COPLAS 2011

Proceedings of the Workshop on Constraint Satisfaction Techniques for Planning and Scheduling Problems

Freiburg, Germany
June 13, 2011

Edited by
Miguel A. Salido, Roman Barták and Nicola Policella

Preface

The area of AI planning and scheduling has seen important advances thanks to the application of constraint satisfaction and optimization techniques. Efficient constraint handling is important for real-world problems in planning, scheduling, and resource allocation to competing goal activities over time in the presence of complex state-dependent constraints. Approaches to these problems must integrate resource allocation and plan synthesis capabilities. We need to manage complex problems where planning, scheduling, and constraint satisfaction must be interrelated, which entail a great potential of application.

The workshop on Constraint Satisfaction Techniques for Planning and Scheduling Problems, or COPLAS, aims at providing a forum for meeting and exchanging ideas and novel works in the field of AI planning, scheduling, and constraint satisfaction techniques, and the many relationships that exist among them. In fact, most of the accepted papers are based on combined approaches of constraint satisfaction for planning, scheduling, and mixing planning and scheduling. This makes the COPLAS workshop an attractive place for both researchers and practitioners (COPLAS is ranked as CORE B in ERA Conference Ranking).

The sixth edition of the workshop, COPLAS 2011, was held in June 2011 in Freiburg, Germany during the International Conference on Automated Planning and Scheduling (ICAPS'11). All the submissions were reviewed by at least three anonymous referees from the program committee. The nine papers accepted for oral presentation in the workshop, provide a mix of constraint satisfaction and optimization techniques for planning, scheduling, and related topics, as well as their applications to real-world problems.

We hope that the ideas and approaches presented in the papers and presentations will lead to a valuable discussion and will inspire future research and developments for all the readers.

The Organizing Committee.
June, 2011

Miguel A. Salido
Roman Barták
Nicola Policella

Organization

Organizing Committee

Miguel A. Salido, Universidad Politécnica de Valencia, Spain
Roman Barták, Charles University, Czech Republic
Nicola Policella, European Space Agency - ESA/ESOC, Germany

Programme Committee

Federico Barber, Universidad Politécnica de Valencia, Spain

Roman Barták, Charles University, The Czech Republic

Amedeo Cesta, ISTC-CNR, Italy

Minh Binh Do, PARC, USA

Enrico Giunchiglia, Universita di Genova, Italy

Peter Jarvis, NASA Ames Research Center, USA

Michela Milano, Università di Bologna, Italy

Alexander Nareyek, National University of Singapore, Singapore

Eva Onaindía, Universidad Politécnica de Valencia, Spain

Nicola Policella, European Space Agency, Germany

Francesca Rossi, University of Padova, Italy

Hana Rudová, Masaryk University, The Czech Republic

Migual A. Salido, Universidad Politecnica Valencia, Spain

Pascal Van Hentenryck, Brown University, USA

Ramiro Varela, Universidad de Oviedo, Spain

Gérard Verfaillie, ONERA, Centre de Toulouse, France

Vincent Vidal, CRIL-IUT, France

Petr Vilím, ILOG, France

Toby Walsh, University of New South Wales, Australia and NICTA, Australia

Neil Yorke-Smith, American University of Beirut, Lebanon and SRI International, USA

Content

A Logic-Based Benders Approach to Scheduling with Alternative Resources and Setup Times

Tony T. Tran and **J. Christopher Beck**
Department of Mechanical and Industrial Engineering
University of Toronto, Toronto, Ontario, Canada
{tran,jcb}@mie.utoronto.ca

Abstract

We study an unrelated parallel machines scheduling problem with sequence and machine dependent setup times. A logic-based Benders decomposition approach is proposed to minimize the makespan. The decomposition approach is a hybrid model that makes use of a mixed integer programming master problem and a travelling salesman problem subproblem. The master problem is a relaxation of the problem and is used to create assignments of jobs to machines, while the subproblem obtains optimal schedules based on the master problem assignments. Computational results comparing the Benders decomposition and mixed integer program formulation show that the Benders model is able to find optimal solutions to problems up to five orders of magnitude faster as well as solving problems four times the size possible previously.

1 Introduction

In many practical scheduling problems, the scheduler is faced with both resource alternatives and sequence dependent setup times. That is, a job may be assigned to one of a set of resources and consecutive jobs on the same resource must have a minimum setup time between them. For example, in a chemical plant, reactors must be cleaned when changing from processing one mixture to another. The cleaning times may depend on which specific job comes before the cleaning and which comes after. If the preceding chemical affects the succeeding one, cleaning may take longer to ensure that the reactor is properly prepared. The products processed in the reverse order may not have the same ill effects because the offending product in the previous example may not suffer the same contamination and so, cleaning may take less time. Further examples can be found in the plastic, glass, paper and textile industries where setup times of significant length exist (França et al. 1996). Allahverdi et al.(1999) review the importance of setup times in real world problems.

This paper addresses the unrelated parallel machines scheduling problem (PMSP) with machine and sequence dependent setup times. In this problem, jobs must be assigned to one of a set of alternative resources. Jobs assigned to the same resource have a setup time which is defined as the time that must elapse between the end of one job and the start of the next. This setup time is sequence and machine dependent in that the elapsed time between jobs j and k will differ depending on whether j precedes k or k precedes j and which machine the pair of jobs are assigned to. We concern ourselves with minimizing the makespan or the maximum completion time of a schedule, C_{max}, as the objective function. Using the three-field notation given by Graham et al. (1979), this problem can be denoted as (R|sds|Cmax).

We develop an exact method to solve the PMSP based on logic-based Benders decomposition. This approach was developed to verify logic circuits by Hooker (1995) and further explained later (Hooker and Ottosson 2003). Hooker (2005) applied logic-based Benders decomposition to a problem similar to the PMSP with release and due dates, but without setup times. The Benders decomposition introduced in this paper makes use of a Mixed Integer Program (MIP) master problem and either a Constraint Program (CP) solver or a specialized Travelling Salesman Problem (TSP) solver for the subproblems. The MIP model is used to find tight lower bounds and machine assignments, while the CP or TSP solvers sequence the jobs on machines. The new model is compared to an existing MIP model which finds the optimal solution.

2 Background

In this section, we will define the PMSP with sequence and machine dependent setup times. A review of related work is presented and finally, an existing MIP model to solve the PMSP is shown.

2.1 Problem Definition

In the PMSP, a set of N jobs are to be scheduled on M machines with the objective of minimizing the makespan. Each job has a processing time p_{ij}, the time to process job j on machine i. The machines in this system are unrelated, i.e., a job j can have a processing time greater than k on one machine, but the reverse could be true on another machine. There is a sequence and machine dependent setup time, s_{ijk}, which is the time that must elapse before a machine can begin processing job k if job j precedes it on machine i. The setup times are assumed to follow the triangle inequality $s_{ijk} \leq s_{ilk} + s_{ljk}$ and are only incurred when switching

from one job to another. The goal of the problem is to determine how to assign jobs to machines and then sequence these jobs, independently on each machine.

2.2 Related Work

Most research has been focused on the PMSP with identical machines (Graves 1981; Cheng and Sin 1990; Dunstall and Wirth 2005; Kurz and Askin 2001). Research on the PMSP with unrelated machines has concentrated on the problem without setup times. An exact algorithm was developed by Lancia (2000) to minimize makespan. A genetic algorithm, simulated annealing, and tabu search were compared by Glass et al. (1994).

