PEX value greater than 0 that can start before the smallest *ect* and that require any resource that C requires. Then it randomly selects an activity A from the set $\hat{\Omega}$ and schedule A to est_A and assign $PEX_A = 1$. The alternative commitement, is changed significantly. In the original version, it can happen that alternative commitment doesn't change the est_A at all. In that case, if we are search for the best solution, the search can end up in an infinite loop.

To prevent such situations, the alternative commitment is implemented in a way that always increases the est_A. The computation if following, let

$$E_1 = \{ect_B \mid B \in \mathcal{A},\ pex_B > 0,\ A, B \text{ share a resource}\},$$

$$E_2 = \{ect_B + minDist_{(A,B)} \mid B \in \mathcal{A},\ pex_B = 1,\ \exists \text{ temporal link between } A, B\}$$

and

$$E = E_1 \cup E_2$$

then the new *est* for activity A is computed:

$$est_A = max\{est_A + 1, min\{x > est_A | x \in E\}\}$$

No change to pex_A is done in alternative commitment.

4.5 Two Level Branching Strategy for Alternatives

The work of Bartak [6] describes a branching strategy to efficiently solve constraint model proposed in [7]. Although the strategy is explained in context of temporal networks with alternatives, the strategy can be clearly used in other scheduling problems with alternatives, such as x-RCPSP (see 2.2.4), too.

First few definitions are requiered. An activity is called *invalid* if the validity variable is set to 0. An activity is *valid* if its validity variable is set to 1. An activity is *non-invalid* if it is valid or no value has been assigned to the validity variable so far.

The branching strategy consists of two stages. The first stage selects a non-invalid activity A with unknown position and decides whether it is valid or invalid. The second branching stage is available only for the valid activities. The second stage decides about the position of the activity on the resource. Let's denote $\mathcal{X}_A$ the set of all non-invalid activities requiring the same resource. Then for each $B \in \mathcal{X}_A$ disjunction $(A \ll B) \vee (B \ll A \wedge val_B = 1)$ is resolved in the second stage (propagation of previous decisions can remove some activities from $\mathcal{XA}$ during the search). An example showing the strategy can be found in figure 4.3. The figure shows a part of search tree for a non-invalid activity A, $\mathcal{X}_A = \{B, C, D\}$, the leafs denote the partial orders of the activites, bold letters denote valid activities, other activities are still non-invalid, order of the activities in brackets haven't been decided yet.

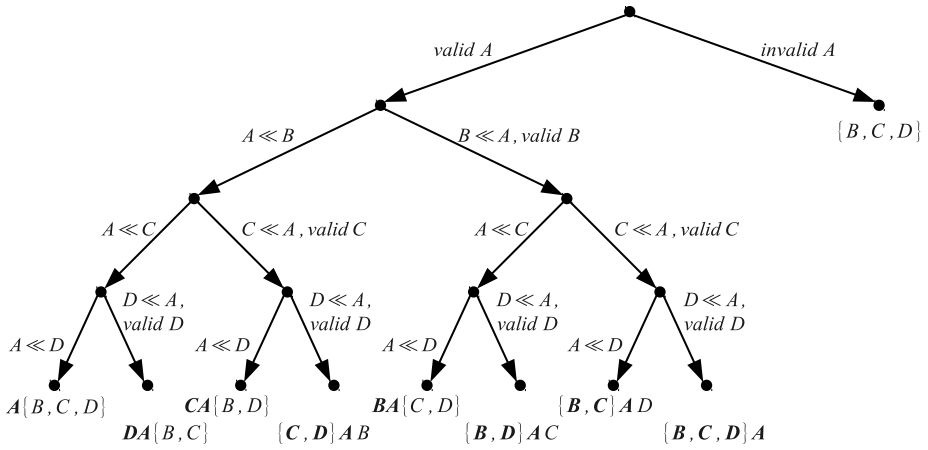


Figure 4.3: An example of search tree of OptActTwoLevel branching strategy for a non-invalid activity A.

If it is decided that activity A is invalid then the branching is repeated for all the remaining non-invalid unallocated activities until all the branches are explored. All leafs in complete search tree contain only valid and allocated activities. An example of complete search tree is in figure 4.4, the search explores all posibilities for three non-invalid not-yet allocated activities A, B, C.

It is obvious from the figure that the strategy explores all possible orders for these three variables and no order is explored twice. These features of the search strategy are proven in [6].

The strategy as shown in [6] selects the non-invalid activity which should be scheduled in the first stage according to *est*, ties are broken with the smallest duration and the smallest *lct*.

4.5.1 OptActTwoLevel Strategy

In the test program an adjusted version of the strategy was implemented. It works almost the same as described above. Still, there's one slight difference.

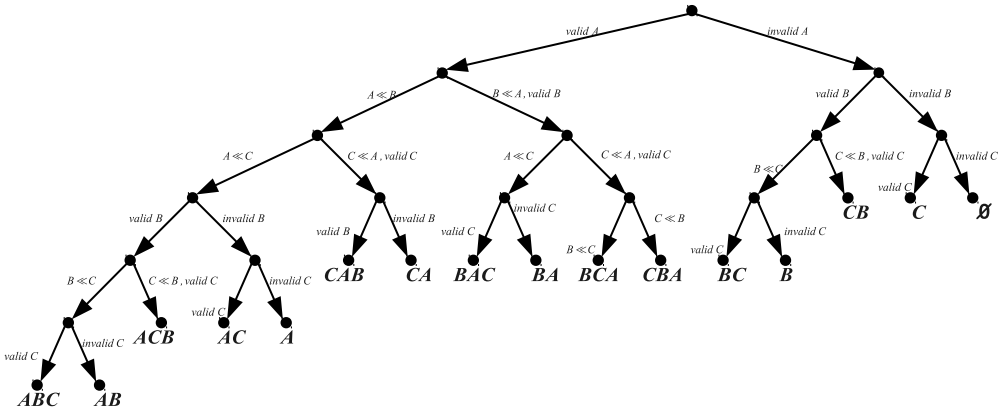


Figure 4.4: An example of complete search tree of OptActTwoLevel branching strategy.

Let's denote $\mathcal{Y}_A$ the set of all non-invalid activities requiring any resource that activity A requires (it's assumed that an activity A can require any number q_A of resources). Then the *OptActTwoLevel* strategy (the name is further used in evalution in section 5 and graphes in appendix B) resolves in the second stage for each $B \in \mathcal{Y}_A$ disjunction $(A \ll B) \vee (B \ll A \wedge val_B = 1)$. In case an activity doesn't require any resource, only the first stage of the branching strategy is performed.

4.6 Proposed Search Strategies for Alternatives

One of the goals of the thesis is to try to develop, customize or enhance a strategy for a scheduling problem. Since, the thesis is focused on scheduling problem with alternatives, some customized search strategies for the scheduling problems are presented in further sections.

4.6.1 OptActTwoLevelPair Strategy

A strategy inspired by the OptActTwoLevel strategy is presented here, let it be called *OptActTwoLevelPair* strategy.

One of the problems of the branching strategies described in section 4.5 is the imbalace of the search trees. In this case, some solution could require significantly more search decisions to find a solution, an early wrong decision can cause lot of backtracking to find a solution. The figure 4.4 shows such imbalanced tree generated while scheduling three activities. If there is more activities being scheduled on a resource the tree becomes even more imbalanced. The problem of the imbalanced search tree lead to the proposal of a new search strategy for scheduling problems with alternatives that would produce a search tree with a better shape, thus fewer search decision would be necessary. Also it was desirable that the new strategy would cover the same types of problems as the strategy described in section 4.5. The strategy is suitable for scheduling any presented scheduling problem with alternatives - either described by a temporal network with alternatives or an instance of x-RCPSP.

The strategy basically decides about the validity and ordering of pairs of activities that share a resource, it distinguishes valid and invalid activities, and also valid and invalid pairs of activities. Let's denote $pairval_{AB}$ the validity of a pair of activities A and B. The pair of activities is valid if both activities are valid, $val_A = val_B = 1 = pairval_{AB}$. A pair of activities is invalid if at least one of the activities is invalid, $pairval_{AB} = 0 \Leftrightarrow (val_A = 0 \text{or} val_B = 0)$. A pair of activities is non-invalid if it is valid or no value has been assigned to the pair validity variable so far.

The branching strategy consists, similarly to OptActTwoLevel strategy, of two stages. The first stage selects a pair of non-invalid unsequenced activities that share a resource and decided about validity and sequencing of the pair of activities. The second branching stage is available only for pairs of activities that are decided to be invalid in the first branching stage. The second branching stage decided about the validity of the activities in the pair.

In the first stage the pair of acitivties that have the largest sum of durations (of the activities in the pair) is selected according to the fail-first principle. Let A and B be the activities in the currently selected pair of activities. In the first starge the strategy selects one of these commitments:

1. Set $pairval_{AB} = 0$.
2. Set $pairval_{AB} = 1$ and $A \ll B$.
3. Set $pairval_{AB} = 1$ and $B \ll A$.

The value are selected accoring to the succeed-first principle. The first commitment is to set validity of the pair to 0. From the alternative commitments, the sequencing is selected according to the simplified centroid heuristic. The simplified centroid for activity A is computed as $(lct_A + est_A)/2$, first the ordering that preserves the current ordering is selected. In case one of the sequencing commitment is selected the constraint propagation immediately deduce that $val_A = val_B = 1$. In the next search step, the strategy chooses a new pair of non-invalid unordered activities and the previous process is repeated.

In case the first commitment was selected in the first stage, the second stage decides about one of these commitments:

1. Set $val_A = val_B = 0$.
2. Set $val_A = 1$ and $val_B = 0$.
3. Set $val_A = 0$ and $val_B = 1$.

Again, the order of the commitments follows the succeed-first principle. The first commitment is to set both activities invalid. From the alternative commitments, the activity with longer duration is selected first. Let A be the activity with longer duration and B be the other activity. Then val_A is set to 0 and val_B is set to 1. The last alternative is to set $val_A = 1$ and $val_B = 0$.

An example of a part of search tree of the strategy is shown in figure 4.5, the leafs denote partial orders of the activities, bold letters denote activities whose validity hasn't been determined yet [1], order of the activities in brackets haven't been decided yet.

[1] Although it's not clear from the picture, the leftmost leaf is annotated with bold C

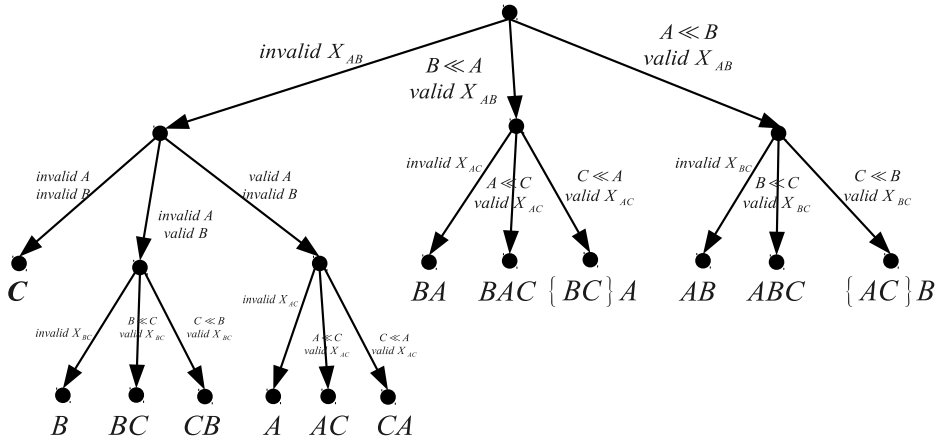


Figure 4.5: An example of a search tree of OptActTwoLevelPair search strategy.

In case that, there's no non-invalid unorder pair of activities left on any resource, the strategy selects an activity from all remaining non-invalid activities and decides about its validity. The longest activity is selected first, the first commimtent is to set its validity to 0, the alternative commitment is to set its validity to 1.

Obviously, some of the branches of the search tree might be pruned during the search due to constraint propagation. These branches might be inconsistent. Typical example is a choice of two valid activities. The first commitment in the first level of the strategy is immeditely pruned because the validity variables are set to 1 for both activities and the pair validity is also 1.

4.6.2 CBASlackNoPEX Strategy

CBASlackPex strategy (4.4.4) utilizes three values of the PEX variable: 0, 1 or neither-0-nor-1. Such situation can be also reflected by validity variable, the tricky neither-0-nor-1 value can be reflected by the validity variable as not being instantiated. Let's call the altered strategy *CBASlackNoPEX*.

The motivation behind altering CBASlackPEX into CBASlackNoPEX, is the chance to omit PEX propagation. And more importantly, chance to target more types of scheduling problems with alternatives. Without PEX variables, the strategy can target more general temporal networks with alternatives n-TNA and TNA, and it can be also used for solving x-RCPSP.

Without PEX propagation and its cascading propagation algorithm, there's a possibility that the algorithm would run faster.

4.6.3 LJRandNoPEX Strategy

Similarly to CBASlackPEX, LJRandPEX utilizes only three values of the PEX variable: 0, 1 or neither-0-nor-1. Thus, it can be altered into *LJRandPEX* strategy, that uses activities' validity variables instead of PEX variables. The motivation behind LJRandPEX is the same as behind CBASlackNoPEX.

The implementation of LJRandNoPEX strategy has the same adjustments as LJRandPEX strategy.

5. Empirical Evaluation

To experimentally evaluate the strengths and weaknesses of different search strategies a benchmark framework was created as part of the thesis. It is a program created in Java programming language using Choco solver [37] as the underlying constraint technology. Choco solver in the version 2.1.1. was used. Further description of the benchmark framework can be found in documentation to the program in appendix A. The program is contained on the enclosed compact disc D.

5.1 Experiments

In chapter 4 several search strategies for scheduling problems with alternatives were presented. The main goal of the thesis is to evaluate the performance of the strategies on various data with various configurations. Since all the strategies are developed for scheduling problems containing only unary resources, no cumulative resources were present in the test data.

Test data used in the evaluation were of three types: instances of nested temporal networks with alternatives (see 2.2.2), instances of extension to PEX temporal networks (see 2.2.3) and instances of extended resource-constrained project scheduling problems (see 2.2.4). The tests are split according to the data types.

5.1.1 Experiments on n-TNA Data

The performance of search strategies on n-TNA data were tested in two configurations of the strategies: with the temporal filtering and without the temporal filtering. Four strategies were tested on the n-TNA data:

- OptAltTwoLevel
- OptAltTwoLevelPair
- CBASlackNoPEX
- LJRandNoPEX

Several data sets according to two parameters were created. The parameters and the values used are summarized in table 5.1. To sum it up, it's 15 different data sets configurations. For each configuration 16 problem instances are generated.

The created data contain unary resources only, each activity can require any non-zero number of resources. Temporal links with minimum and maximum distance are also created. Release dates of activities are specified. Problem instances can be defined by one or more temporal networks.

Parameter	Values
Number of activities (resources)	20 (10), 60 (20), 100 (30)
Probability of alternative branching	0%, 25%, 50%, 75%, 100%

Table 5.1: Configurations of n-TNA nad PEXTNA data sets.

5.1.2 Experiments on PEXTNA Data

Similarly to n-TNA data, the performance of search strategies on PEXTNA data were tested in two configurations of the strategies: with the temporal filtering and without the temporal filtering. Seven strategies were tested on the PEXTNA data:

- SumHeightPEX
- CBASlackPEX
- LJRandPEX
- OptAltTwoLevel
- OptAltTwoLevelPair
- CBASlackNoPEX
- LJRandNoPEX

Several data sets according to two parameters were created. The parameters and the values used are summarized in table 5.1. The number of different data sets configurations is the same as for n-TNA data, it's 15 different configurations. For each configuration 16 problem instances were generated.

The data configuration is similar to n-TNA data. The created data contain unary resources only, each activity can require any non-zero number of resources. Temporal links with minimum and maximum distance are also created. Release dates of activities are specified. Problem instances can be defined by one or more temporal networks.

5.1.3 Experiments on x-RCPSP Data

On x-RCPSP data the strategies were tested in one configurations - without the temporal filtering, it can't be used on x-RCPSP data. There were four strategies tested on the x-RCPSP data:

- OptAltTwoLevel
- OptAltTwoLevelPair
- CBASlackNoPEX
- LJRandNoPEX

The data used in x-RCPSP were from [36]. Originally, the data were designed for testing x-RCPSP for modeling and solving Disruption Management Problems. The resources in the data are cumulative. Since, only unary resources are being targetted by the implemented search strategies, the data were adjusted for out needs. Changing dat from cumulative to unary involved changing resource capacities to 1, changing resource requirements to 1 and prolonging due date of the instances. The data doesn't contain any temporal links (only precedences, acitivity substitutions and activity dependencies). Due to the original purpose of the data - disruption management - the data contain some information that are being ignored, they are useless for our needs.