The PMSP with sequence and machine dependent setup times is strongly NP-Hard because the single machine scheduling problem with sequence dependent setup times (1|sds|C$_{max}$) is equivalent to a travelling salesman problem (TSP) (Baker 1974). Thus, the PMSP with setup times can be thought of as an allocation and routing problem where cities are allocated to salesmen who then must find their own tour. Even in the case where all machines are identical (P|sds|C$_{max}$), the problem is strongly NP-hard (França et al. 1996; Mendes et al. 2002). The combinatorial complexity of the PMSP has resulted in little research in exact optimization methods. A MIP model does exist (Guinet 1991) which obtains the optimal schedule for the PMSP with setup times with total completion time or total tardiness as objectives. However, this MIP model is only able to solve for small instances:9 jobs, 2 machines or 8 jobs, 4 machines. The sizes of these problems are not very practical for real life use where the number of jobs can be much larger.

Al-Salem (2004) developed a constructive heuristic named the partitioning heuristic to solve large instances of the PMSP with setup times. Helal et al. (2006) developed a tabu search to solve the same problem. Rabadi et al.(2006) presented a meta-heuristic titled Meta-RaPS to minimize makespan. An ant colony optimization method was implemented and shown to perform better then the partitioning heuristic, tabu search, and Meta-RaPS for the unrelated PMSP with setup times (Arnaout, Rabadi, and Musa 2008). In all these studies, optimal makespans were not the goal because the problem sizes were too large for current methods to find optimality. The performance of the heuristics was evaluated by comparison of the solutions they provided with lower bounds on the makespan for any instances with more than 10 jobs. All heuristics tested performance for instances of sizes up to 100 jobs and 10 machines; Rabadi et al. (2006) and Arnaout et al. (2008) tested problems of 120 jobs and 12 or 8 machines respectively.

Focacci et al. (2000) proposed a two phase algorithm based on CP to optimize both the makespan and the sum of setup times for a similar problem to the PMSP with sequence dependent setup times. In this paper, jobs consist of multiple activities and precedence constraints exist between these activities. In the first phase of the algorithm, a time limited, incomplete branch-and-bound method is used to find solutions with small makespan. The second phase minimizes the sum of setup times with the constraint that any schedule found must have a makespan equal to or less than the makespan found in the first phase. They run their model on problems of up to 16 jobs, each consisting of 12 activities, and 16 machines.

2.3 Mixed Integer Programming Model

A MIP model used to find optimal solutions for the unrelated PMSP with setup times is presented by various researchers (Helal, Rabadi, and Al-Salem 2006; Rabadi, Moraga, and Al-Salem 2006). This formulation is based on a similar problem by Guinet (1991) with different objectives of total completion time or total tardiness.

$$\min \quad C_{max}$$

$$\text{s.t.} \quad \sum_{j=0,j\neq k}^{N}\sum_{i=1}^{M} x_{ijk} = 1, \qquad k \in N \qquad (1)$$

$$\sum_{j=0,j\neq h}^{N} x_{ijh} = \sum_{k=0,k\neq h}^{N} x_{ihk}, \quad h \in N, i \in M \qquad (2)$$

$$C_k \geq C_j + \sum_{i=1}^{M} x_{ijk}(s_{ijk} + p_{ik}) + V(\sum_{i=1}^{M} x_{ijk} - 1) \quad j \in N, k \in N \qquad (3)$$

$$\sum_{j=0}^{N} x_{i0j} = 1, \qquad i \in M \qquad (4)$$

$$C_j \leq C_{max}, \qquad j \in N \qquad (5)$$

$$C_0 = 0 \qquad (6)$$

$$C_j \geq 0, \qquad j \in N \qquad (7)$$

$$x_{ijk} \in (0;1), \qquad j \in N, \; k \in N, i \in M \qquad (8)$$

where

C_{max}:	Maximum completion time (makespan)
C_j:	Completion time of job j
x_{ijk}:	1 if job k is processed directly after job j on machine i
x_{i0k}:	1 if job k is the first job to be processed on machine i
x_{ij0}:	1 if job j is the last job to be processed on machine i
s_{i0k}:	setup time before job k if it is the first job on machine i
V:	A large positive number

Constraint (1) ensures that each job is scheduled on a single machine only and after exactly one other job. Constraint (2) ensures that each job cannot be preceded or succeeded by more than one job. Constraint (3) sets the completion times of each job such that if job j precedes job k, job k cannot also precede job j to create an infeasible cycle. If job k is processed directly after job j, $\sum_{i=1}^{M} x_{ijk} - 1 = 0$ and

the constraint makes it so that $C_k \geq C_j + s_{ijk} + p_{ik}$ with i being the machine on which the two jobs are assigned. If job k is not scheduled directly after job j on a machine, $\sum_{i=1}^{M} x_{ijk} - 1 = -1$ and the large V term makes the constraint redundant. Constraint (4) guarantees that only one job can be scheduled first on each machine. Constraint (5) sets the makespan to be at least as large as the largest completion time of all jobs. Constraint (6) sets the completion time of job 0, an auxiliary job used to enforce the start of a schedule, to zero and constraint (7) ensures positive completion times. Constraint (8) defines the decision variables as binary.

3 Logic-Based Benders Decomposition

The PMSP with setup times can be decomposed into an assignment master problem and sequencing subproblem. In the assignment master problem, the jobs are assigned to machines. This assignment results in multiple subproblems where each machine is a scheduling problem to sequence the assigned jobs. A MIP model is presented for the master problem. The use of MIP takes advantage of operations research tools to obtain tight lower bounds on the PMSP with setup times. Sequencing is accomplished in the subproblem where CP and TSP solvers are more adept to obtaining answers quickly.

3.1 Assignment Master Problem

The MIP formulation of the master problem is a relaxation of the PMSP with setup times. In this relaxation, jobs are assigned to machines, but instead of solving for a single sequence of jobs on each machine, many smaller subsequences are allowed. The setup times are calculated for each subsequence; their sum is a lower bound on the actual total setup time on a machine. This assignment and subsequencing leads to an infeasible schedule if multiple subsequences are created. However, the relaxation gives a tighter lower bound than if setup times are completely ignored, while being significantly less difficult than the full problem. The master problem is,

$$\min \quad C_{max}$$

$$\text{s.t.} \quad \sum_{j \in N} x_{ij} p_{ij} + \xi_i \leq C_{max} \qquad i \in M \qquad (9)$$

$$\sum_{i \in M} x_{ij} = 1 \qquad j \in N \qquad (10)$$

$$\xi_i = \sum_{j \in N} \sum_{k \in N, k \neq j} y_{ijk} s_{ijk} \qquad i \in M \qquad (11)$$

$$x_{ik} = \sum_{j \in N} y_{ijk} \qquad k \in N; i \in M \qquad (12)$$

$$x_{ij} = \sum_{k \in n} y_{ijk} \qquad j \in N; i \in M \qquad (13)$$

$$cuts \qquad (14)$$

$$x_{ij} \in \{0; 1\} \qquad j \in N; i \in M \qquad (15)$$

$$0 \leq y_{ijk} \leq 1 \qquad j, k \in N; i \in M \qquad (16)$$

where

C_{max}:	Makespan of the master problem
ξ_i:	Total setup time incurred from all sequences on machine i
x_{ij}:	1 if job j is processed on machine i
y_{ijk}:	1 if job k is processed directly after job j on machine i
y_{i0k}:	1 if job k is processed first on machine i
y_{ij0}:	1 if job j is process last on machine i

The makespan on each machine with the relaxed setup times is defined in constraint (9) as the summation of processing times for all jobs that are assigned to that machine and the relaxed total setup times. Setting the makespan to be greater than or equal to the relaxed makespan of each machine enforces that the MIP model optimizes the maximum makespan across all machines. Constraint (10) ensures that each job is assigned to exactly one machine. Constraint (11) assigns the relaxed setup time of a machine i, ξ_i, to be a lower bound on the additional time required from the sequencing of jobs, y_{ijk}, and their respective setup times, s_{ijk}. The relaxation of setup times allows, instead of a sequence of jobs from the first to last job processed on a machine, many smaller sequences independent of each other. For example, given jobs j, k, j_3, j_4, and j_5, a feasible sequence is [*start* - j - k - j_3 - j_4 - j_5 - *end*]. However, (12) and (13) set each job to have exactly one other job scheduled directly before and after it without the restriction of cycles as was seen in constraint (3). This will make it possible to assign two sequences, [*start* - j - k - j_3 - *end*] and the cycle [j_4 - j_5 - j_4 - j_5...]. Constraint (14) are *cuts* added to the master problem from the subproblem each time an infeasible solution is found. In the first iteration of the master problem, the set of *cuts* is empty. The last constraints, (15) and (16), force the decision variables x_{ij} to be binary, i.e., either a job j is assigned machine i or not and y_{ijk} to be between 0 and 1.

This formulation is equivalent to solving the (R|sds|Cmax), but instead of solving for the exact single sequence of jobs to process on a machine, many subsequences are allowed which will include all jobs. This relaxation creates a tight lower bound for the actual makespan of a machine and is similar to solving the assignment problem. Therefore, the makespan found from solving the master problem may be infeasible given a proper sequencing of jobs.

3.2 Sequencing Subproblem

Once a solution of the master problem is found, the set of jobs to schedule on each machine is known. These sets of jobs create m subproblems, one for each machine. In this section, two different subproblem formulations are presented: a CP and a TSP model. Both models will create a sequence of jobs on a machine such that the makespan is

minimized. This objective can also be thought of as minimizing the sum of setup times since the set of jobs to be processed and their processing times are already determined.

Constraint Program Let t_j and e_j be the start and end times for job j respectively. C^{hi}_{max} represents the makespan for machine i in iteration h. The CP formulation of the subproblem is shown below.

$$\min \quad C^{hi}_{max}$$

$$\text{s.t.} \quad t_j + p_{ij} = e_j \qquad j \in N' \qquad (17)$$

$$t_j \geq e_k + s_{ijk} \vee t_k \geq e_j + s_{jk} \qquad k, j \in N'; k \neq j \qquad (18)$$

$$t_0 = 0 \qquad (19)$$

$$\text{disjunctive}(t_j, p_{ij}) \qquad (20)$$

The objective of the subproblem is to sequence the jobs in such a way as to minimize the makespan. The set of jobs to schedule is N' which are the jobs chosen in the master problem with an additional auxiliary job (job 0) which has setup times and processing times equal to 0. This extra job acts as the first job in the sequence which incurs no setup times for the job that is scheduled after it. Constraint (17) sets the end time of each job to be the start time plus the processing time. Constraint (18) ensures that setup times are adhered to. If a job k is processed directly after a job j, then constraint (18) becomes an equality constraint. In the case where job k is not directly processed after a job j, but still processed later, the constraint holds given that setup times adhere to the triangle inequality. Constraint (19) sets the start time of job 0 to zero. Finally, constraint (20) is a global constraint that ensures the unary resource constraint is satisfied.

We implement this model in IBM ILOG Scheduler. Each job is an activity in IBM ILOG Scheduler, represented by the variables t_j and e_j. The machine that these jobs are assigned to is represented by a unary resource and the setup times, s_{jk}, are assigned as the elements of an IloTransitionParam matrix. The IloTransitionParam matrix is an asymmetric square matrix that defines the sequence dependent setup times in IBM ILOG Scheduler for each activity pair.

Travelling Salesman Problem We know that the sequencing of jobs on a single machine is equivalent to a TSP with directed edges, also known as an asymmetric TSP. In this TSP, jobs are the nodes and distances between nodes are the setup time between the two connected jobs and the processing time of the job which is the start node. This representation ensures that travelling along any edge from node a to node b will contribute the cost of processing job a and the setup time from job a to job b. With this translation, it is possible to see that the setup time problem on a single machine is equivalent to a TSP and a cycle of a TSP from a start node to all other nodes and back to the initial start node is the sequence of jobs to process and the distance travelled being the makespan. This representation is shown in Figure 1.

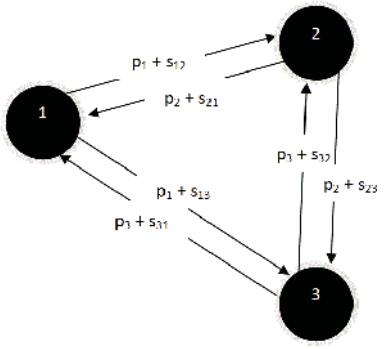

Figure 1: TSP representation

It can be seen that if the order of jobs to be processed is 1, 3 then 2, the distance travelled would be, $p_1 + s_{13} + p_3 + s_{32} + p_2$. This tour distance is equal to the makespan of processing jobs in that order. Therefore, it may be better to use a TSP solver which has been optimized to solve such problems in place of a generic CP solver which is more expressive, but not as good at solving this problem.

Feasible Schedules from the Subproblem Solving the sequencing subproblem leads to a feasible schedule. Often with Benders decomposition, a model is searching in the infeasible region of solutions and the first feasible solution found is the optimal one. In our logic-based Benders decomposition approach, while the search is performed in the infeasible region, the sequencing subproblem solves the local schedule once the jobs are allocated to machines. The solution from the subproblem creates a feasible schedule with a makespan equal to the largest makespan found across all subproblems. Therefore, it is possible to stop the solving at any time after the first complete iteration and obtain a feasible schedule. This schedule may not be optimal if the problem solving is stopped prematurely. However, it is possible to compare this value against the makespan found from the most recent master problem solution to calculate an upper bound on how far the current schedule is from optimality. Therefore, the Benders decomposition will store the best solution found so far. At the completion of all subproblems during an iteration, the schedule created will be compared to the best solution found so far and it will be updated if necessary.

3.3 Cuts

If the makespan found in the subproblem is less than or equal to the master problem's makespan C_{max}, then this subproblem is feasible and no cuts are added to the master problem. In the case where the makespan found is greater than C_{max}, a cut is created and sent to the master problem. The master problem is then re-solved with the added cut. The cut from such a subproblem in an iteration h is,

$$C_{max} \geq C^{hi*}_{max}[1 - \sum_{j \in N'}(1 - x_{ij})] \quad i \in M \quad (C1)$$

Here, C_{max} is the makespan variable in the master problem and C^{hi*}_{max} is the makespan found in iteration h when solving the subproblem for machine i. The cut states that the future solutions of the master problem can only decrease the makespan if another assignment of jobs is given. That is, if the same assignment is given to the subproblem, the x_{ij} variables that are part of this cut will all equal to 1. If this is the case, then $(1 - x_{ij}) = 0$ for all j and the makespan of the subproblem becomes a lower bound on C_{max}. When a different assignment is made and at least one of the x_{ij} variables that previously had a value of 1 is 0, the cut becomes redundant as the right hand side will be at most zero. This cut follows the 2 conditions defined by Chu and Xia (2005) to be a valid cut; the cut removes the current solution from the master problem and does not eliminate any global optimal solutions.