The data is available in two sets - set of small instances and set of large instances. Each set further contain 16 groups of problem instances described by 4 binary configuration switches, each group containing 10 instances. From the 4 original configuration switches, only 2 are relevant in our case - *process complexity* and *resource complexity*. The values are of the switches are low and high in both cases. Other configuration switches are ignored. As results each set of the two sets contains 4 distinct group of problem instances, each group containing 40 instances. The parameters for x-RCPSP data and values for the parameters are summarized in table 5.2.

Parameter	Values
Data set	i1 (small intances), i2 (large instances)
Process complexity	0 (low), 1 (high)
Resource complexity	0 (low), 1 (high)

Table 5.2: Configurations of x-RCPSP data sets.

5.1.4 Test Environment Configuration

Because of the infavourable complexity of the tested problems, the solver must have been limited during scheduling. Although Choco solver offers number of available limits to be applied on the search procedure, namely backtrack, node, fail, restart and time limits, only one of them can't be used at a time. Therefore time limit was chosen to limit the search. Maximum time available for solving a problem instance was set to 20 s.

All the tests were performed on personal computer, configuration of the computer is summarized in table 5.3.

Parameter	Value
OS	Ubuntu Linux 10.04
CPU	Intel(R) Core^{TM}2 Duo T5870 2.00 GHz
Memory	3 GB DDR2 SDRAM, 800 MHz

Table 5.3: Test computer configuration

5.1.5 Performance Measurement

In all the tests the optimal schedule is being searched for. One cost function is used in all the test cases - *makespan*. When a cost function is present, the Choco solver uses Branch and Bound method (see 1.2.5).

There are several statistics measured for each test instance during the search:

- Number of backtracks performed
- Number of nodes performed
- Number of failures (contradictions) performed
- Running time of the search algorithm
- Overall time spent scheduling the instance (modelling, and I/O operations included)

- Information whether a feasible schedule was found (any schedule was found), the instance is infeasible or the feasibility is unknown

For each strategy and for each test configuration, all feasible results are selected, and then averages for all the statistics (except feasibility information) are calculated. Results for n-TNA and PEXTNA are averages from up to 16 instances. Results for x-RCPSP are averages from up to 40 instances.

Two main statistics are measured (and their graphs are plotted) - number of backtracks and running time of the search algorithm. The remaining statistics (except feasibility information) should correlate with these statistics. Still, the computed data and plotted graphs are available on the enclosed compact disc. Plotted graphs of average number of backtracks and average running time are available in appendix B.

5.2 Tested Hypothesis

There are several expected results, the experiments should confirm. First of all, it's generally expected that strategies will performe worse on larger instances.

A next result should be the confirmation of the importance of temporal filtering defined in section 3.2. When there are many optional activities in the problem instance, the filtering is supposed to significantly prune domains of the possible start times, there should be clear outcome in performance boost. On n-TNA and PEXTNA data it should hold that the higher the probability of alternative branching is, the higher the increase in speed of the search should be. Also if the probability of alternative branching is equal to 0, it's possible that the version with temporal filtering could be slower because the cost of computation of the temporal filtering could outweight the benefits of derived temporal information.

The LJRandPEX and LJRandNoPEX strategies are expected to perform worse than other strategies. Because of the way they are randomized and the commitments are made, they are not suitable in cases where we are searching for an optimum. The number of backtracks performed should be higher compared with other strategies. Also the running time should be higher, and there's a high probability that a significant portion of the test instances would time out. On the other hand, due to the high randomization of the strategies, they might be able to find a solution (not optimal) for very complex problem instances, the ones for which other strategies might fail to find any solution.

It's not clear if the PEX versions of LJRandPEX and CBASlackPEX strategies would be better than those not using PEX variables. On one side, PEX propagation requires more computation, on the other side, it migh offer a better information about the problem structure.

According to [12] the SumHeightPEX and CBASlackPEX strategies should perform much better than LJRandPEX.

The OptActTwoLevelPair strategy compared with the CBASlackPEX (CBASlackNoPEX) would probably perform slightly worse because it uses more constraint variables thus the computation should be more expensive.

OptActTwoLevel strategy would probably perform worse than CBASlack or OptActTwoLevelPair, it doesn't always follow the succeed-first principle. And as

mentioned in the motivation for OptActTwoLevelPair strategy in section 4.6.1, it's early bad decisions could be more expensive.

5.3 Results

Only the most important result are presented in the section. All the test results for all the strategies, test data and configurations can be found summarized in tables in appendix C. All the plotted graphs of number of backtracks performed and time used for scheduling, can be found in appendix B.

All the tests confirmed that temporal filtering significantly improves the performance of the strategies when there are alternative activities present in the problem instance. It's clearly visible in figure 5.1 containing the comparison of strategies on PEXTNA data with 50% probability of alternative branching.

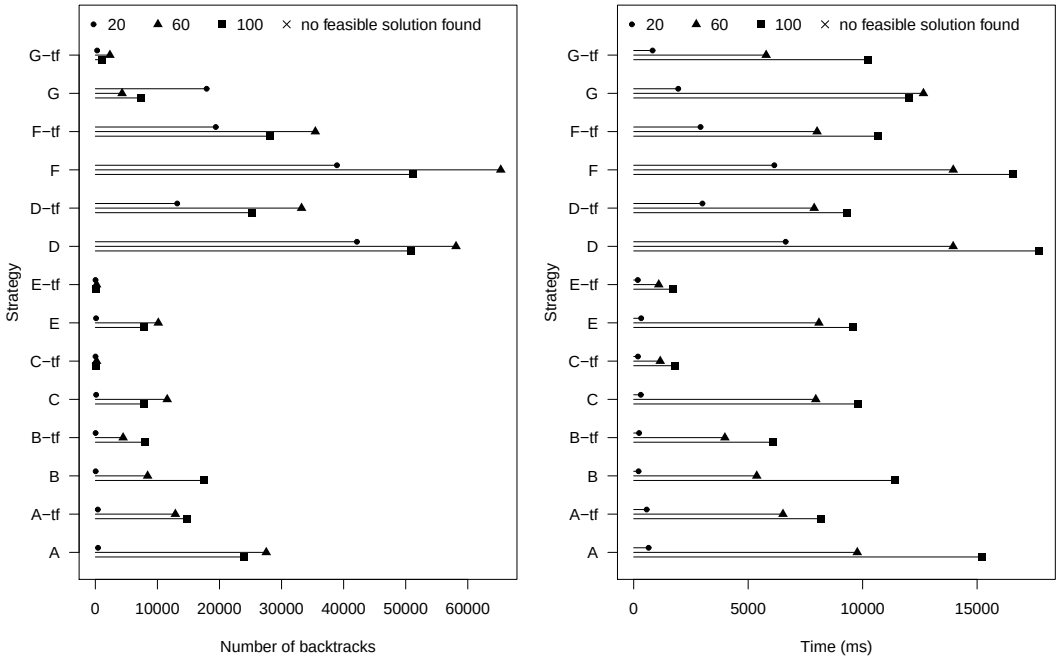


Figure 5.1: Strategy comparison on PEXTNA data with probability of alternative 50%. The strategies are: OptActTwoLevel (A), OptActTwoLevelPair (B), CBASlackNoPEX (C), CBASlackPEX (E), LJRandNoPEX (D), LJRandPEX (F), SumHeightPEX (G). Suffix "-tf" denotes usage of temporal filtering.

Practically all the data confirmed that both LJRandPEX and LJRandNoPEX performed much worse than all other strategies. For example it can be seen in figure 5.1.

The CBASlackNoPEX and the CBASlackPEX seems to the fastest with the least number of backtrack. Also OptActTwoLevelPair seems to be perform well, see for example figure 5.2.

5.3.1 Results for PEXTNA Data

Experiments performed on PEXTNA data showed, that there isn't any significant difference in performance of the two variants of CBASlackPEX and LJRandPEX

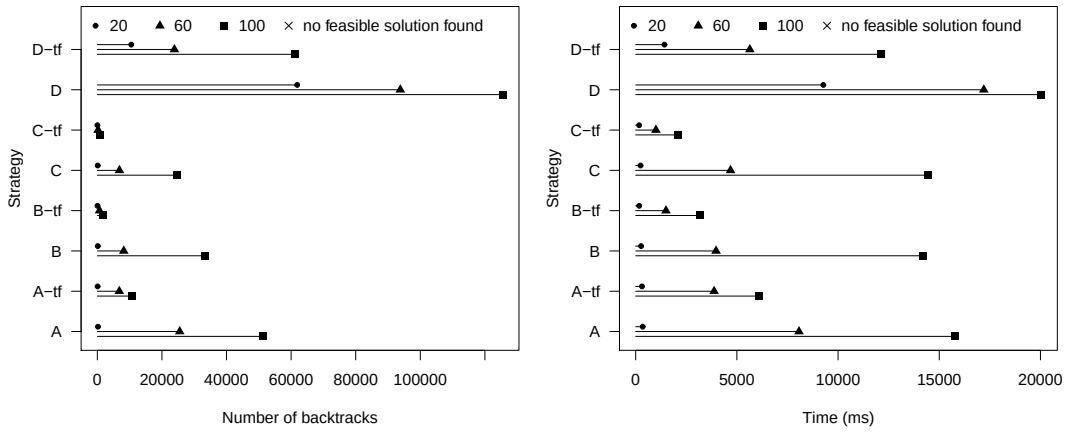


Figure 5.2: Strategy comparison on n-TNA data with probability of alternative 100%. The strategies are: OptActTwoLevel (A), OptActTwoLevelPair (B), CBASlackNoPEX (C), LJRandNoPEX (D). Suffix "-tf" denotes usage of temporal filtering.

heuristic strategies. Both the CBASlackPEX and the CBASlackNoPEX performed similarly. So did LJRandPEX and LJRandNoPEX.

Now if we look at the comparision of the three strategies studied in [12] - SumHeightPEX, CBASlackPEX and LJRandPEX. LJRandPEX is clearly the worst performing strategy. Meanwhile, the CBASlackPEX seems to be the fastest strategy. Although, SumHeighPEX makes in some cases very few backtracks (see figure 5.1).

5.3.2 Results for x-RCPSP Data

Experiments with x-RCPSP showed that there is a significant difference in complexity of test set $i1$ and test set $i2$. While 20 s time limit was sufficient for all strategies to optimize almost all the problem instances in i1, it was very low for i2 data set. Most of the strategies had problem to find any solution within the time limit for most of the problem instances.

Still, we can conclude that LJRandNoPEX peform as expected - much worse than other strategies on standard problem instances. But if we look at the tables in section C.3, it's clear that the strategy found solutions for the biggest number of the problem instances in the $i2$ data set. On most of the test data configurations, the OptActTwoLevelPair strategy performed better than OptActTwoLevel strategy.

CBASlackNoPEX seems to be performing better than all other strategies, and OptActTwoLevelPair strategy seem to perform slightly better than OptActTwoLevel strategy.

Other results for x-RCPSP test data aren't much conclusive.

Conclusion

In the thesis several approaches to the modelling of the scheduling problems with alternative activities as constraint satisfaction problems were presented. Several search strategies for scheduling such problems were described, implemented and compared.

A new strategy, the OptActTwoLevelPair strategy, was proposed and tested. The experiments show that it is one of the faster strategies.

CBASlackPEX and LJRandPEX strategies were enhanced such that they can be used without the complex PEX propagation, and therefore they can target other types of scheduling problems with alternatives. The experiments shows that there isn't any loss in performance.

Temporal filtering algorithm for temporal network with alternatives was tested and a recognized to bring a significant improvement in performance.

A special type of constraint variable, the PEX variable, and its complex propagation was implemented into a constraint solver and tested. However, the results of the strategies utilizing the extra information didn't show any significant benefit of it.

During the implementation of the PEX propagation Choco solver was studied as a sideeffect. Unfortunatelly, Choco solver seems to suffer from some bad architectural decisions and therefore some additions might be difficult to implement without changing or reimplementing big amount of its code.

Future Work

In future work there's a space for further improvements. Probably the most important future work would be the implementation of more clever search strategies, filtering algorithms and constraints with better propagation. For example Beck and Fox proposed rather strong, but computationally expensive, filtering algorithm - *PEX-edge-finding* (for details see [12, 13]).

Reimplementation of the search streategies in different constraint solvers would be convenient to confirm or disprove the obtained results.

Furthermore, there are another experiments the studied strategies can be compared at, for example the search for a first solution might be an intersting result. Some strategies might find a solution quickly, the found solution might be far from optimum. While other strategies might find a solution slower, but much closer to the optimum.

A future improvements might be possible in the implementation of some search strategies in Choco solver. The strategies don't always work ideally when dealing with optional activities. For example built-in *SetTimes* search strategy tries to assign start times for all activities, even for the invalid ones. Fortunatelly, it's not a problem and it doesn't affect the found schedule, but it's an unnecessary computation. Also there's always a possibility that the implementation of the studied search strategies might be suboptimal.

Experiments on the x-RCPSP data weren't much conclusive due to the high complexity of the problem instances in the i2 data set. In a future work experiments with better x-RCPSP data should be performed.

At last experiments with larger data sets should be performed to obtain more precise results. Also larger time limit could be provided to larger instances.

Bibliography

[1] Baptiste, P., Le Pape, C., Nuijten, W. *Constraint-Based Optimization and Approximation for Job-Shop Scheduling*. Proceedings of the AAAI-SIGMAN Workshop on Intelligent Manufacturing Systems, IJCAI-95, 1995.

[2] Baptiste, P., Le Pape, C. *Edge-finding constraint propagation algorithms for disjunctive and cumulative scheduling*. Proceedings of the 15th Workshop of the U.K. Planning Special Interest Group (PLANSIG), 1996.

[3] Baptiste, P., Le Pape, C., Nuijten, W. *Constraint-based Scheduling: Applying Constraints to Scheduling Problems*. International Series in Operations Research and Management Science. Dordrecht: Kluwer Academic Publishers, 2001, vol. 39.

[4] Barták, R. *Constraint Satisfaction for Planning and Scheduling*. In Ionannis Vlahavas, Dimitris Vrakas (eds.): Intelligent Techniques for Planning. Idea Group, 2005, pp. 320-353.

[5] Barták, R. *Constraint programming: In pursuit of the holy grail*. In Proceedings of WDS99, 1999, pp. 555-564.

[6] Barták, R. *Search Strategies for Scheduling Problems with Optional Activities*. In Search in Artificial Intelligence and Robotics: Papers from the 2007 AAAI Workshop, Technical Report WS-08-10. AAAI Press, 2009.

[7] Barták, R., Čepek, O. *Temporal Networks with Alternatives: Complexity and Model*. Proceedings of the 20th International Florida AI Research Society Conference (FLAIRS). AAAI Press, 2007, pp. 641-646.

[8] Barták, R., Čepek, O. *Nested Temporal Networks with Alternatives*. Papers from the 2007 AAAI Workshop on Spatial and Temporal Reasoning, Technical Report WS-07-12. AAAI Press, 2007, pp. 1-8.

[9] Barták, R., Čepek, O., Hejna, M. *Temporal Reasoning in Nested Temporal Networks with Alternatives*. Recent Advances in Constraints, LNAI 5129. Springer-Verlag, 2008.

[10] Barták, R., Salido, M. A., Rossi, F. *Constraint satisfaction techniques in planning and scheduling*. Journal of Intelligent Manufacturing. Springer Verlag, 2010, vol. 21, no. 1, pp. 5-15.

[11] Beck, J. C., et al. *Texture-based heuristics for scheduling revisited*. In Proceedings of AAAI-97. Menlo Park, California: AAAI Press, 1997.

[12] Beck, J. C., Fox, M. S. *Scheduling alternative activities*. Proceedings of the 16th national conference on Artificial intelligence and the 11th Innovative applications of artificial intelligence conference innovative applications of artificial intelligence, AAAI '99/IAAI '99, 1999, pp. 680-687.

[13] Beck, J. C., Fox, M. S. *Constraint-directed techniques for scheduling alternative activities*. Artificial Intelligence, 2000, pp. 211-250, vol. 121 (1-2).

[14] Blazewicz, J., Lenstra, J. K., Rinnooy Kan, A. H. G. *Scheduling projects to resource constraints: Classification and complexity.* Discrete Applied Mathematics, 5, 1983, pp. 11-24.

[15] Brucker, P. *Scheduling Algorithms.* Third Edition. Springer Verlag, 2001.

[16] Brucker, P., Drexl, A., Möhring, R., et al. *Resource-constrained project scheduling: Notation, classification, models and methods.* European Journal of Operational Research, 1999, vol. 112, pp. 3-41.

[17] Caseau, Y., Laburthe, F. *Disjunctive scheduling with task intervals.* LIENS Technical Report 95-25. Laboratoire d'Informatique de I'Ecole Normale Superieure, 1995.

[18] Conway, Richard Walter, Maxwell, William L., Miller, Louis W. *Theory of Scheduling.* Reading, MA: Addison-Wesley, 1967. ISBN 0-486-42817-6.

[19] Frost, D., Dechter, R. *Dead-end driven learning.* In Proceedings with MIPS-XXL. In 5th International Planning Competition Booklet (IPC-2006). Lake District, England, 1994, pp. 28-30.

[20] Gaschnig, J. *A general backtrack algorithm that eliminates most redundant tests.* In Proceedings of IJCAI. Cambridge, MA, USA, 1977, pp. 457.

[21] Gaschnig, J. *Performance measurement and analysis of certain search algorithms.* Technical Report CMU-CS-79-124. Carnegie-Mellon University, 1979.

[22] Graham, R. L., Lawler, E. L., Lenstra, J.K., et al. *Optimization and approximation in deterministic sequencing and scheduling: A survey.* Annals of Discrete Mathematics. 1979, vol. 5, pp. 287-326.

[23] Haralick, R., Elliot, G. *Increasing tree search efficiency for constraint satisfaction problems.* Artificial Intelligence, 1980, vol. 14, pp. 263-314.

[24] Hartmann, S. *Project scheduling with multiple modes: A genetic algorithm.* Annals of Operations Research, 2001, vol. 102, pp. 111-135.

[25] Kuster, J., Jannach, D., Friedrich, G. *Handling Alternative Activities in Resource-Constrained Project Scheduling Problems.* IJCAI. 2007, pp. 1960-1965.

[26] Kuster, J., Jannach, D., Friedrich, G. *Extending the RCPSP for modeling and solving disruption management problems.* Applied Intelligence. Springer, 2008.

[27] Muscettola, N. *Scheduling by Iterative Partition of Bottleneck Conflicts.* Technical Report CMU-RI-TR-92-05. The Robotics Institute, Carnegie Mellon University, 1992.

[28] Nuijten, W. P. M., et al. *Randomized constraint satisfaction for job shop scheduling.* In Proceedings of the IJCAIJ'93 Workshop on Knowledge-Based Production, Scheduling and Constrol. 1993, pp. 251-262.

[29] Oddi, A., Cesta, A., Policella, N., et al. *Iterative Flattening Search for Multi-Capacity Scheduling Problems*. In In Miguel A. Salido, Juan Fdez-Olivares (Eds.) Planning, Scheduling and Constraint Satisfaction. Universidad de Salamanca, 2007, pp. 10-21.

[30] Oddi, A., Cesta, A., Policella, N., et al. *Iterative Improvement Strategies for Multi-Capacity Scheduling Problems*. In: COPLAS-07. Proceedings of CP/ICAPS Joint Workshop on Constraint Satisfaction Techniques for Planning and Scheduling Problems. 2007.

[31] Prosser, P. *Hybrid algorithm for the constraint satisfaction problem*. Computational Intelligence. 1993, vol. 9, pp. 268-299.

[32] Rinnooy Kan, A.H.G. *Machine Scheduling Problems: Classifications, Complexity and Computations*. The Hague: Nijhoff, 1976. ISBN 9-024-71848-1.

[33] Smith, B. M. *Succeed-first or Fail-first: A Case Study in Variable and Value Ordering*. In G. Smolka (ed.), Proc. 3rd Intl. Conf. on Principles and Practice of Constraint Programming, Lecture Notes in Computer Science. Springer Verlag, 1997, vol. 1330, pp. 321-330.

[34] Smith, S., Cheng, C. *Slack-Based Heuristics for Constraint Satisfaction Scheduling*. Proceedings 11th National Conference on Artificial Intelligence. 1993, pp. 139-144.

[35] Vilím, P., Barták, R., Čepek, O. *Extension of O(n log n) filtering algorithms for the unary resource constraint to optional activities*. Constraints, 2005, vol. 10, no. 4, pp. 403-425.

[36] Kuster, Jürgen, Jannach, Dietmar, Friedrich, Gerhard, *Extending the RCPSP for Modeling and Solving Disruption Management Problems* [test data]. 2011 [cited 2011-11-25] Available from Internet: <http://rcpsp.serverside.at/applied-intelligence-07.html>.

[37] *Choco Solver* [computer program]. Ver. 2.1.1. 2011 [cited 2011-11-22]. Available from Internet: <http://www.emn.fr/z-info/choco-solver>.

[38] *The R Project for Statistical Computing* [computer program], Ver. 2.10.1. Available from Internet: <http://www.r-project.org>.

[39] *The Java Development Kit* [computer program], Ver. 6. 2011 [cited 2011-11-22]. Available from Internet: <http://www.oracle.com/technetwork/java/javase/downloads/index.html>.

[40] *The Java Runtime Environment* [computer program], Ver. 6. 2011 [cited 2011-11-22]. Available from Internet: <http://www.oracle.com/technetwork/java/javase/downloads/index.html>.

[41] *Apache Maven Project* [computer program], Ver. 2.2.1. 2011 [cited 2011-11-22]. Available from Internet: <http://maven.apache.org>.

List of Tables

4.1 Examples of standard variable selectors in constraint solvers. . . . 29
4.2 Examples of basic value selectors in constraint solvers. 29

5.1 Configurations of n-TNA nad PEXTNA data sets. 43
5.2 Configurations of x-RCPSP data sets. 45
5.3 Test computer configuration 45

A.1 List of program's command-line arguments. 57
A.2 Available values for the model type parameter. 58
A.3 Available values for the strategy parameter. 58

C.1 Test results for CBASlackNoPex strategy on n-TNA data. 75
C.2 Test results for CBASlackNoPex-TF strategy on n-TNA data. . . 75
C.3 Test results for LJRandNoPex strategy on n-TNA data. 76
C.4 Test results for LJRandNoPex-TF strategy on n-TNA data. . . . 76
C.5 Test results for OptActTwoLevel strategy on n-TNA data. 76
C.6 Test results for OptActTwoLevelPair strategy on n-TNA data. . . 77
C.7 Test results for OptActTwoLevelPair-TF strategy on n-TNA data. 77
C.8 Test results for OptActTwoLevel-TF strategy on n-TNA data. . . 77
C.9 Test results for CBASlackNoPex strategy on PEX data. 78
C.10 Test results for CBASlackNoPex-TF strategy on PEX data. . . . 78
C.11 Test results for CBASlackPex strategy on PEX data. 78
C.12 Test results for CBASlackPex-TF strategy on PEX data. 79
C.13 Test results for LJRandNoPex strategy on PEX data. 79
C.14 Test results for LJRandNoPex-TF strategy on PEX data. 79
C.15 Test results for LJRandPex strategy on PEX data. 80
C.16 Test results for LJRandPex-TF strategy on PEX data. 80
C.17 Test results for OptActTwoLevel strategy on PEX data. 80
C.18 Test results for OptActTwoLevelPair strategy on PEX data. . . . 81
C.19 Test results for OptActTwoLevelPair-TF strategy on PEX data. . 81
C.20 Test results for OptActTwoLevel-TF strategy on PEX data. . . . 81
C.21 Test results for SumHeightPex strategy on PEX data. 82
C.22 Test results for SumHeightPex-TF strategy on PEX data. 82
C.23 Test results for CBASlackNoPex strategy on data. 82
C.24 Test results for LJRandNoPex strategy on data. 83
C.25 Test results for OptActTwoLevel strategy on data. 83
C.26 Test results for OptActTwoLevelPair strategy on data. 83

List of Abbreviations

- *CSP* – Constraint Satisfaction Problem
- *JSSP* — Job Shop Scheduling Problem.
- *MRCPSP* — Multi-Mode Resource-Constrained Project Scheduling Problem
- *n-TNA* — Nested Temporal Networks with Alternatives
- *PCP* — Precedence Constraint Posting
- *PEX* — Probability of Existence. Estimated probability of existence of an activity at a search state.
- *PEXTNA* — Extension of PEX temporal networks.
- *RCPSP* — Resource-Constrained Project Scheduling Problem
- *SST* — Set Start Times
- *TNA* — Temporal Networks with Alternatives
- *x-RCPSP* — Extended Resource-Constrained Project Scheduling Problem

Attachments

A. Test Program Documentation

To evaluate the search strategies a test program was created. The test program is implemented in Java programming language. Choco solver [37] is used as the underlining constraint technology. To implement all the tested strategies, it was necessary to adjust and extend Choco solver.

Besides the test program, several small programs and scripts were created to provide batch processing of the test data, processing of the results and plotting the graphs. Also, test data generator for n-TNA and PEXTNA data is provided. The *test framework* - the test program together with all the other mentioned parts - is available on the enclosed compact disc. Documentation of the test framework can be found in section A.2.

Documentation of the sole test program implementing the search strategies is located in section A.1.

Only a short documentation of the test program and test framework is presented, neither the program nor the framework are core of the thesis. The short documentation is provided to allow the reader to reproduce the results presented in the thesis.

A.1 Test Program Documentation

The test program implements the studied search strategies, it reads an instance from an input file and writes the schedule instance into an output file.

A.1.1 Test Program User Documentation

The executable jar file of the is located in directory `program` on the enclosed compact disc. To run the program, Java SE Runtime Environment [40] is necessary.

The test program doesn't have any graphical user interface, it's being run from the command line. The command to run the program is:

```
java -jar search-strategies.jar [OPTIONS]
```

The command line arguments used by the program are summarized in table A.1.

Parameter	Required	Description
`-i INPUT_FILE`	yes	Input file
`-o OUTPUT_FILE`	yes	Output file
`-t MODEL`	yes	Type of the used model. Available values are listed in table A.2
`-s STRATEGY`	yes	Name of the strategy. Available values are listed in table A.3
`-lb NUMBER`	no	Number of backtracks limiting the search.
`-lf NUMBER`	no	Number of failures limiting the search
`-ln NUMBER`	no	Number of nodes limiting the search
`-lt NUMBER`	no	Time limiting the search

Table A.1: List of program's command-line arguments.

Both input and output file are xml files. The input data can be found in the `inputdata` directory on the compact disc. Processed output data can be found in the `outputdata` directory. Choco solver can use only one parameter limiting the search. The priority of the limits is (from higher to lower): time, backtrack, fail, node. Only the limit with the highest priority is used. If no limit is specified, time limit is set to 20 s.

Value	Description
`unary-xrcpsp`	model for x-RCPSPS data
`unary-ntna`	model for n-TNA data not using temporal filtering
`unary-ntna-tf`	model for n-TNA data using temporal filtering
`unary-pex`	model for PEXTNA data not using temporal filtering
`unary-pex-tf`	model for PEXTNA data using temporal filtering

Table A.2: Available values for the model type parameter.

Value	Description
`OptActTwoLevel`	OptActTwoLevel strategy
`OptActTwoLevelPair`	OptActTwoLevelPair strategy
`CBASlackNoPex`	CBASlackNoPEX strategy
`CBASlackPex`	CBASlackPEX strategy
`LJRandNoPex`	LJRandNoPEX strategy
`LJRandPex`	LJRandPEX strategy
`SumHeightPex`	SumHeightPEX strategy

Table A.3: Available values for the strategy parameter.

Example usage

Several examples of running the test program are presented.

```
java -jar search-strategies.jar -t "unary-ntna" -s "OptActTwoLevelPair"
   -i "problem-instance.xml" -o "scheduled-instance.xml"

java -jar search-strategies.jar -t "unary-pex" -s "CBASlackPex"
   -i "problem-instance.xml" -o "scheduled-instance.xml" -lt 5000
```

A.2 Test Framework Documentation

The test framework is located in the `test-framework` directory on the enclosed compact disc. The framework is a Maven project contained in the `search-strategies-choco` subdirectory.

A.2.1 Project Structure

The `search-strategies-choco` Maven project consists of several other Maven subprojects that are located in subdirectories of the main project. The projects are:

- `choco` — Adjusted Choco solver.

- `search-strategies` — The test program described in section A.1, it's dependend on `choco` project.
- `test-data-generator` — Utility project containing program for generating temporal networks (n-TNA and PEXTNA) test data.
- `test-strategies` — Utility project containing scripts for batch testing.
- `test-analysis` — Utility project containing scripts and utility programs for extracting results from the output files, summarizing them and plotting the graphs.

The test framework located on the compact disc contains all the necessary compiled files to run the program and all the utilities. However, since the utility scripts are *Bash* scipts, there might be problems running them in some environments. Therefore Unix/Linux operating system is preferred to run the utility scripts.

To compile the framework Java Development Kit [39] in version 6 or higher and Apache Maven [41] in version 2.1.1 or higher are required. To compile the project, change (on command line) to the `search-strategies-choco` directory (where `pom.xml` for the whole framework is located) and run:

```
mvn package -Dmaven.test.skip=true
```

The argument `-Dmaven.test.skip=true` is present to skip the test phase (running the unit tests) of the build. It first, speeds up the build, and second, some tests in project `choco` fails and stops the compilation.

Besides source and compiled files, all the data files - input data files, output data files and processed output files - are present (so that the results can be checked) at the location defined by each of the projects.

Choco Project

Choco project in the `choco` directory contains source files of the Choco solver [37] at version 2.1.1. The Choco solver had to be adjusted to enable implementation of the PEX search strategies.

Search Strategies

The main project is the implementation of the search strategies in the `search-strategies` directory. Besides implementation of the strategies, the models, the input and output, it contains the PEX extension to the Choco solver.

Usage of the program is described in section A.1.

The project is dependent on the `choco` project, these two projects together contain the implementation of the strategies. All the remaining projects mentioned further are only utilities that ease the data processing and analysis.

Generating Test Data

The project in the `test-data-generator` directory is a utility program for generating n-TNA nad PEXTNA test data. To generate test data, it's easier to run `generate-dataset.sh` bash script that is located in the `script` subdirectory. The script generates test data in configurations that were used in experiments, the configurations are summarized in table 5.1. The data are created in `generated-test-data` subdirectory. The test data generator contains x-RCPSP data too, they are located in `static-test-data` subdirectory, these data are static.

Testing Strategies

For batch testing, some scripts were created in the `test-strategies` project. To run the experiments on the x-RCPSP, n-TNA and PEXTNA data, run `test-data.sh` script that's located in the `script` subdirectory. It would run all the experiments on the data that are located in test data generator in its subdirectories `generated-test-data` and `static-test-data`. The output files of the experiments are created in `test-output` subdirectory.

Running all the experiments requires many hours of processing time.

Processing the Test Output Data

In `test-analysis` project, in the `script` subdirectory several scripts for processing the scheduled data are placed. The `summarize-test-outputs.sh` script summarizes data for each data type and each strategy into one csv file. These csv files are created in subdirectory `analysis-output/summary`.

Then there's script `count-averages.sh` that would compute the statistics, that are present in the tables in appendix C, from the csv files. It creates in subdirectory `analysis-output/avg` csv files containing the calculated statistics.

To generate graphs, script `plot-graphs.sh` can be used. It uses statistical program R [38] to plot the graphs. The files containing the graphs are created in subdirectory `plot`. Most of the plotted graphs is present in appendix B, however there are graphs that aren't visualized in the thesis. For example, there graphs for other statistics - number of failures and number of nodes.

B. Graphs

B.1 Graphs of n-TNA Results

B.1.1 Performace of the Strategies

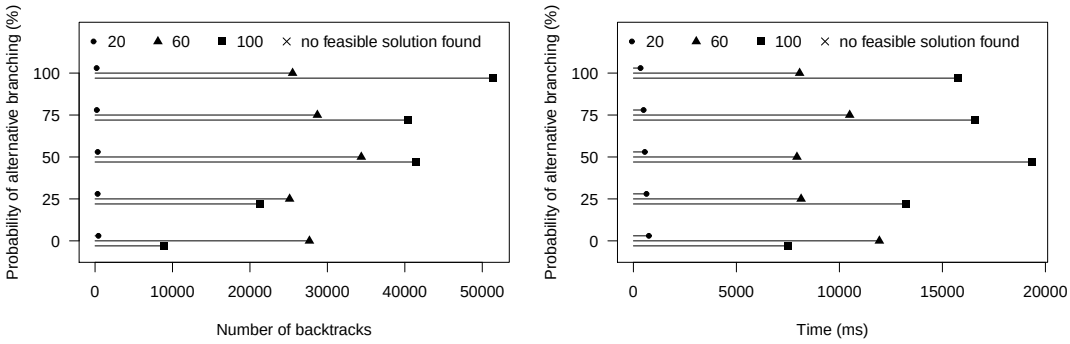


Figure B.1: Performance of OptActTwoLevel strategy.