The cut presented is a type of *nogood* cut (Hooker 2005), stating that the current solution is infeasible and so is removed from the search space. We can tighten the cut by introducing the values minPre_{hj}, minSuc_{hj}, maxPre_{hj}, maxSuc_{hj} and $\mathit{minTran}_{hj}$. minPre_{hj} and maxPre_{hj} are the minimum and maximum setup times if job j directly succeeds another job that is assigned to the same machine in iteration h respectively. Similarly, minSuc_{hj} and maxSuc_{hj} are the minimum and maximum setup times if job j directly precedes another job that is assigned to the same machine in iteration h. Finally, $\mathit{minTran}_{hj}$ is the minimum setup time between any two jobs in N' excluding job j. These values are defined as,

$$\begin{aligned}
\mathit{minPre}_{hj} &= MIN_{k \in N', k \neq j}(s_{ikj}) \\
\mathit{minSuc}_{hj} &= MIN_{k \in N', k \neq j}(s_{ijk}) \\
\mathit{maxPre}_{hj} &= MAX_{k \in N', k \neq j}(s_{ikj}) \\
\mathit{maxSuc}_{hj} &= MAX_{k \in N', k \neq j}(s_{ikj}) \\
\mathit{minTran}_{hj} &= MIN_{k \in N', l \in N', k \neq l}(s_{ikl})
\end{aligned}$$

A cut that uses more information is to find a tight upper bound on the total reduction of the makespan when a job is removed from a schedule. To find this upper bound, we work backwards and calculate the increase in the makespan if job j is added to an optimal sequence consisting of the jobs in N' except j. Assume that the optimal makespan of the schedule, C', is known. If job j is inserted into the schedule greedily, the insertion may occur at either a position that minimizes the setup time to job j or from job j. In the worst case scenario, if the insertion that occurred to minimize the setup to (from) job j, the job that succeeds (precedes) j may result in the largest setup time from (to) job j. This insertion will further remove a single setup time since job j may be scheduled between any two other jobs. In the worst case, the removed setup time is the minimum setup time between any two jobs in N'. We know that this insertion results in a feasible schedule and in the best case will give the optimal makespan for scheduling N'. The increase in makespan of adding job j is denoted as δ_{hij}. We know that,

$$C^{hi*}_{max} \leq C' + \delta_{hij}$$

which rearranged is,

$$C' \geq C^{hi*}_{max} - \delta_{hij}$$

Therefore, if the optimal makespan of a schedule consisting of all jobs in N' is known, removing δ_{hij} from the optimal makespan is guaranteed to result in a makespan less than or equal optimal if job j is removed. The cut can then be,

$$C_{max} \geq C^{hi*}_{max} - \sum_{j \in N'}(1 - x_{ij})\delta_{hij} \quad i \in M \quad (C2)$$

where,

$$\begin{aligned}
\delta_{hij} = \quad & p_{ij} - \mathit{minTran}_{hj} \\
& + MIN[(\mathit{minPre}_{hj} + \mathit{maxSuc}_{hj}), \\
& \qquad (\mathit{maxPre}_{hj} + \mathit{minSuc}_{hj})]
\end{aligned}$$

A third cut is developed to further improve upon (C2). This cut attempts to not only include extra information about the jobs being assigned, but also account for the jobs that are not assigned to the subproblem machine. This is done by incrementing the makespan by the processing time of a job if that job is to be assigned to a machine. The cut is then,

$$\begin{aligned}
C_{max} \geq\ & C^{hi*}_{max} + \textstyle\sum_{k \notin N'} x_{ik} p_{ik} \\
& - \textstyle\sum_{j \in N'}(1 - x_{ij})\delta_{hij} \qquad i \in M \quad (C3)
\end{aligned}$$

3.4 Stopping Condition

The Benders approach will iterate between master problem and subproblems until an optimal solution is found and proved. Optimality is proven if one of two conditions is met. The first condition that can prove optimality is if all subproblems solved during an iteration find makespans less than or equal to the C_{max} from the master problem. This solution is optimal because the master problem provides a lower bound on the achievable makespan of the problem. If all subproblems prove that a schedule can be created with the makespan less than or equal to C_{max} of the master problem, it is proven that the schedule is optimal. The second condition that can prove optimality and provide a stopping condition requires that the C_{max} found from the master problem be equal to the best feasible makespan found so far as defined in Section 3.2.

4 Computational Results

The Benders decomposition model and MIP model were tested on an Intel Pentium 4 CPU 3.00GHz Hypterthread Tech with 2 MB cache per core, 1 GB of main memory, running Red Hat 3.4.6-3. The MIP master problem and MIP model were implemented with IBM ILOG CPLEX 12.1 and the CP subproblem was implemented with IBM ILOG Solver 6.7 and IBM ILOG Scheduler 6.7. The TSP solver used was tsp_solve.[1] Experiments were run for problem instances of 10, 20, 30, and 40 jobs. For each job size, between 2 and 5 machines were tested. Each of these combinations

[1] A free TSP solver available online at (http://www.or.deis.unibo.it/research_pages/tspsoft.html) which is written in C++.

had a total of 10 instances for a total of 160 instances. A time limit of 3 hours was used. Processing times for each machine job pair were generated from a uniform distribution between 1 and 100. To obtain setup times that were sequence dependent and follow the triangular inequality assumption, each job was given two different sets of coordinates on a Cartesian plane for every machine. The setup times are the Manhattan distances from one job's coordinates to the other's. Distances between the second set of coordinates is used to provide asymmetric setup times. For example, job 0 and job 1 would be given coordinates χ_{0a}, χ_{0b}, ψ_{0a}, ψ_{0b}, χ_{1a}, χ_{1b}, ψ_{1a}, and ψ_{1b}. Setup time from job 0 (1) to job 1 (0) would then be $|\chi_{0a} - \chi_{1a}| + |\psi_{0a} - \psi_{1a}|$ ($|\chi_{0b} - \chi_{1b}| + |\psi_{0b} - \psi_{1b}|$).

Table 1 shows results comparing the MIP model, CP Benders, and TSP Benders. In both Benders decomposition models, cut (C2) was used. For these results, the time until an optimal solution was found and proved were recorded. Where the solving timed out, 3 hours was used.

The Benders decomposition model's results, both CP and TSP versions, are a significant improvement over the MIP performance. It is clear that the Benders decomposition approach is capable of solving much larger problems in significantly shorter run times. The Benders decomposition approach, with a CP solver, solves up to 20 jobs in the time limit and the majority of instances of up to 30 jobs. Replacing the CP solver with a TSP solver, the Benders model is able to increase the number of solvable jobs up to 30 consistently. This is in contrast to the MIP model which is able to solve only 10 jobs in the 3 hour time limit.

We see that increasing the number of machines has a greater effect on the performance of the models than increasing the number of jobs. In both Benders models, the master problem had difficulty solving for increased machines. In fact, the TSP subproblem is able to solve each subproblem in milliseconds while the MIP master problem can spend hours searching for an assignment. The opposite is seen for the MIP model. The MIP model has difficulties sequencing large number of jobs on machines and so, in the case where there are only 2 machines, the MIP model has a very high runtime for instances of 10 jobs and 2 machines. When the number of machines is increased to 5, the sequencing problem is simpler and results in fast runtimes.

Using the Benders decomposition model with the TSP subproblem, the three different cuts presented in Section 3.3 are tested on the same set of instances for problem sizes of 10, 20, and 30 jobs. The results are presented in table 2.

Table 2 shows that the performance of cut (C2) is the best overall. In most cases, the difference is not significant, but there are cases where clear differences are found. Specifically, (C2) is able to, in a small number of instances, create a cut that removes an assignment that (C1) would not. This improved cut reduces the total number of iterations required and the time needed to solve the problem. In the 20 jobs and 5 machines test case, the average number of iterations for (C1) is 4.4 while cut (C2) was able to decrease this value to 4. This resulted in a 30 second runtime difference on average to reduce the runtime by about 25%.