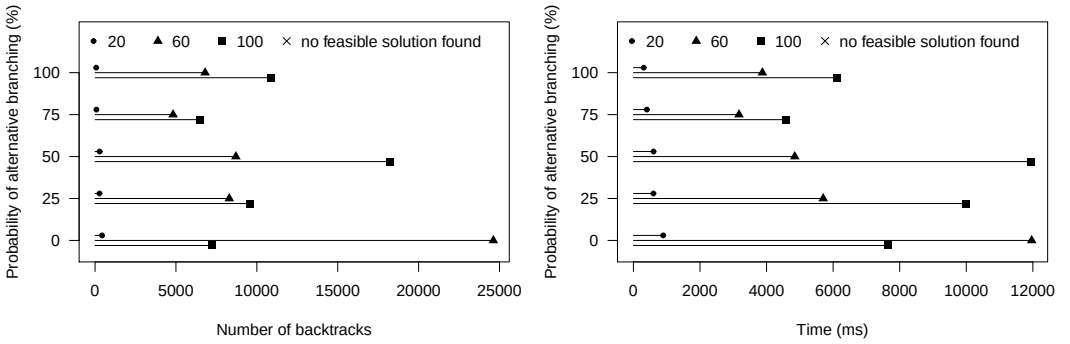


Figure B.2: Performance of OptActTwoLevel strategy with temporal filtering.

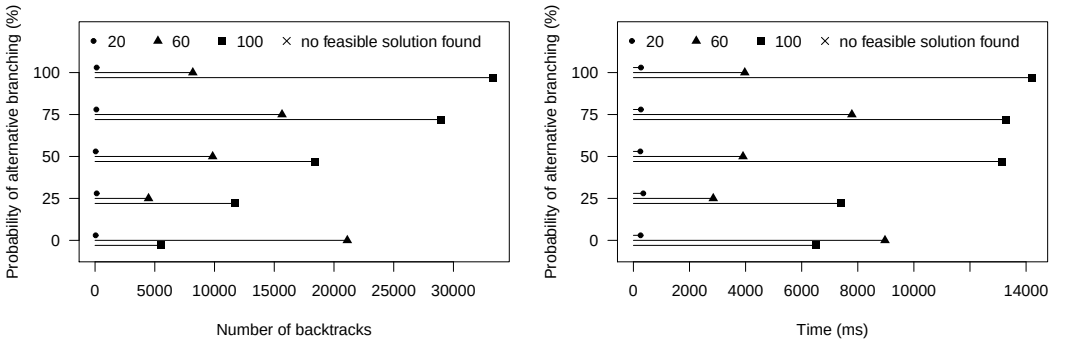


Figure B.3: Performance of OptActTwoLevelPair strategy.

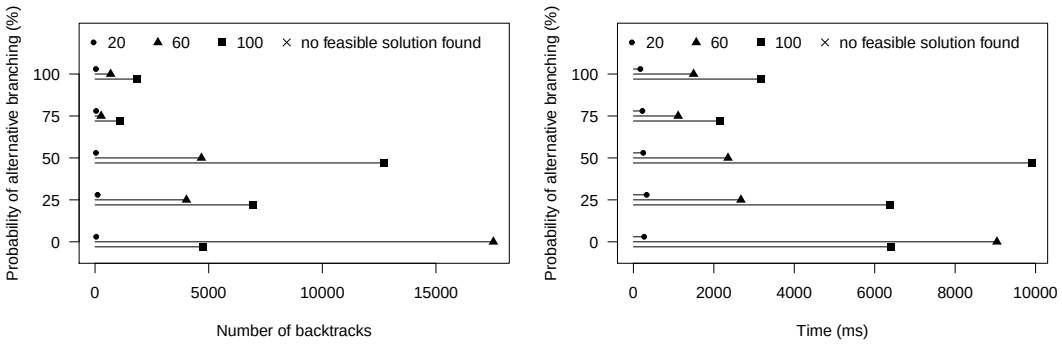


Figure B.4: Performance of OptActTwoLevelPair strategy with temporal filtering.

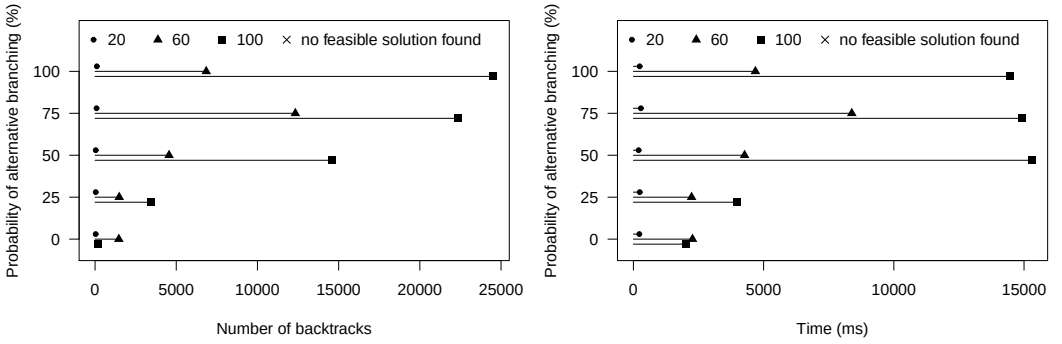


Figure B.5: Performance of CBASlackNoPEX strategy.

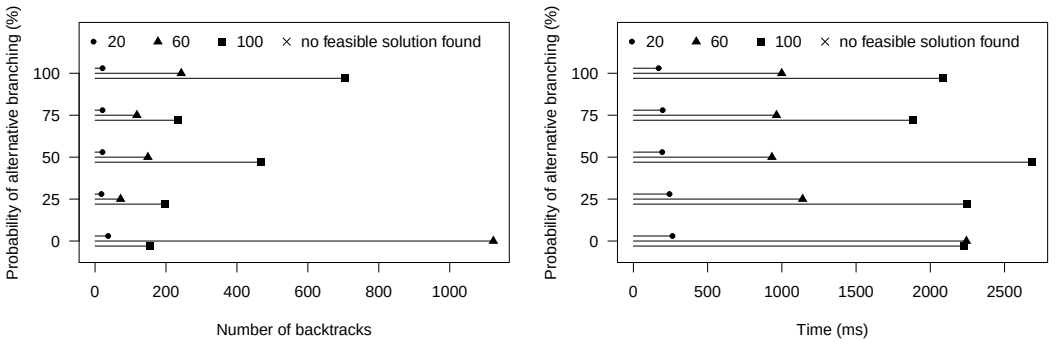


Figure B.6: Performance of CBASlackNoPEX strategy with temporal filtering.

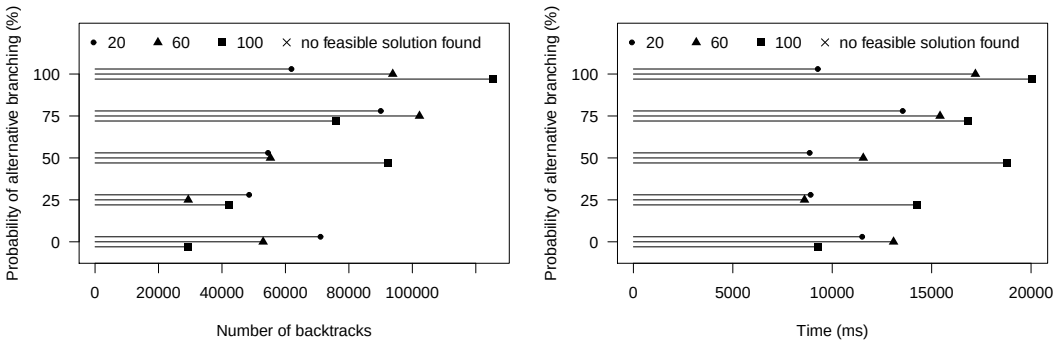


Figure B.7: Performance of LJRandNoPEX strategy.

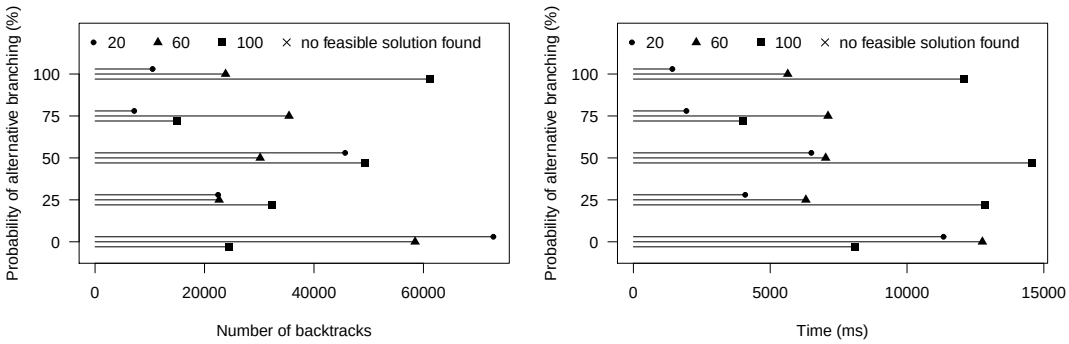


Figure B.8: Performance of LJRandNoPEX strategy with temporal filtering.

B.1.2 Comparison of the Strategies

In the following graphs, the values on the y axis denotes search strategies. Suffix "-tf" denotes that the strategy used temporal filtering. The strategies are:

- A — OptActTwoLevel
- B — OptActTwoLevelPair
- C — CBASlackNoPEX
- D — LJRandNoPEX

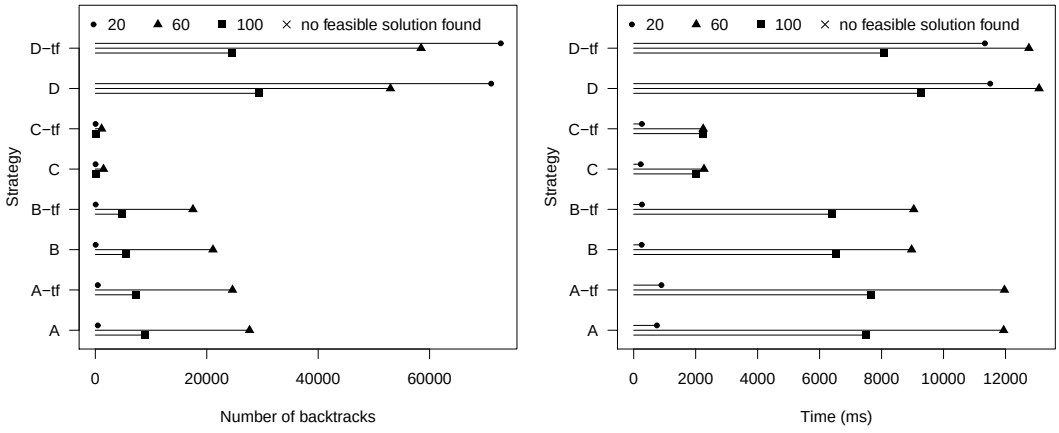


Figure B.9: Strategy comparison on n-TNA data with probability of alternative 0%.

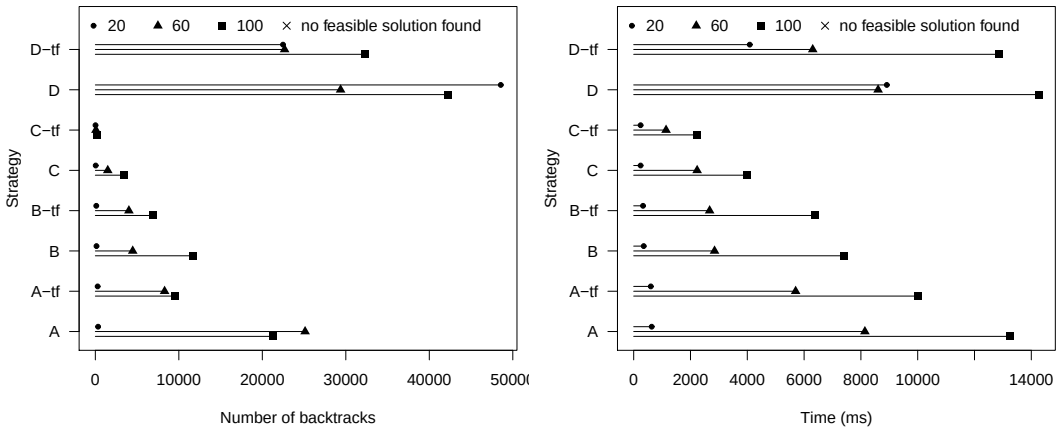


Figure B.10: Strategy comparison on n-TNA data with probability of alternative 25%.

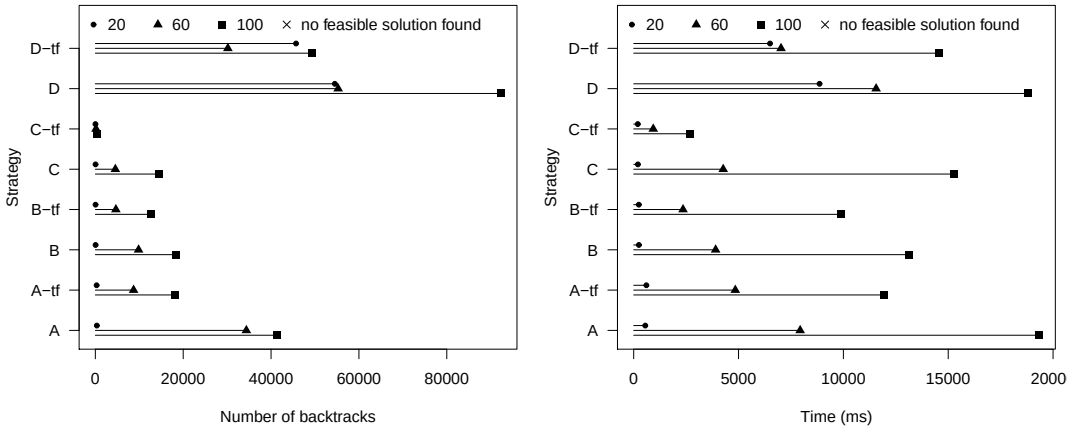


Figure B.11: Strategy comparison on n-TNA data with probability of alternative 50%.

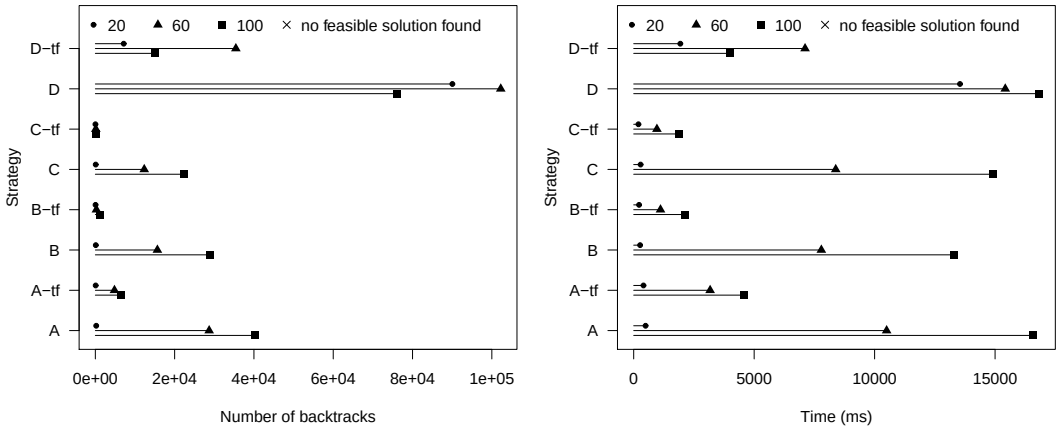


Figure B.12: Strategy comparison on n-TNA data with probability of alternative 75%.