One would then assume that cut (C3), using more information, would create better cuts. However, the results showed that cut (C3) performs worst than cut (C2). This is because cut (C3) created a more difficult master problem to solve increasing the total solve time per iteration, while not becoming much tighter than (C2) to reduce the overall number of iterations. The test case with 20 jobs and 5 machines shows how (C3) only reduces the number of iterations by a very small amount, a further 0.2 iterations from cut (C2), but increases on average by 2 seconds of runtime. Including into the cut all jobs that were not assigned to a machine accounts for more information, but proves detrimental to the performance of the Benders model.

In the 30 jobs and 4 machines case, we even see cut (C3) increasing the number of iterations over cut (C2). This seems to contradict the fact that (C3) is a tighter cut. The increase in iteration occurs because some degeneracy exists in the problem. Varying the cut may lead to different assignments in the Benders master problem with equal objective functions. In rare instances, it is possible that the other cuts will lead to the optimal assignment while (C3) leads to an equivalent master problem that does not extend to a global solution.

5 Future Work

Though the Benders decomposition approach obtains significant speed ups, the problem sizes that can be solved are limited compared to what heuristic models are currently solving. For larger problems, the Benders decomposition is not able to complete a single instance of the master problem. Problem instances of 100 jobs are tested in previous papers using heuristics and local search (Helal, Rabadi, and Al-Salem 2006; Rabadi, Moraga, and Al-Salem 2006). Specifically, in work done by Helal et al. (2006), schedules for problem instances of 100 jobs and 10 machines are found within minutes. The quality of these schedules is difficult to assess, given that the optimal solutions are not known. However, for smaller instances (8 jobs and 4 machines) the optimal solution was known from the MIP model and the heuristic used was experimentally shown to have on average 2.5% deviation from optimal.

If optimal solutions are not possible because of large instances, it is clear that heuristic solutions are necessary. Stopping the Benders program early and using the best found schedule as shown in Section 3.2 is one approach to increase the size of problems for which the Benders model can obtain schedules. However, this approach is only useful if the problems are small enough such that the Benders model can solve for one complete iteration in a reasonable time. From our experiments, we found some instances of 40 jobs and 5 machines where solving for one complete iteration took more than one hour. Solve times of the subproblem are almost instantaneous when the TSP solver is used on all instances, but the MIP master problem may be intractable once the size of the problem reaches 50 or more jobs and 5 machines. This means that for problems as large as 100 jobs, the Benders decomposition model is not likely to solve the master problem within the time limit.

Therefore, we plan to investigate not solving the master problem all the way to optimality. This would reduce

		MIP		CP Benders		TSP Benders	
n	m	Avg Runtime	# uns.	Avg Runtime	# uns.	Avg Runtime	# uns.
10	2	9885.14	9	0.24	0	0.12	0
	3	7924.16	7	0.29	0	0.22	0
	4	1169.69	1	0.55	0	0.42	0
	5	13.59	0	0.56	0	0.45	0
20	2	10800.00	10	9.79	0	1.08	0
	3	10800.00	10	6.41	0	2.02	0
	4	10800.00	10	239.15	0	53.12	0
	5	10800.00	10	509.85	0	122.46	0
30	2	10800.00	10	7997.13	5	2.15	0
	3	10800.00	10	3244.75	1	27.85	0
	4	10800.00	10	10041.10	2	203.13	0
	5	10800.00	10	8804.01	5	750.89	0
40	2	10800.00	10	10800.00	10	4.78	0
	3	10800.00	10	10800.00	10	651.30	0
	4	10800.00	10	10800.00	10	1538.69	0
	5	10800.00	10	10800.00	10	7404.07	5

Table 1: CPU runtime in seconds. Comparison of MIP, CP Benders, and TSP Benders.

		(C1)		(C2)		(C3)	
n	m	Avg Runtime	Avg # of Iter	Avg Runtime	Avg # of Iter	Avg Runtime	Avg # of Iter
10	2	0.24	2.1	0.12	2.1	0.12	2.1
	3	0.25	1.9	0.22	1.9	0.26	1.9
	4	0.42	1.8	0.42	1.8	0.40	1.7
	5	0.56	1.5	0.45	1.4	0.47	1.4
20	2	1.15	4.5	1.08	4.5	1.09	4.5
	3	2.06	2.7	2.02	2.7	2.05	2.7
	4	56.87	3.9	53.12	3.9	70.06	3.9
	5	153.33	4.4	122.46	4.0	124.30	3.8
30	2	2.17	4.4	2.15	4.4	2.28	4.4
	3	29.31	5.4	27.85	5.2	30.52	5.2
	4	224.67	4.9	203.13	4.5	222.83	4.7
	5	784.39	5.6	750.89	5.3	817.22	5.3

Table 2: CPU runtime in seconds. Comparison of different cuts on the TSP-Benders model.

the effort required in the master problem at producing assignments and enable the model to generate feasible schedules faster. Whether the method chosen is to allow the MIP model to solve with an optimality gap tolerance in mind or to solve the master problem through some other heuristic is to be determined.

The change to the Benders decomposition master problem means that the master problem would no longer act as a true lower bound. However, if the optimality gap suggestion is the method of choice, it could be guaranteed that the solution found would be within that chosen gap of optimal. Experimental results from large instances show that the master problem often spends most of time proving optimality or obtaining optimality with a gap of less than 2%. If the master problem was allowed to solve until it has proven it is within 2% of optimal, the Benders approach would be able to obtain solutions much faster. What is of concern however, is whether prematurely solving the master problem with this gap is detrimental to the quality of a solution if optimality is not hard to find or prove for some problems.

Another extension to this work is to test the performance of branch-and-check (Thorsteinsson 2001) on the PMSP with setup times. Beck (2010) identified that using branch-and-check, on problems with proportionately larger master problems in comparison to subproblems, is better than logic-based Benders. The Benders decomposition could benefit from solving the subproblem more often and create feasible schedules earlier.

6 Conclusion

In this paper, we presented a logic-based Benders decomposition approach to minimize the makespan of an unrelated parallel machine scheduling problem with sequence and machine dependent setup times. A MIP model was defined to solve for the assignment of jobs to machines and produce

a lower bound on the achievable makespan of the problem. Two subproblems, based on a CP and TSP solver, were implemented to find optimal schedules for the assignment of jobs on each individual machine. The computational results show that the cooperation of MIP and TSP can effectively find optimal solutions. We are able to solve instances four times larger than what was previously possible using a MIP formulation found in the literature and obtain optimal solutions on problems of the same size up to five orders of magnitude faster. Although finding optimal solutions was the goal of the paper, possible heuristics based on the Benders decomposition approach are proposed to solve larger instances.

References

Al-Salem, A. 2004. Scheduling to minimize makespan on unrelated parallel machines with sequence dependent setup times. *Engineering Journal of the University of Qatar* 17:177–187.

Allahverdi, A.; Gupta, J.; and Aldowaisan, T. 1999. A review of scheduling research involving setup considerations. *Omega* 27(2):219–239.

Arnaout, J.; Rabadi, G.; and Musa, R. 2008. A two-stage ant colony optimization algorithm to minimize the makespan on unrelated parallel machines with sequence-dependent setup times. *Journal of Intelligent Manufacturing* 1–9.

Baker, K. 1974. *Introduction to sequencing and scheduling*. John Wiley & Sons.

Beck, J. 2010. Checking-Up on Branch-and-Check. In *Proceeings of the Sixteenth International Conference of Principles and Practice of Constraint Programming (CP2010)*, 84–98. Springer.

Cheng, T., and Sin, C. 1990. A state-of-the-art review of parallel-machine scheduling research. *European Journal of Operational Research* 47(3):271–292.