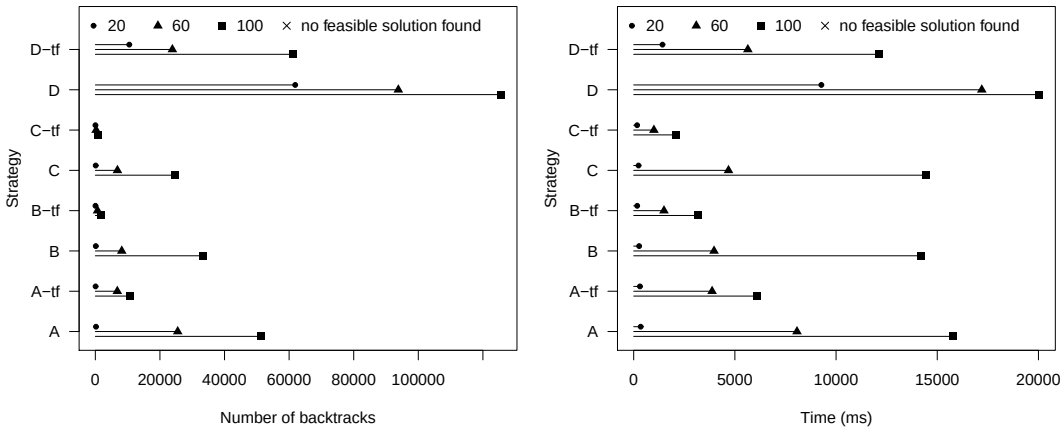


Figure B.13: Strategy comparison on n-TNA data with probability of alternative 100%.

B.2 Graphs of PEXTNA Results

B.2.1 Performace of the Strategies

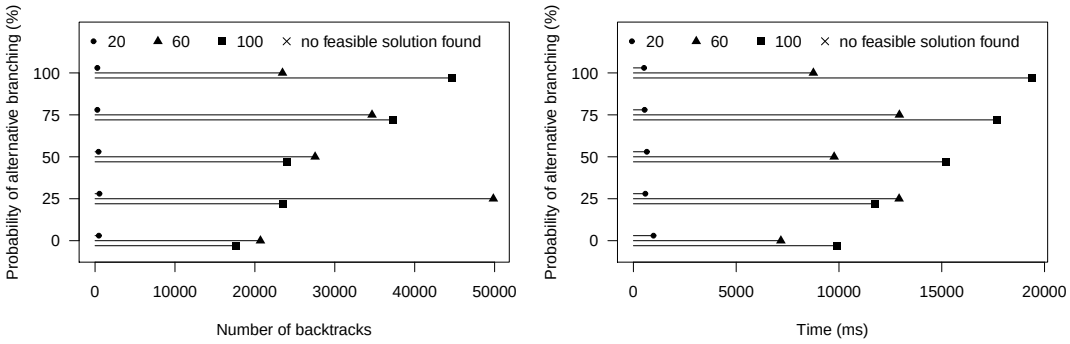


Figure B.14: Performance of OptActTwoLevel strategy.

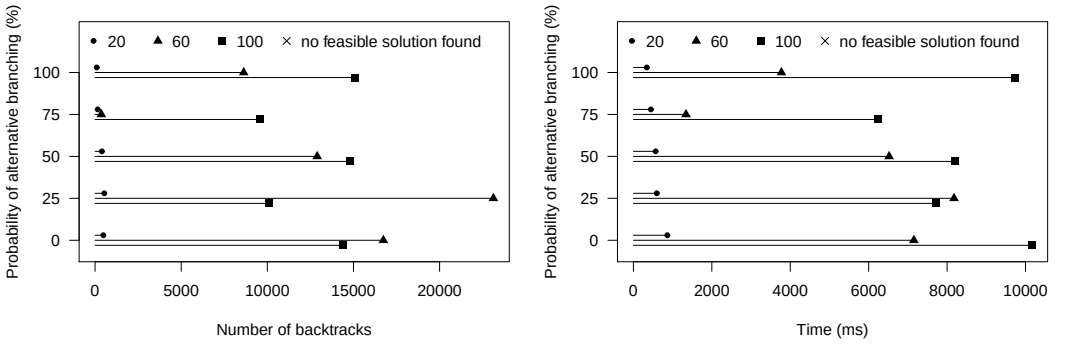


Figure B.15: Performance of OptActTwoLevel strategy with temporal filtering.

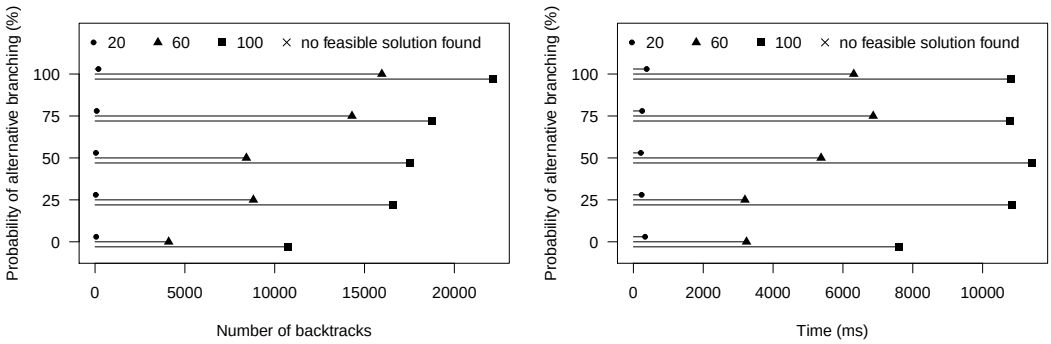


Figure B.16: Performance of OptActTwoLevelPair strategy.

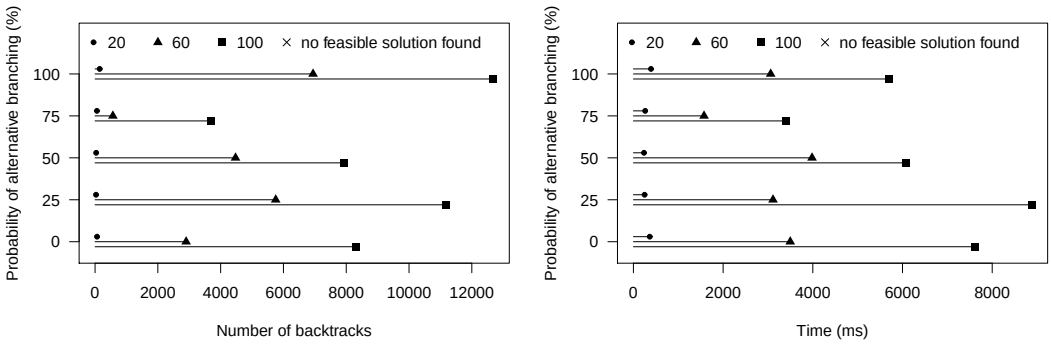


Figure B.17: Performance of OptActTwoLevelPair strategy with temporal filtering.

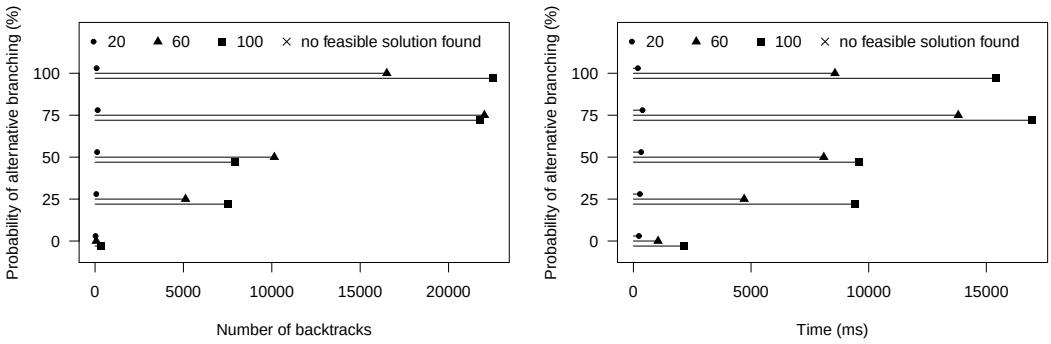


Figure B.18: Performance of CBASlackNoPEX strategy.

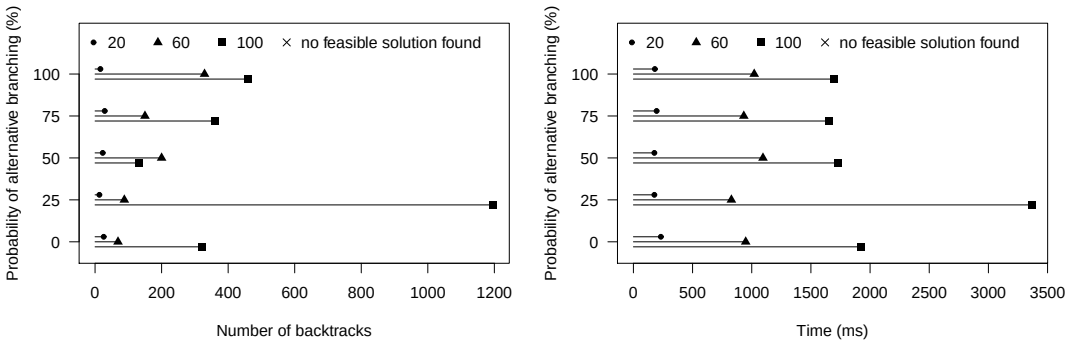


Figure B.19: Performance of CBASlackNoPEX strategy with temporal filtering.

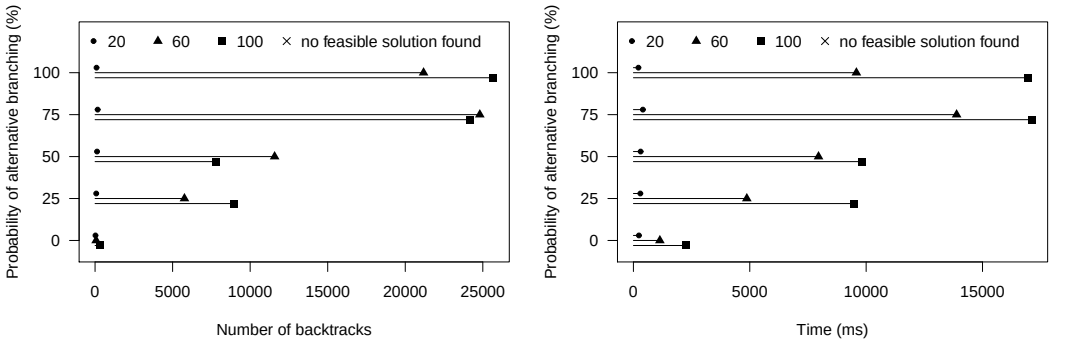


Figure B.20: Performance of CBASlackPEX strategy.

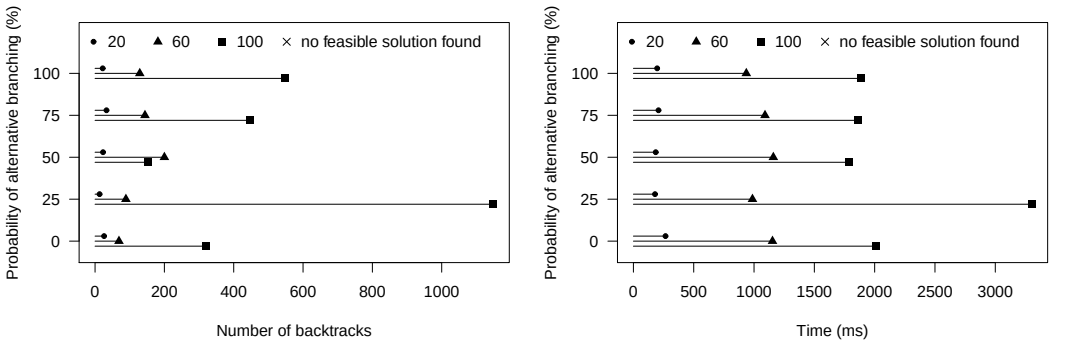


Figure B.21: Performance of CBASlackPEX strategy with temporal filtering.

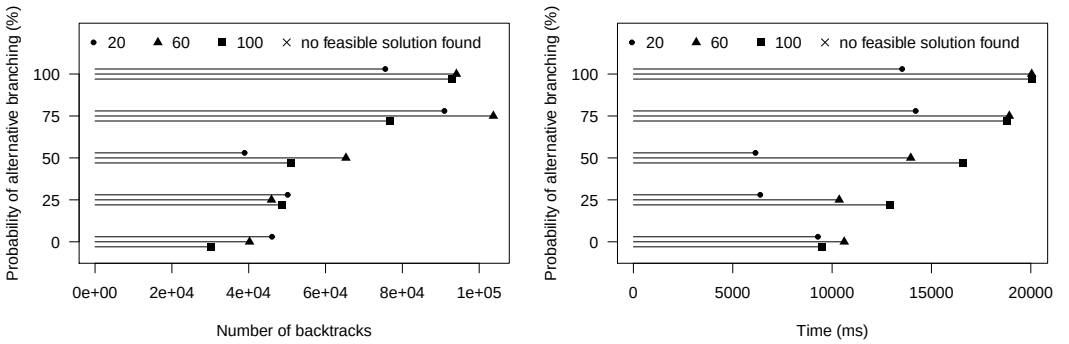


Figure B.22: Performance of LJRandNoPEX strategy.

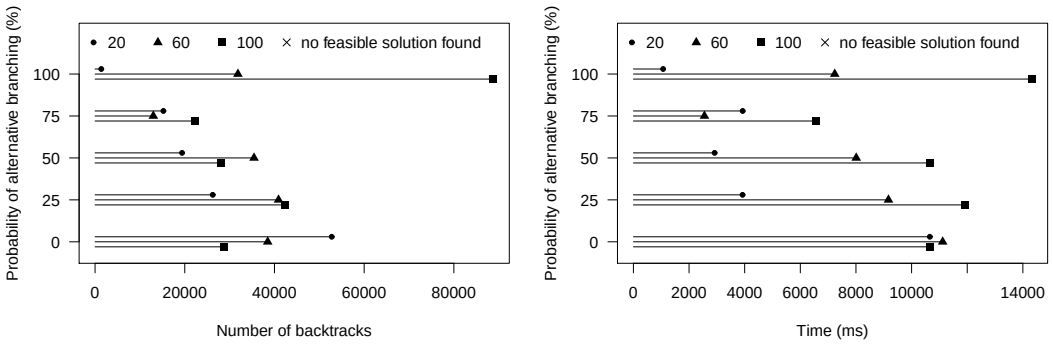


Figure B.23: Performance of LJRandNoPEX strategy with temporal filtering.

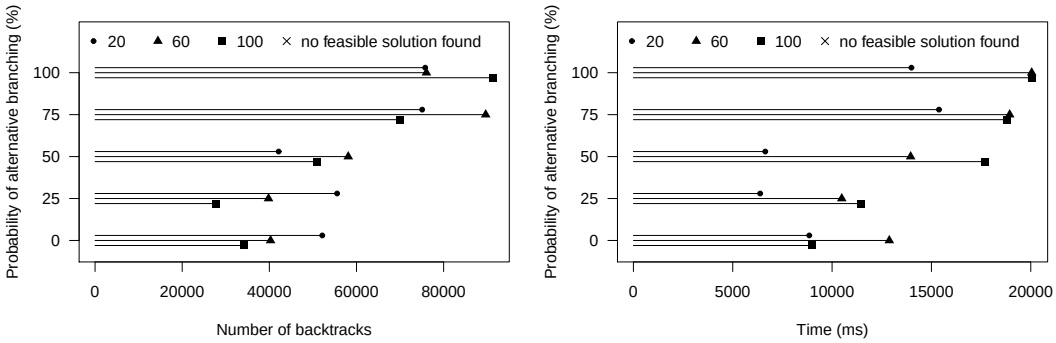


Figure B.24: Performance of LJRandPEX strategy.

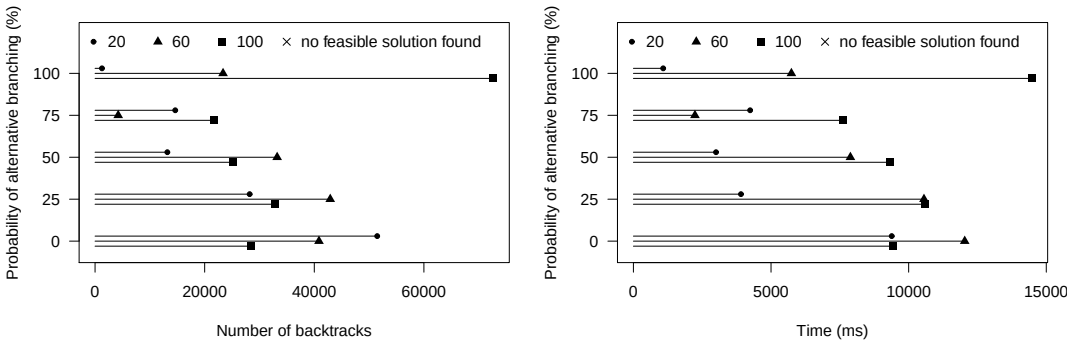


Figure B.25: Performance of LJRandPEX strategy with temporal filtering.