Chu, Y., and Xia, Q. 2005. A hybrid algorithm for a class of resource constrained scheduling problems. In *Proceedings of the 2nd Conference on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, 110–124. Springer.

Dunstall, S., and Wirth, A. 2005. Heuristic methods for the identical parallel machine flowtime problem with set-up times. *Computers & Operations Research* 32(9):2479–2491.

Focacci, F.; Laborie, P.; and Nuijten, W. 2000. Solving scheduling problems with setup times and alternative resources. In *Proceedings of the 5th International Conference on Artificial Intelligence Planning and Scheduling*.

França, P.; Gendreau, M.; Laporte, G.; and Muller, F. 1996. A tabu search heuristic for the multiprocessor scheduling problem with sequence dependent setup times. *International Journal of Production Economics* 43(2-3):79–89.

Glass, C.; Potts, C.; and Shade, P. 1994. Unrelated parallel machine scheduling using local search. *Mathematical and Computer Modelling* 20(2):41–52.

Graham, R.; Lawler, E.; Lenstra, J.; and Kan, A. 1979. Optimization and approximation in deterministic sequencing and scheduling: A survey. *Annals of Discrete Mathematics* 5(2):287–326.

Graves, S. 1981. A review of production scheduling. *Operations Research* 29(4):646–675.

Guinet, A. 1991. Textile production systems: a succession of non-identical parallel processor shops. *The Journal of the Operational Research Society* 42(8):655–671.

Helal, M.; Rabadi, G.; and Al-Salem, A. 2006. A tabu search algorithm to minimize the makespan for the unrelated parallel machines scheduling problem with setup times. *International Journal of Operations Research* 3(3):182–192.

Hooker, J., and Ottosson, G. 2003. Logic-based Benders decomposition. *Mathematical Programming* 96(1):33–60.

Hooker, J. 1995. Verifying logic circuits by Benders decomposition. *Principles and Practice of Constraint Programming: The Newport Papers, MIT Press, Cambridge, MA* 267–288.

Hooker, J. 2005. A hybrid method for the planning and scheduling. *Constraints* 10(4):385–401.

Kurz, M., and Askin, R. 2001. Heuristic scheduling of parallel machines with sequence-dependent set-up times. *International Journal of Production Research* 39(16):3747–3769.

Lancia, G. 2000. Scheduling jobs with release dates and tails on two unrelated parallel machines to minimize the makespan. *European Journal of Operational Research* 120(2):277–288.

Mendes, A.; Muller, F.; França, P.; and Moscato, P. 2002. Comparing meta-heuristic approaches for parallel machine scheduling problems. *Production Planning & Control* 13(2):143–154.

Rabadi, G.; Moraga, R.; and Al-Salem, A. 2006. Heuristics for the unrelated parallel machine scheduling problem with setup times. *Journal of Intelligent Manufacturing* 17(1):85–97.

Thorsteinsson, E. 2001. Branch-and-check: A hybrid framework integrating mixed integer programming and constraint logic programming. In *Proceedings of the Seventh International Conference on Principles and Practice of Constraint Programming (CP2001)*, 16–30. Springer.

Applying Iterative Flattening Search to the Job Shop Scheduling Problem with Alternative Resources and Sequence Dependent Setup Times

Angelo Oddi [1] and **Riccardo Rasconi** [1] and **Amedeo Cesta** [1] and **Stephen F. Smith** [2]
[1] Institute of Cognitive Science and Technology, CNR, Rome, Italy
{*angelo.oddi, riccardo.rasconi, amedeo.cesta*}*@istc.cnr.it*
[2] Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, USA
sfs@cs.cmu.edu

Abstract

This paper tackles a complex version of the Job Shop Scheduling Problem (JSSP) that involves both the possibility to select alternative resources to activities and the presence of sequence dependent setup times. The proposed solving strategy is a variant of the known Iterative Flattening Search (IFS) metaheuristic. This work presents the following contributions: (1) a new constraint-based solving procedure produced by means of enhancing a previous JSSP-solving version of the same metaheuristic; (2) a new version of both the variable and value ordering heuristics, based on temporal flexibility, that capture the relevant features of the extended scheduling problem (i.e., the flexibility in the assignment of resources to activities, and the sequence dependent setup times); (3) a new relaxation strategy based on the random selection of the activities that are closer to the critical path of the solution, as opposed to the original approach based on a fully random relaxation. The performance of the proposed algorithm are tested on a new benchmark set produced as an extension of an existing well-known testset for the Flexible Job Shop Scheduling Problem by adding sequence dependent setup times to each original testset's instance, and the behavior of the old and new relaxation strategies are compared.

Introduction

This paper describes an iterative improvement approach to solve job-shop scheduling problems involving both *sequence dependent* setup-times and the possibility of selecting *alternative routes* among the available machines. Over the last years there has been an increasing interest in solving scheduling problems involving both setup-times and flexible shop environments (Allahverdi and Soroush 2008; Allahverdi et al. 2008). This fact stems mainly from the observation that in various real-word industry or service environments there are tremendous savings when setup times are explicitly considered in scheduling decisions. In addition, the possibility of selecting alternative routes among the available machines is motivated by interest in developing Flexible Manufacturing Systems (FMS) (Sethi and Sethi 1990) able to use multiple machines to perform the same operation on a job's part, as well as to absorb large-scale changes, such as in volume, capacity, or capability.

The proposed problem, called in the rest of the paper Flexible Job Shop Scheduling Problem with Sequence Dependent Setup Times (SDST-FJSSP) is a generalization of the classical Job Shop Scheduling Problem (JSSP) where each activity requires a single machine and there are no setup-times. This problem is more difficult than the classical JSSP (which is itself *NP-hard*), since it is not just a sequencing problem; in addition to deciding how to sequence activities that require the same machine (involving sequence-dependent setup-times), it is also necessary to choose a *routing policy*, i.e., deciding which machine will process each activity. The objective remains that of minimizing *makespan*.

Despite this problem is often met in real manufacturing systems, not many papers consider both sequence dependent setup-times in flexible job-shop environments. On the other hand, a richer literature is available when setup-times and flexible job-shop environments are considered separately. In particular, on the side of setup-times a first reference work is (Brucker and Thiele 1996), which relies on an earlier proposal presented in (Brucker, Jurisch, and Sievers 1994). More recent works are (Vela, Varela, and González 2009) and (González, Vela, and Varela 2009), which propose effective heuristic procedures based on genetic algorithms and local search. In these works, the introduced local search procedures extend an approach originally proposed by (Nowicki and Smutnicki 2005) for the classical job-shop scheduling problem to the setup times case. A last noteworthy work is (Balas, Simonetti, and Vazacopoulos 2008), which extends the well-known *shifting bottleneck* procedure (Adams, Balas, and Zawack 1988) to the setup-time case. Both (Balas, Simonetti, and Vazacopoulos 2008) and (Vela, Varela, and González 2009) have produced reference results on a previously studied benchmark set of JSSP with sequence dependent setup-times problems initially proposed by (Brucker and Thiele 1996). About the Flexible Job Shop Scheduling FJSSP an effective synthesis of the existing solving approaches is proposed in (Hmida et al. 2010). The core set of procedures which generate the best results include the genetic algorithm (GA) proposed in (Gao, Sun, and Gen 2008), the tabu search (TS) approach of (Mastrolilli and Gambardella 2000) and the discrepancy-based method, called climbing depth-bound discrepancy search (CDDS), defined in (Hmida et al. 2010). Among the papers dealing with both sequence dependent setup times and flexible shop environments there is the work (Rossi and Dini 2007), which considers a shop type composed of pools of identical ma-

chines as well as two types of setup times: one modeling the transportation times between different machines (sequence dependent) and the other one modeling the required reconfiguration times (not sequence dependent) on the machines. The other work (Ruiz and Maroto 2006) considers a flow-shop environment with multi-purpose machines, such that each stage of a job can be processed by a set of unrelated machines (the processing times of the jobs depend on the machine they are assigned to). (Vallada and Ruiz 2011) considers a problem similar to the previous one, where the jobs are composed by a single step, but setup-times are both sequence and machine dependent. Finally, (Valls, Perez, and Quintanilla 1998) considers a job-shop problem with parallel identical machines, release times and due dates but sequence independent setup-times.