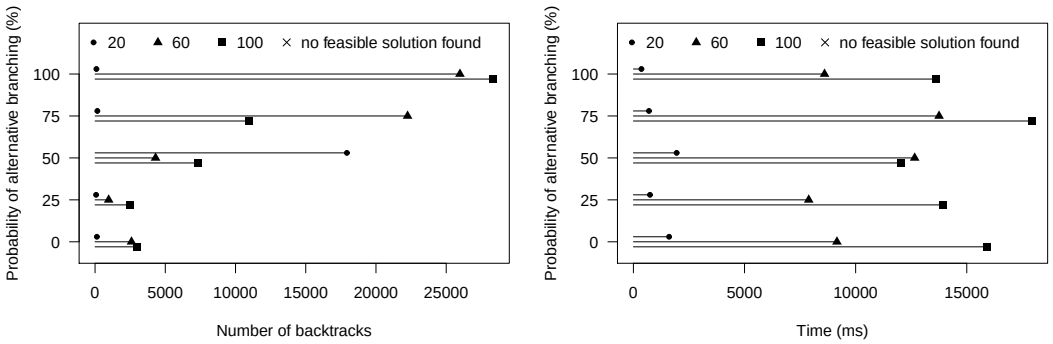


Figure B.26: Performance of SumHeightPEX strategy.

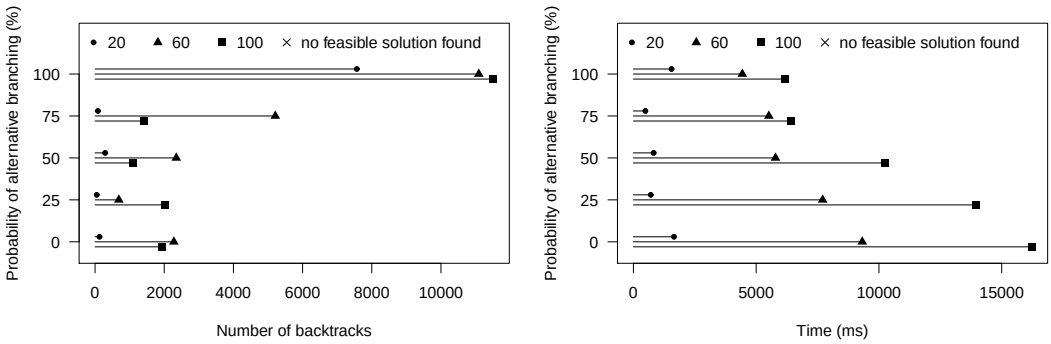


Figure B.27: Performance of SumHeightPEX strategy with temporal filtering.

B.2.2 Comparison of the Strategies

In the following graphs, the values on the y axis denotes search strategies. Suffix "-tf" denotes that the strategy used temporal filtering. The strategies are:

- A — OptActTwoLevel
- B — OptActTwoLevelPair
- C — CBASlackNoPEX
- D — LJRandNoPEX
- E — CBASlackPEX
- F — LJRandPEX
- G — SumHeightPEX

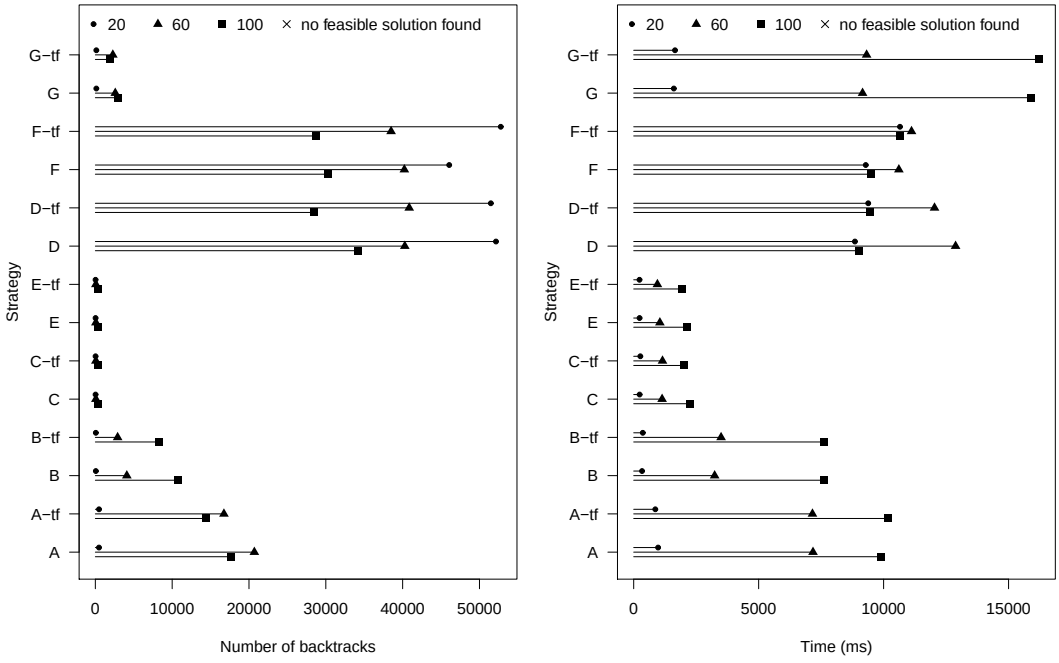


Figure B.28: Strategy comparison on PEXTNA data with probability of alternative 0%.

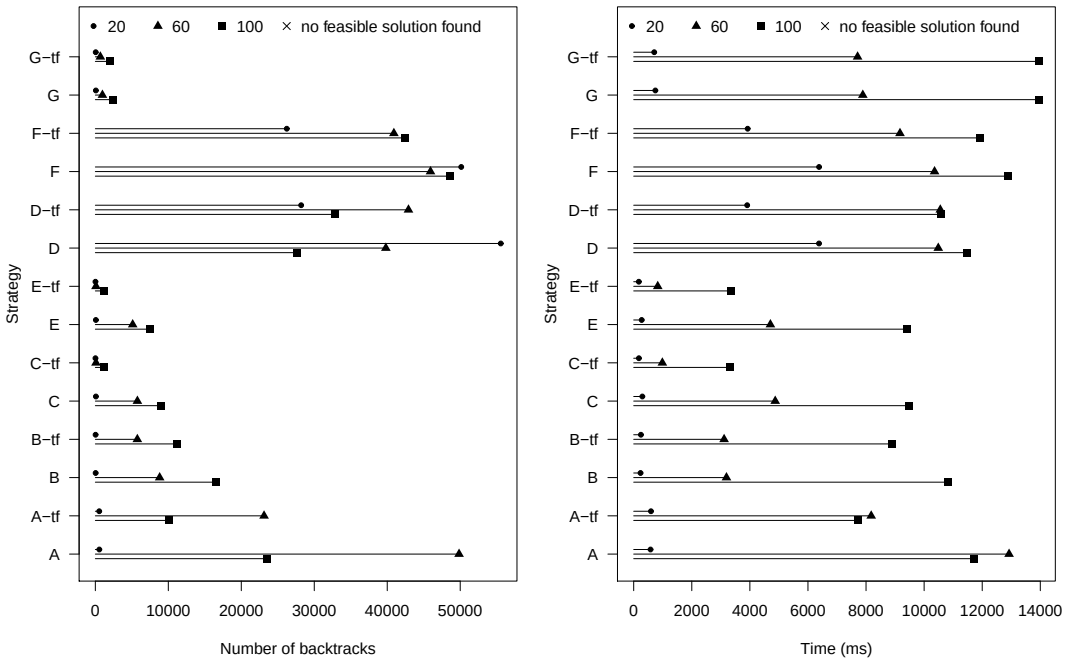


Figure B.29: Strategy comparison on PEXTNA data with probability of alternative 25%.

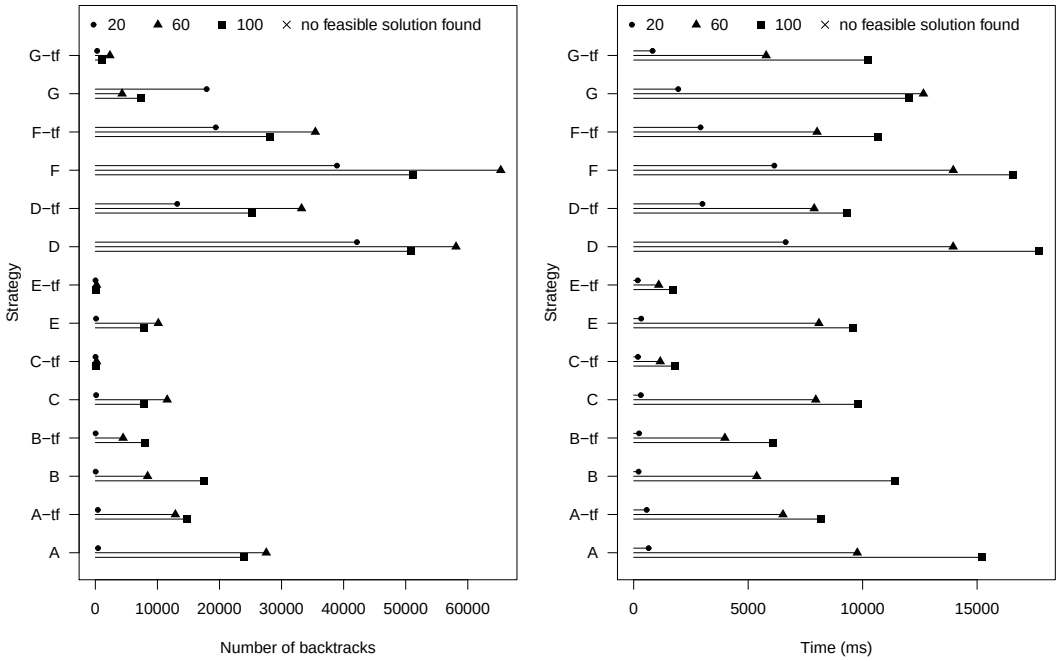


Figure B.30: Strategy comparison on PEXTNA data with probability of alternative 50%.

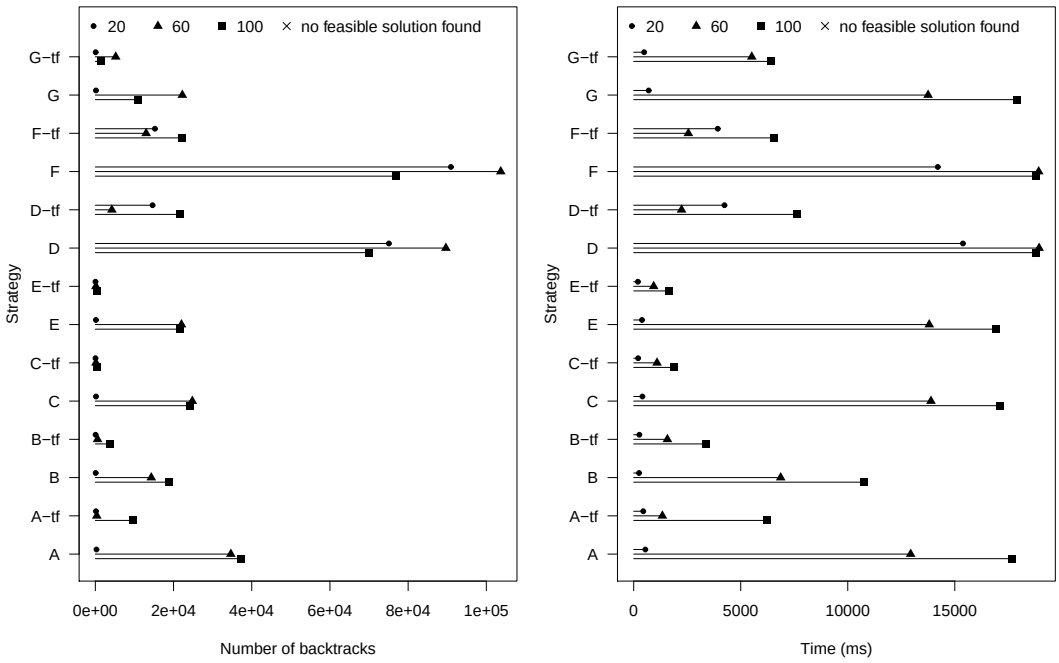


Figure B.31: Strategy comparison on PEXTNA data with probability of alternative 75%.

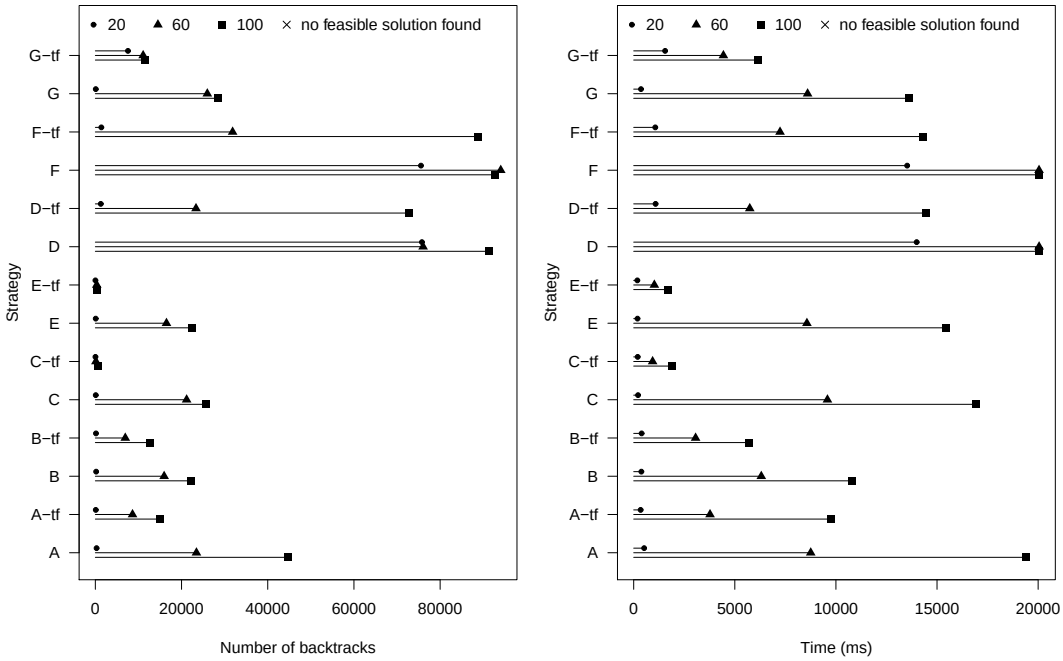


Figure B.32: Strategy comparison on PEXTNA data with probability of alternative 100%.

B.3 Graphs of x-RCPSP Results

Results of strategies tested on x-RCPSP.

B.3.1 Performace of the Strategies

In the following graphs, the values on the y axis denotes data configuration. The first digit denotes the proces complexity: 0 - low process complexity, 1 - high process complexity. The second digit denotes the resource complexity: 0 - low resource complexity, 1 - high resource complexity.

The left figure shows the average number of backtracks, the right figure shows the average running time of the solver.

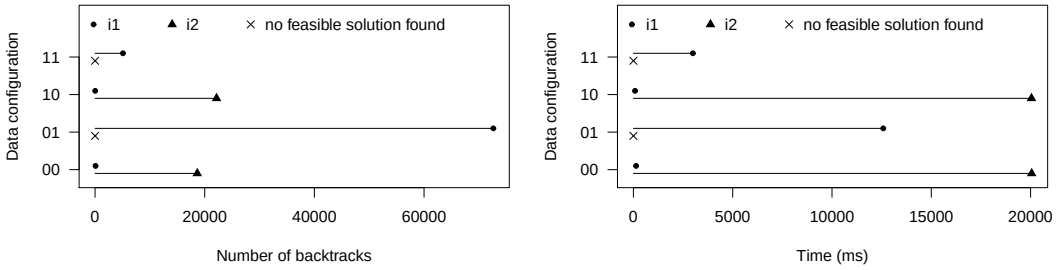


Figure B.33: Performance of OptActTwoLevel strategy.

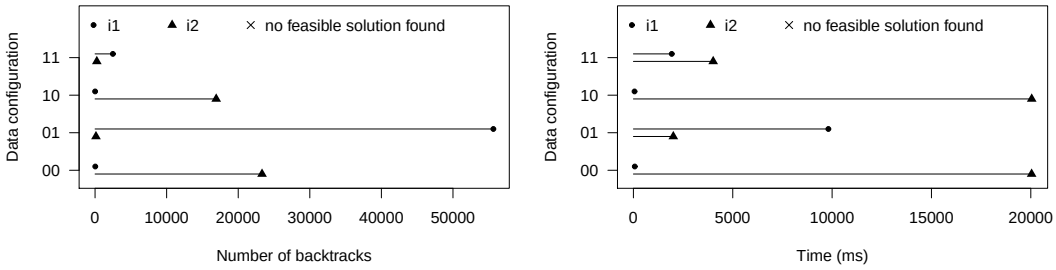


Figure B.34: Performance of OptActTwoLevelPair strategy.

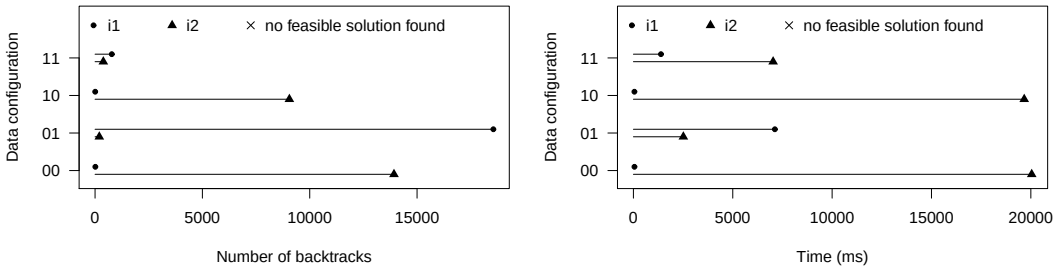


Figure B.35: Performance of CBASlackNoPEX strategy.

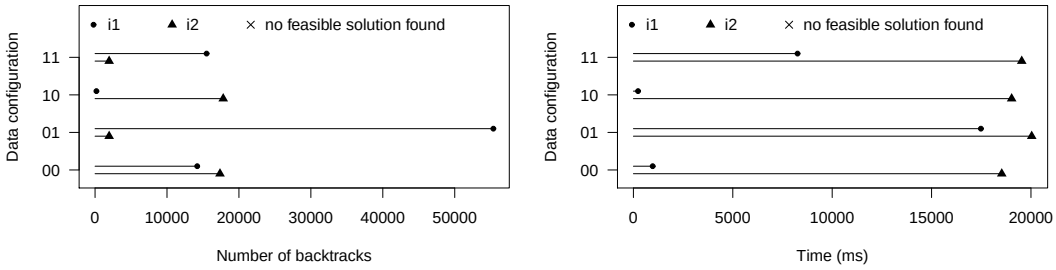


Figure B.36: Performance of LJRandNoPEX strategy.

B.3.2 Comparison of Strategies by Data Configuration

In the following graphs, the values on the y axis denotes search strategies. The strategies are:

- A — OptActTwoLevel
- B — OptActTwoLevelPair
- C — CBASlackNoPEX
- D — LJRandNoPEX

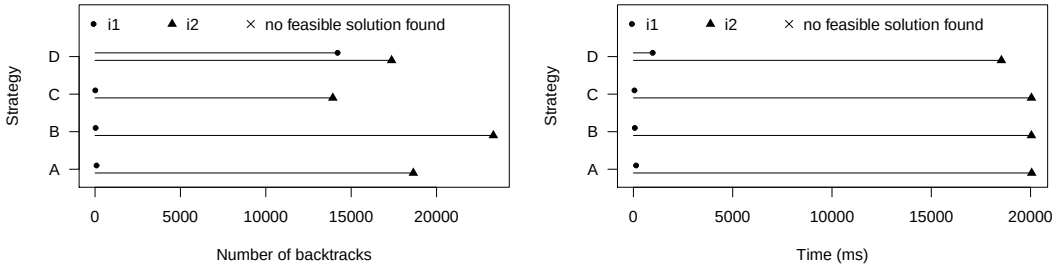


Figure B.37: Strategy comparison on data configuration 00: low process complexity, low resource complexity.

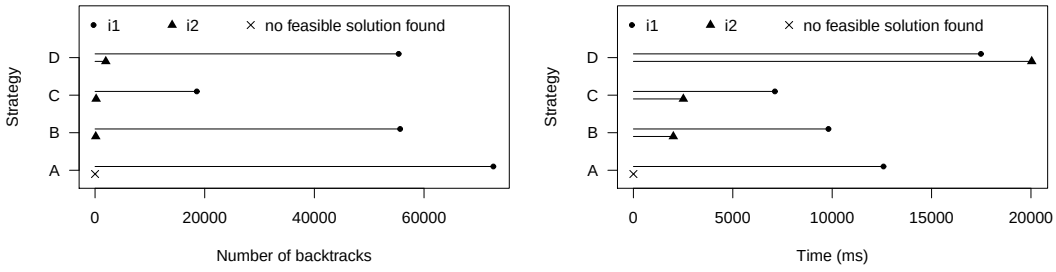


Figure B.38: Strategy comparison on data configuration 01: low process complexity, high resource complexity.

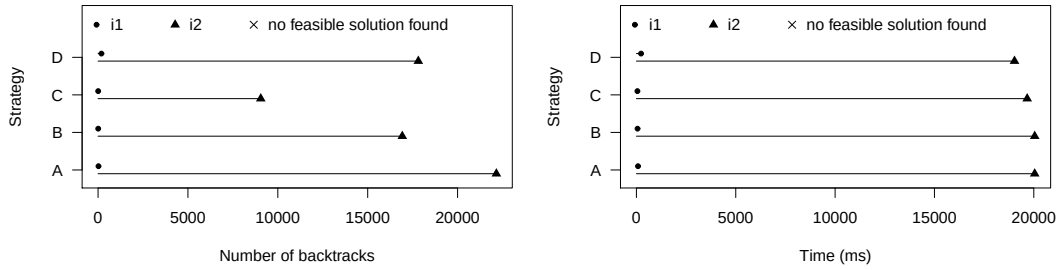


Figure B.39: Strategy comparison on data configuration 10: high process complexity, low resource complexity

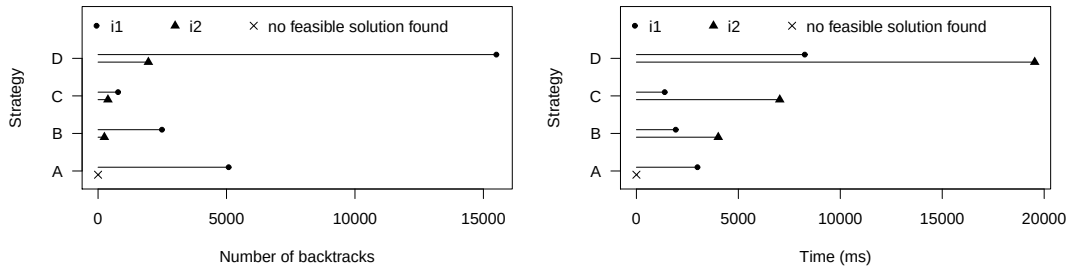


Figure B.40: Strategy comparison on data configuration 11: high process complexity, high resource complexity

C. Tables

Data used for plotting the graphs are summarized in the tables provided here. The parameters in the tables denote:

- Bt — Average number of backtracks
- F — Average number of failures (contradictions)
- N — Average number of nodes
- T — Average time
- FI — Number of feasible instances that ended before time limit
- II — Number of infeasible instances (thay ended before time limit)
- TFI — Number of feasible instances that ended on time limit
- UF — Number of instances with unknown feasibility (they ended on time limit)

C.1 Tables with n-TNA Results

Configuration	Parameter							
	Bt	F	N	T	FI	II	TFI	UF
20-0	37	37	51	227	15	1	0	0
60-0	1464	1463	1438	2269	15	0	1	0
100-0	155	155	348	2015	15	1	0	0
20-25	41	41	51	243	15	1	0	0
60-25	1488	1487	1351	2234	15	1	0	0
100-25	3469	3468	3010	3996	12	2	2	0
20-50	41	41	60	201	16	0	0	0
60-50	4556	4555	3853	4269	13	2	1	0
100-50	14578	14577	12039	15295	3	0	12	1
20-75	91	90	95	286	16	0	0	0
60-75	12334	12332	10135	8386	9	0	5	2
100-75	22332	22329	18002	14930	5	0	11	0
20-100	110	109	106	240	16	0	0	0
60-100	6845	6842	5664	4685	14	0	2	0
100-100	24527	24524	19893	14455	5	0	10	1

Table C.1: Test results for CBASlackNoPex strategy on n-TNA data.

Configuration	Parameter							
	Bt	F	N	T	FI	II	TFI	UF
20-0	37	37	51	263	15	1	0	0
60-0	1123	1122	1124	2243	15	0	1	0
100-0	155	155	348	2226	15	1	0	0
20-25	18	18	48	243	15	1	0	0
60-25	72	72	192	1140	15	1	0	0
100-25	198	198	373	2244	14	2	0	0
20-50	21	20	48	194	16	0	0	0
60-50	149	149	265	933	14	2	0	0
100-50	469	468	660	2683	16	0	0	0
20-75	21	21	59	197	16	0	0	0
60-75	118	117	285	964	16	0	0	0
100-75	233	233	557	1885	16	0	0	0
20-100	21	20	70	170	16	0	0	0
60-100	243	242	393	999	16	0	0	0
100-100	704	703	978	2083	16	0	0	0

Table C.2: Test results for CBASlackNoPex-TF strategy on n-TNA data.

Configuration	Parameter							
	Bt	F	N	T	FI	II	TFI	UF
20-0	71054	35541	35558	11506	6	1	9	0
60-0	52973	26510	26561	13084	6	0	10	0
100-0	29442	14750	14830	9285	8	1	7	0
20-25	48558	24300	24307	8912	11	0	4	1
60-25	29375	14817	14839	8606	9	1	6	0
100-25	42283	21357	21405	14275	3	2	11	0
20-50	54512	27315	27312	8863	10	0	6	0
60-50	55277	27928	27902	11562	5	2	9	0
100-50	92286	46623	46562	18782	0	0	15	1
20-75	90047	45078	45086	13544	8	0	8	0
60-75	102251	51399	51432	15423	4	0	12	0
100-75	75997	38784	38763	16828	3	0	13	0
20-100	61885	30996	31000	9273	12	0	4	0
60-100	93820	47315	47316	17201	3	0	13	0
100-100	125531	63368	63381	20033	0	0	16	0

Table C.3: Test results for LJRandNoPex strategy on n-TNA data.

Configuration	Parameter							
	Bt	F	N	T	FI	II	TFI	UF
20-0	72774	36404	36418	11334	6	1	9	0
60-0	58475	29267	29315	12754	6	0	10	0
100-0	24474	12298	12379	8091	10	1	5	0
20-25	22477	11245	11263	4086	13	0	2	1
60-25	22670	11371	11413	6304	11	1	4	0
100-25	32348	16351	16306	12865	4	2	10	0
20-50	45689	22867	22874	6502	11	0	5	0
60-50	30170	15135	15174	7027	9	2	5	0
100-50	49261	24740	24809	14557	5	0	11	0
20-75	7154	3593	3613	1936	15	0	1	0
60-75	35441	17776	17825	7113	11	0	5	0
100-75	15024	7671	7749	3997	14	0	2	0
20-100	10511	5266	5287	1423	15	0	1	0
60-100	23853	12003	12043	5641	13	0	3	0
100-100	61265	30790	30846	12099	7	0	9	0

Table C.4: Test results for LJRandNoPex-TF strategy on n-TNA data.

Configuration	Parameter							
	Bt	F	N	T	FI	II	TFI	UF
20-0	434	224	240	750	15	1	0	0
60-0	27682	13887	14026	11940	7	0	9	0
100-0	8938	4554	4799	7493	10	1	5	0
20-25	329	197	210	634	15	0	0	1
60-25	25123	12757	12861	8141	10	1	5	0
100-25	21308	10845	11041	13247	4	2	10	0
20-50	343	200	221	553	16	0	0	0
60-50	34380	17516	17583	7939	9	2	5	0
100-50	41391	21369	21485	19335	1	0	15	0
20-75	222	162	175	493	16	0	0	0
60-75	28708	14979	15027	10499	10	0	5	1
100-75	40357	21153	21212	16585	1	0	13	2
20-100	201	170	174	345	16	0	0	0
60-100	25507	13503	13554	8069	10	0	5	1
100-100	51432	27579	27634	15765	3	0	11	2

Table C.5: Test results for OptActTwoLevel strategy on n-TNA data.

Configuration	Parameter							
	Bt	F	N	T	FI	II	TFI	UF
20-0	50	35	50	256	15	1	0	0
60-0	21116	10592	10694	8970	10	0	6	0
100-0	5506	2787	3009	6518	11	1	4	0
20-25	135	86	96	352	15	1	0	0
60-25	4478	2291	2384	2847	14	1	1	0
100-25	11750	6014	6063	7397	9	2	5	0
20-50	51	41	61	247	16	0	0	0
60-50	9845	5247	4885	3908	12	2	2	0
100-50	18412	9825	9040	13153	5	0	10	1
20-75	108	91	94	266	16	0	0	0
60-75	15651	8732	7347	7789	13	0	3	0
100-75	28968	16683	13057	13283	9	0	7	0
20-100	131	109	102	269	16	0	0	0
60-100	8184	4847	3940	3971	14	0	1	1
100-100	33341	19850	14358	14201	6	0	10	0

Table C.6: Test results for OptActTwoLevelPair strategy on n-TNA data.

Configuration	Parameter							
	Bt	F	N	T	FI	II	TFI	UF
20-0	50	35	50	267	15	1	0	0
60-0	17525	8799	8897	9039	10	0	6	0
100-0	4764	2416	2635	6403	11	1	4	0
20-25	112	65	95	328	15	1	0	0
60-25	4021	2054	2165	2674	14	1	1	0
100-25	6937	3525	3738	6390	10	2	4	0
20-50	39	32	60	244	16	0	0	0
60-50	4680	2461	2500	2353	13	2	1	0
100-50	12692	6604	6700	9904	9	0	7	0
20-75	46	43	81	223	16	0	0	0
60-75	267	246	408	1113	16	0	0	0
100-75	1107	863	1136	2149	16	0	0	0
20-100	38	36	85	173	16	0	0	0
60-100	686	613	739	1497	16	0	0	0
100-100	1828	1227	1366	3182	16	0	0	0

Table C.7: Test results for OptActTwoLevelPair-TF strategy on n-TNA data.

Configuration	Parameter							
	Bt	F	N	T	FI	II	TFI	UF
20-0	434	224	240	895	15	1	0	0
60-0	24620	12358	12499	11966	7	0	9	0
100-0	7241	3664	3908	7648	10	1	5	0
20-25	275	154	185	602	15	0	0	1
60-25	8295	4245	4372	5702	12	1	3	0
100-25	9565	4921	5155	9998	7	2	7	0
20-50	284	156	184	604	16	0	0	0
60-50	8707	4486	4617	4846	11	2	3	0
100-50	18239	9516	9761	11958	8	0	8	0
20-75	82	66	104	406	16	0	0	0
60-75	4823	2558	2720	3173	15	0	1	0
100-75	6463	3693	3984	4578	14	0	2	0
20-100	70	64	111	310	16	0	0	0
60-100	6803	3617	3755	3875	15	0	1	0
100-100	10883	6175	6449	6106	13	0	3	0

Table C.8: Test results for OptActTwoLevel-TF strategy on n-TNA data.

C.2 Tables with PEXTNA Results

Configuration	Parameter							
	Bt	F	N	T	FI	II	TFI	UF
20-0	26	26	42	234	15	1	0	0
60-0	69	69	187	1048	15	0	0	1
100-0	321	321	478	2142	14	2	0	0
20-25	69	68	88	275	16	0	0	0
60-25	5119	5118	4248	4707	12	1	3	0
100-25	7533	7532	6360	9415	7	2	7	0
20-50	117	116	123	327	16	0	0	0
60-50	10143	10141	8541	8092	9	1	5	1
100-50	7914	7912	6639	9582	10	0	6	0
20-75	147	147	136	387	16	0	0	0
60-75	22029	22026	18248	13807	7	0	9	0
100-75	21772	21770	17566	16929	2	1	13	0
20-100	86	86	89	188	16	0	0	0
60-100	16506	16504	13302	8566	10	0	6	0
100-100	22535	22533	17818	15419	5	0	11	0

Table C.9: Test results for CBASlackNoPex strategy on PEX data.