This paper focuses on a family of solving techniques referred to as Iterative Flattening Search (IFS). IFS was first introduced in (Cesta, Oddi, and Smith 2000) as a scalable procedure for solving multi-capacity scheduling problems. IFS is an iterative improvement heuristic designed to minimize schedule makespan. Given an initial solution, IFS iteratively applies two-steps: (1) a subset of solving decisions are randomly retracted from a current solution (*relaxation-step*); (2) a new solution is then incrementally recomputed (*flattening-step*). Extensions to the original IFS procedure were made in two subsequent works (Michel and Van Hentenryck 2004; Godard, Laborie, and Nuitjen 2005) and more recently (Oddi et al. 2010) have performed a systematic study aimed at evaluating the effectiveness of single *component strategies* within the same uniform software framework. The IFS variant that we propose relies on a *core* constraint-based search procedure as its solver. This procedure is an extension of the SP-PCP procedure proposed in (Oddi and Smith 1997). SP-PCP generates consistent orderings of activities requiring the same resource by imposing precedence constraints on a temporally feasible solution, using *variable* and *value* ordering heuristics that discriminate on the basis of temporal flexibility to guide the search. We extend both the procedure and these heuristics to take into account both sequence dependent setup-times and flexibility in machine selection. To provide a basis for embedding this core solver within an IFS optimization framework, we also specify an original relaxation strategy based on the idea of randomly breaking the execution orders of the activities on the machines with a activity selection criteria based on their *proximity* to the solution's critical path.

The paper is organized as follows. The first two sections define the SDST-FJSSP problem and its CSP representation. The main contribution of the work is given by two further sections which respectively describes the core constraint-based search procedure and the definition of the IFS meta-heuristic. An experimental section describes the performance of our algorithm on a set of benchmark problems and explains the most interesting results. Some conclusions end the paper.

The Scheduling Problem

The SDST-FJSSP entails synchronizing the use of a set of machines (or resources) $R = \{r_1, \ldots, r_m\}$ to perform a set of n activities $A = \{a_1, \ldots, a_n\}$ over time. The set of activities is partitioned into a set of nj jobs $\mathcal{J} = \{J_1, \ldots, J_{nj}\}$. The processing of a job J_k requires the execution of a strict sequence of n_k activities $a_i \in J_k$ and cannot be modified. All jobs are released at time 0. Each activity a_i requires the exclusive use of a *single resource* r_i for its entire duration chosen among a *set of available resources* $R_i \subseteq R$. No *preemption* is allowed. Each machine is available at time 0 and can process more than one operation of a given job J_k (*recirculation* is allowed). The processing time p_{ir} of each activity a_i depends on the selected machine $r \in R_i$, such that $e_i - s_i = p_{ir}$, where the variables s_i and e_i represent the start and end time of a_i. Moreover, for each resource r, the value st^r_{ij} represents the setup time between two generic activities a_i and a_j (a_j is scheduled immediately after a_i) requiring the same resource r, such that $e_i + st^r_{ij} \leq s_j$. As is traditionally assumed in the literature, the setup times st^r_{ij} satisfy the so-called *triangular inequality* (see (Brucker and Thiele 1996; Artigues and Feillet 2008)). The triangle inequality states that, for any three activities a_i, a_j, a_k requiring the same resource, the inequality $st^r_{ij} \leq st^r_{ik} + st^r_{kj}$ holds. A *solution* $S = \{(\overline{s_1}, \overline{r_1}), (\overline{s_2}, \overline{r_2}), \ldots, (\overline{s_n}, \overline{r_n})\}$ is a set of pairs $(\overline{s_i}, \overline{r_i})$, where $\overline{s_i}$ is the assigned start-time of a_i, $\overline{r_i}$ is the selected resource for a_i and all the above constraints are satisfied. Let C_k be the completion time for the job J_k, the *makespan* is the value $C_{max} = max_{1 \leq k \leq nj}\{C_k\}$. An *optimal* solution S^* is a solution S with the minimum value of C_{max}. The SDST-FJSSP is *NP-hard* since it is an extension of the JSSP problem (Garey and Johnson 1979).

A CSP Representation

There are different ways to model the problem as a *Constraint Satisfaction Problem* (CSP) (Montanari 1974), we use an approach similar to (Oddi and Smith 1997). In particular, we focus on *assigning resources to activities*, a distinguishing aspect of SDST-FJSSP and on *establishing sequence dependent setup time constraints* between pairs of activities that require the same resource, so as to eliminate all possible conflicts in the resource usage.

Let $G(A_G, J, X)$ be a graph where the set of vertices A_G contains all the activities of the problem together with two dummy activities, a_0 and a_{n+1}, respectively representing the beginning (reference) and the end (horizon) of the schedule. Each activity a_i is labelled with the set of available resource choices R_i. J is a set of directed edges (a_i, a_j) representing the precedence constraints among the activities (job precedences constraints) and are labelled with the set of processing times p_{ir} ($r \in R_i$) of the edge's source activity a_i. The set of undirected edges X represents the *disjunctive constraints* among the activities requiring the same resource r; there is an edge for each pair of activities a_i and a_j requiring the same resource r ($R_i = R_j = \{r\}$) and the related label represents the set of possible ordering between a_i and a_j: $a_i \preceq a_j$ or $a_j \preceq a_i$. Hence, in CSP terms, there are *two sets of decision variables*: (1) a variable x_i is defined for each activity a_i to select one resource for its execution, the domain of x_i is the set of available resource R_i: (2) A variable o_{ijr} is defined for each pair of activities a_i and a_j

requiring the same resource r ($x_i = x_j = r$), which can take one of two values $a_i \preceq a_j$ or $a_j \preceq a_i$. It is worth noting that in considering either ordering we have to take into account the presence of sequence dependent setup times, which must be included when an activity a_i is executed on the same resource *before* another activity a_j. As we will see in the next sections, the previous decisions for o_{ijr} can be represented as the following two temporal constraints: $e_i + st^r_{ij} \leq s_j$ (i.e. $a_i \preceq a_j$) or $e_j + st^r_{ji} \leq s_i$ (i.e. $a_j \preceq a_i$).

To support the search for a consistent assignment to the set of decision variables x_i and o_{ijr}, for any SDST-FJSSP we define the directed graph $G_d(V, E)$, called *distance graph*, which is an extended version of the graph $G(A_G, J, X)$. The set of nodes V represents time points, where tp_0 is the *origin* time point (the reference point of the problem), while for each activity a_i, s_i and e_i represent its start and end time points respectively. The set of edges E represents all the imposed temporal constraints, i.e., precedences and durations. In particular, for each activity a_i we impose the interval duration constraint $e_i - s_i \in [p^{min}_i, p^{max}_i]$, such that p^{min}_i (p^{max}_i) is the minimum (maximum) processing time according to the set of available resources R_i. Given two time points tp_i and tp_j, all the constraints have the form $a \leq tp_j - tp_i \leq b$, and for each constraint specified in the SDST-FJSSP instance there are two weighted edges in the graph $G_d(V, E)$; the first one is directed from tp_i to tp_j with weight b and the second one is directed from tp_j to tp_i with weight $-a$. The graph $G_d(V, E)$ corresponds to a *Simple Temporal Problem* (STP) and its consistency can be efficiently determined via shortest path computations; the problem is consistent if and only if no closed paths with negative length (or negative cycles) are contained in the graph G_d (Dechter, Meiri, and Pearl 1991). Thus, a search for a solution to a SDST-FJSSP instance *can proceed by repeatedly adding new precedence constraints into* $G_d(V, E)$ and recomputing shortest path lengths to confirm that $G_d(V, E)$ remains consistent.