Configuration	Parameter							
	Bt	F	N	T	FI	II	TFI	UF
20-0	26	26	42	232	15	1	0	0
60-0	69	70	187	949	15	0	0	1
100-0	321	321	478	1924	14	2	0	0
20-25	13	13	44	177	16	0	0	0
60-25	88	88	180	828	15	1	0	0
100-25	1197	1197	1308	3366	13	2	1	0
20-50	23	23	50	177	16	0	0	0
60-50	200	199	294	1095	14	1	0	1
100-50	132	132	399	1729	16	0	0	0
20-75	29	29	57	196	16	0	0	0
60-75	150	150	314	933	16	0	0	0
100-75	360	359	624	1652	15	1	0	0
20-100	16	16	52	181	16	0	0	0
60-100	329	329	462	1021	16	0	0	0
100-100	460	459	675	1692	16	0	0	0

Table C.10: Test results for CBASlackNoPex-TF strategy on PEX data.

Configuration	Parameter							
	Bt	F	N	T	FI	II	TFI	UF
20-0	26	26	42	237	15	1	0	0
60-0	69	69	187	1137	15	0	0	1
100-0	321	321	478	2255	14	2	0	0
20-25	76	67	88	298	16	0	0	0
60-25	5763	4662	3891	4873	12	1	3	0
100-25	8937	7480	6382	9482	7	2	7	0
20-50	128	116	123	310	16	0	0	0
60-50	11577	9454	8100	7954	9	1	5	1
100-50	7795	6352	5553	9805	10	0	6	0
20-75	168	143	136	407	16	0	0	0
60-75	24807	20291	17275	13887	7	0	9	0
100-75	24188	18701	15926	17114	3	1	12	0
20-100	94	88	93	215	16	0	0	0
60-100	21180	16420	13777	9581	9	0	7	0
100-100	25680	19370	16347	16945	3	0	13	0

Table C.11: Test results for CBASlackPex strategy on PEX data.

Configuration	Parameter							
	Bt	F	N	T	FI	II	TFI	UF
20-0	26	26	42	266	15	1	0	0
60-0	69	70	187	1153	15	0	0	1
100-0	321	321	478	2011	14	2	0	0
20-25	13	13	44	179	16	0	0	0
60-25	89	88	180	987	15	1	0	0
100-25	1149	1148	1264	3302	13	2	1	0
20-50	23	23	50	185	16	0	0	0
60-50	200	198	285	1160	13	2	0	1
100-50	153	152	420	1785	16	0	0	0
20-75	33	31	57	208	16	0	0	0
60-75	144	144	294	1091	16	0	0	0
100-75	446	441	664	1860	15	1	0	0
20-100	22	21	55	197	16	0	0	0
60-100	129	125	259	936	16	0	0	0
100-100	548	540	693	1889	16	0	0	0

Table C.12: Test results for CBASlackPex-TF strategy on PEX data.

Configuration	Parameter							
	Bt	F	N	T	FI	II	TFI	UF
20-0	46061	23040	23058	9285	8	1	7	0
60-0	40222	20133	20182	10611	7	0	8	1
100-0	30220	15173	15254	9499	7	2	7	0
20-25	50138	25091	25110	6383	11	0	5	0
60-25	45927	23055	23100	10359	7	1	8	0
100-25	48665	24475	24525	12900	5	2	9	0
20-50	38923	19488	19501	6142	12	0	4	0
60-50	65313	32846	32835	13958	3	1	11	1
100-50	51097	25980	26017	16578	3	0	13	0
20-75	90935	45515	45525	14205	7	0	9	0
60-75	103688	52207	52224	18919	1	0	15	0
100-75	76830	38972	39004	18783	0	1	15	0
20-100	75551	37850	37845	13524	8	0	8	0
60-100	94065	47505	47466	20045	0	0	16	0
100-100	92809	47101	47043	20046	0	0	16	0

Table C.13: Test results for LJRandNoPex strategy on PEX data.

Configuration	Parameter							
	Bt	F	N	T	FI	II	TFI	UF
20-0	52771	26395	26413	10655	8	1	7	0
60-0	38497	19321	19333	11117	7	0	8	1
100-0	28681	14382	14461	10648	6	2	8	0
20-25	26234	13132	13149	3924	13	0	3	0
60-25	40910	20489	20536	9169	8	1	7	0
100-25	42419	21407	21349	11928	5	2	9	0
20-50	19398	9715	9733	2919	14	0	2	0
60-50	35444	17744	17792	8010	8	1	6	1
100-50	28174	14210	14282	10650	9	0	7	0
20-75	15236	7634	7651	3928	14	0	2	0
60-75	12992	6554	6598	2554	15	0	1	0
100-75	22205	11205	11285	6554	11	1	4	0
20-100	1391	709	726	1068	16	0	0	0
60-100	31853	16004	16053	7237	12	0	4	0
100-100	88806	44603	44651	14318	5	0	11	0

Table C.14: Test results for LJRandNoPex-TF strategy on PEX data.

Configuration	Parameter							
	Bt	F	N	T	FI	II	TFI	UF
20-0	52149	26085	26101	8852	8	1	7	0
60-0	40281	20163	20211	12880	5	0	10	1
100-0	34211	17142	17224	9007	8	2	6	0
20-25	55555	27801	27820	6381	11	0	5	0
60-25	39809	19989	20032	10487	7	1	8	0
100-25	27681	13965	14021	11461	6	2	8	0
20-50	42137	21096	21110	6637	12	0	4	0
60-50	58108	29231	29217	13954	3	1	11	1
100-50	50879	25888	25936	17708	2	0	14	0
20-75	75074	37584	37594	15381	8	0	8	0
60-75	89653	45199	45193	18939	1	0	15	0
100-75	69996	35576	35599	18789	0	1	15	0
20-100	75773	37972	37963	13993	7	0	9	0
60-100	76046	38485	38431	20048	0	0	16	0
100-100	91410	46321	46335	20044	0	0	16	0

Table C.15: Test results for LJRandPex strategy on PEX data.

Configuration	Parameter							
	Bt	F	N	T	FI	II	TFI	UF
20-0	51480	25748	25767	9385	8	1	7	0
60-0	40856	20497	20500	12037	6	0	9	1
100-0	28495	14304	14386	9443	8	2	6	0
20-25	28205	14114	14133	3908	13	0	3	0
60-25	42910	21501	21547	10557	7	1	8	0
100-25	32893	16520	16594	10585	7	2	7	0
20-50	13190	6611	6629	3002	14	0	2	0
60-50	33224	16637	16677	7884	7	2	6	1
100-50	25196	12722	12796	9314	10	0	6	0
20-75	14632	7335	7351	4241	14	0	2	0
60-75	4226	2177	2225	2236	15	0	1	0
100-75	21695	10972	11042	7609	10	1	5	0
20-100	1262	646	664	1082	16	0	0	0
60-100	23356	11748	11800	5740	13	0	3	0
100-100	72672	36585	36626	14471	5	0	11	0

Table C.16: Test results for LJRandPex-TF strategy on PEX data.

Configuration	Parameter							
	Bt	F	N	T	FI	II	TFI	UF
20-0	476	246	264	977	15	1	0	0
60-0	20697	10388	10530	7175	10	0	5	1
100-0	17669	8969	9203	9880	7	2	7	0
20-25	546	283	309	580	16	0	0	0
60-25	49843	25059	25154	12925	5	1	10	0
100-25	23544	11937	12119	11733	4	2	8	2
20-50	432	252	273	652	16	0	0	0
60-50	27537	14076	14140	9766	8	1	6	1
100-50	23961	12582	12692	15195	5	0	11	0
20-75	286	200	209	544	16	0	0	0
60-75	34660	17896	17925	12939	8	0	8	0
100-75	37282	20447	20491	17668	3	1	12	0
20-100	286	218	226	517	16	0	0	0
60-100	23457	12427	12448	8755	10	0	5	1
100-100	44699	23723	23772	19376	1	0	15	0

Table C.17: Test results for OptActTwoLevel strategy on PEX data.

Configuration	Parameter							
	Bt	F	N	T	FI	II	TFI	UF
20-0	63	38	56	334	15	1	0	0
60-0	4099	2069	2195	3239	14	0	1	1
100-0	10761	5430	5614	7615	9	2	5	0
20-25	38	29	55	235	16	0	0	0
60-25	8814	4489	4508	3194	14	1	1	0
100-25	16566	8565	8378	10838	6	2	8	0
20-50	48	40	57	212	16	0	0	0
60-50	8424	4415	4272	5375	12	2	2	0
100-50	17547	9704	8384	11406	9	0	7	0
20-75	90	73	75	250	16	0	0	0
60-75	14302	7935	6689	6867	13	0	3	0
100-75	18749	10696	8662	10778	9	1	6	0
20-100	188	129	130	379	16	0	0	0
60-100	15964	8699	7690	6310	14	0	2	0
100-100	22171	12282	10904	10799	11	0	5	0

Table C.18: Test results for OptActTwoLevelPair strategy on PEX data.

Configuration	Parameter							
	Bt	F	N	T	FI	II	TFI	UF
20-0	63	38	56	364	15	1	0	0
60-0	2901	1471	1596	3497	14	0	1	1
100-0	8314	4206	4388	7623	9	2	5	0
20-25	31	23	54	249	16	0	0	0
60-25	5754	2924	3014	3113	14	1	1	0
100-25	11169	5652	5846	8881	8	2	6	0
20-50	36	29	55	236	16	0	0	0
60-50	4474	2335	2404	3983	12	2	2	0
100-50	7931	4176	4377	6068	14	0	2	0
20-75	59	46	72	263	16	0	0	0
60-75	568	379	525	1575	16	0	0	0
100-75	3702	2176	2243	3401	14	1	1	0
20-100	148	94	129	392	16	0	0	0
60-100	6944	3612	3761	3061	15	0	1	0
100-100	12686	6885	6803	5707	14	0	2	0

Table C.19: Test results for OptActTwoLevelPair-TF strategy on PEX data.

Configuration	Parameter							
	Bt	F	N	T	FI	II	TFI	UF
20-0	480	248	266	865	15	1	0	0
60-0	16738	8409	8554	7153	10	0	5	1
100-0	14393	7330	7565	10158	7	2	7	0
20-25	536	273	304	595	16	0	0	0
60-25	23120	11640	11747	8178	9	1	6	0
100-25	10103	5248	5471	7708	8	2	5	1
20-50	397	216	245	567	16	0	0	0
60-50	12893	6552	6665	6521	10	1	4	1
100-50	14825	7881	7983	8197	11	0	5	0
20-75	154	101	130	446	16	0	0	0
60-75	368	342	492	1341	16	0	0	0
100-75	9553	5364	5616	6248	13	1	2	0
20-100	99	71	108	343	16	0	0	0
60-100	8628	4565	4729	3775	14	0	2	0
100-100	15101	8286	8471	9734	13	0	3	0

Table C.20: Test results for OptActTwoLevel-TF strategy on PEX data.

Configuration	Parameter							
	Bt	F	N	T	FI	II	TFI	UF
20-0	129	128	136	1606	15	1	0	0
60-0	2594	2523	665	9158	12	0	3	1
100-0	2975	2713	1002	15909	3	2	11	0
20-25	73	70	92	746	16	0	0	0
60-25	971	898	884	7889	9	1	4	2
100-25	2469	2419	1101	13945	1	2	11	2
20-50	17929	13598	4424	1945	15	0	1	0
60-50	4308	4052	1867	12656	5	1	9	1
100-50	7317	5227	3225	12036	5	0	9	2
20-75	163	148	149	700	16	0	0	0
60-75	22250	18169	5972	13753	5	0	10	1
100-75	10958	9082	4323	17922	2	1	13	0
20-100	95	91	95	361	14	0	0	2
60-100	25978	20974	5709	8600	7	0	5	4
100-100	28357	24327	5576	13611	2	0	10	4

Table C.21: Test results for SumHeightPex strategy on PEX data.

Configuration	Parameter							
	Bt	F	N	T	FI	II	TFI	UF
20-0	129	128	136	1653	15	1	0	0
60-0	2278	2219	645	9320	12	0	3	1
100-0	1933	1779	822	16222	2	2	12	0
20-25	50	50	78	707	16	0	0	0
60-25	688	681	727	7709	11	1	4	0
100-25	2018	2012	999	13959	4	2	10	0
20-50	291	276	106	824	16	0	0	0
60-50	2349	2316	747	5786	10	2	3	1
100-50	1102	1080	1250	10233	10	0	6	0
20-75	82	79	108	487	16	0	0	0
60-75	5211	5001	877	5516	14	0	2	0
100-75	1412	1391	1586	6409	13	1	2	0
20-100	7568	7567	2016	1554	15	0	1	0
60-100	11094	9310	2488	4436	14	0	2	0
100-100	11515	9008	4622	6164	14	0	2	0

Table C.22: Test results for SumHeightPex-TF strategy on PEX data.

C.3 Tables with x-RCPSP Results

Configuration	Parameter							
	Bt	F	N	T	FI	II	TFI	UF
i1-00	13	13	20	52	40	0	0	0
i1-01	18554	18554	16296	7118	33	0	7	0
i1-10	7	6	18	50	40	0	0	0
i1-11	779	779	706	1387	40	0	0	0
i2-00	13925	13922	13626	20044	0	0	40	0
i2-01	198	198	482	2513	0	0	5	35
i2-10	9052	9047	9042	19667	1	0	39	0
i2-11	385	384	1317	7031	0	0	14	26

Table C.23: Test results for CBASlackNoPex strategy on data.

Configuration	Parameter							
	Bt	F	N	T	FI	II	TFI	UF
i1-00	14203	5297	13962	969	40	0	0	0
i1-01	55370	27736	27709	17482	11	0	29	0
i1-10	187	98	104	234	40	0	0	0
i1-11	15504	7776	7895	8257	33	0	7	0
i2-00	17368	9086	8879	18531	0	0	37	3
i2-01	1960	1379	1445	20038	0	0	40	0
i2-10	17816	9438	9198	19029	0	0	38	2
i2-11	1959	1326	1365	19536	0	0	39	1

Table C.24: Test results for LJRandNoPex strategy on data.

Configuration	Parameter							
	Bt	F	N	T	FI	II	TFI	UF
i1-00	87	48	57	136	40	0	0	0
i1-01	72645	36331	36346	12581	25	0	15	0
i1-10	23	15	27	81	40	0	0	0
i1-11	5083	2551	2563	2994	40	0	0	0
i2-00	18640	9370	10021	20057	0	0	40	0
i2-01	0	0	0	0	0	0	0	40
i2-10	22157	11280	11894	20045	0	0	40	0
i2-11	0	0	0	0	0	0	0	40

Table C.25: Test results for OptActTwoLevel strategy on data.

Configuration	Parameter							
	Bt	F	N	T	FI	II	TFI	UF
i1-00	31	19	27	69	40	0	0	0
i1-01	55642	27860	27813	9812	30	0	10	0
i1-10	9	6	18	56	40	0	0	0
i1-11	2487	1255	1258	1929	40	0	0	0
i2-00	23327	11787	12313	20041	0	0	40	0
i2-01	128	122	375	2008	0	0	4	36
i2-10	16924	8625	9132	20046	0	0	40	0
i2-11	246	229	750	4015	0	0	8	32

Table C.26: Test results for OptActTwoLevelPair strategy on data.

D. Compact Disc Content

The enclosed compact disc contains electronic version of the thesis, test program implementing the search strategies and the whole testing framework implemented to evaluate the search strategies. Next, it contains input test data as well as scheduled output data for each test case.

The directory structure on the disk is:

- `thesis` — The directory contains electronic version of the thesis.
- `program` — The directory contains the exacutable jar file of the test program used to evalute the strategies, more about it in section A.1.
- `test-framework` — The directory contains the source files for the whole test framework. Besides the test program source files, it contains scripts for batch processing, source files of other utility programs contained in the framework. To allow immediate usage, necessary compiled files are also presented in the fromework. The contents and usage of the whole framework is described in section A.2.
- `inputdata` — The directory contains the input test data.
- `outputdata` — The directory contains in scheduled instances of the input test data for each strategy.