A solution S is given as a affine graph $G_S(A_G, J, X_S)$, such that each undirected edge (a_i, a_j) in X is replaced with a directed edge representing one of the possible orderings between a_i and a_j: $a_i \preceq a_j$ or $a_j \preceq a_i$. In general the directed graph G_S represents a set of temporal solutions $(S_1, S_2, \ldots, S_n)$ that is, a set of assignments to the activities' start-times which are consistent with the set of imposed constraints X_S. Let $d(tp_i, tp_j)$ ($d(tp_j, tp_i)$) designate the shortest path length in graph $G_d(V, E)$ from node tp_i to node tp_j (from node tp_j to node tp_i); then, the constraint $-d(tp_j, tp_i) \leq tp_j - tp_i \leq d(tp_i, tp_j)$ is demonstrated to hold (Dechter, Meiri, and Pearl 1991). Hence, the interval $[lb_i, ub_i]$ of time values associated with a given time variable tp_i respect to the *reference* point tp_0 is computed on the graph G_d as the interval $[-d(tp_i, tp_0), d(tp_0, tp_i)]$. In particular, given a STP, the following two sets of value assignments $S_{lb} = \{-d(tp_1, tp_0), -d(tp_2, tp_0), \ldots, -d(tp_n, tp_0)\}$ and $S_{ub} = \{d(tp_0, tp_1), d(tp_0, tp_2), \ldots, d(tp_0, tp_n)\}$ to the STP variables tp_i represent the so-called *earliest-time solution* and *latest-time solution*, respectively.

Basic Constraint-based Search

The proposed procedure for solving instances of SDST-FJSSP integrates a Precedence Constraint Posting (PCP) one-shot search for generating sample solutions and an Iterative Flattening meta-heuristic that pursues optimization. The one-shot step, similarly to the SP-PCP scheduling procedure (Shortest Path-based Precedence Constraint Posting) proposed in (Oddi and Smith 1997), utilizes shortest path information in $G_d(V, E)$ to guide the search process. Shortest path information is used in a twofold fashion to enhance the search process: to propagate problem constraints and to define variable and value ordering heuristics.

Propagation Rules

The first way to exploit shortest path information is by introducing conditions to remove infeasible values from the domains of the decision variables x_i, representing the assignment of resources to activities. Namely, for each activity a_i we relax the disjunctive duration constraint into the interval constraint $e_i - s_i \in [p^{min}_i, p^{max}_i]$, such that p^{min}_i (p^{max}_i) is the minimum (maximum) processing time according to the set of available resources R_i (R_i is the domain of the decision variable x_i). As the search proceeds, as soon as the interval of distance between the start-time and the end-time of a_i $[-d(s_i, e_i), d(e_i, s_i)]$ is updated, the duration $p_{ir} \notin [-d(s_i, e_i), d(e_i, s_i)]$ is removed from the domain of x_i and a new interval $[p^{min}_i, p^{max}_i]$ is recomputed accordingly. In case of the domain of the decision variable x_i becomes *empty*, the search reaches a *failure* state.

The second way to exploit shortest path is by introducing new *Dominance Conditions* (which adapt those presented in (Oddi and Smith 1997) to the setup times case), through which problem constraints are *propagated* and mandatory decisions for promoting early pruning of alternatives are identified. The following concepts of $slack(e_i, s_j)$ and *co-slack*(e_i, s_j) (complementary slack) play a central role in the definition of such new dominance conditions. Given two activities a_i, a_j and the related interval of distances $[-d(s_j, e_i), d(e_i, s_j)]$ [1] and $[-d(s_i, e_j), d(e_j, s_i)]$ [2] on the graph G_d, they are defined as follows:

- $slack(e_i, s_j) = d(e_i, s_j) - st^r_{ij}$ is the difference between the maximal distance $d(e_i, s_j)$ and the setup time st^r_{ij}. Hence, it provides a measure of the degree of *sequencing flexibility* between a_i and a_j [3] taking into account the setup time constraint $e_i + st^r_{ij} \leq s_j$. If $slack(e_i, s_j) < 0$, then the ordering $a_i \preceq a_j$ is not feasible.
- *co-slack*$(e_i, s_j) = -d(s_j, e_i) - st^r_{ij}$ is the difference between the minimum possible distance between a_i and a_j, $-d(s_i, e_j)$, and the setup time st^r_{ij}; if *co-slack*$(e_i, s_j) \geq 0$, then there is no need to separate a_i and a_j, as the setup time constraint $e_i + st^r_{ij} \leq s_j$ is already satisfied.

[1] Between the end-time e_i of a_i and the start-time s_j of a_j

[2] Between the end-time e_j of a_j and the start-time s_i of a_i

[3] Intuitively, the higher is the degree of *sequencing flexibility*, the larger is the set of feasible assignments to the start-times of a_i and a_j

For any pair of activities a_i and a_j that can **compete for the same resource** r ($R_i \cap R_j \neq \emptyset$), given the corresponding durations p_{ir} and p_{jr}, the Dominance Conditions, describing the four main possible cases of conflict, are defined as follows:

1. $slack(e_i, s_j) < 0 \wedge slack(e_j, s_i) < 0$
2. $slack(e_i, s_j) < 0 \wedge slack(e_j, s_i) \geq 0 \wedge co\text{-}slack(e_j, s_i) < 0$
3. $slack(e_i, s_j) \geq 0 \wedge slack(e_j, s_i) < 0 \wedge co\text{-}slack(e_i, s_j) < 0$
4. $slack(e_i, s_j) \geq 0 \wedge slack(e_j, s_i) \geq 0$

Condition 1 represents an *unresolvable conflict*. There is no way to order a_i and a_j taking into account the setup times st^r_{ij} and st^r_{ji}, without inducing a negative cycle in the graph $G_d(V, E)$. When Condition 1 is verified there are four different interesting sub-cases generated on the basis of the cardinality of the domain sets R_i and R_j.

a. $|R_i| = |R_j| = 1$: the search has reached a *failure* state;

b. $|R_i| = 1 \wedge |R_j| > 1$: the resource requirement r can be removed from R_j;

c. $|R_i| > 1 \wedge |R_j| = 1$: the resource requirement r can be removed from R_i;

d. $|R_i| > 1 \wedge |R_j| > 1$: the activities a_i and a_j cannot use the same resource r.

Conditions 2, and 3, alternatively, distinguish *uniquely resolvable conflicts*, i.e., there is only one feasible ordering of a_i and a_j when both the activities require r, and the decision of which constraint to post is thus unconditional. In the particular case where $|R_i| = |R_j| = 1$ the decision $a_j \preceq a_i$ is mandatory; if Condition 2 is verified, only $a_j \preceq a_i$ leaves $G_d(V, E)$ consistent. It is worth noting that the presence of the condition $co\text{-}slack(e_j, s_i) < 0$ entails that the minimal distance between the end time e_j and the start time s_i is shorter than the minimal required setup time st^r_{ji}; hence, we still need to impose the constraint $e_j + st^r_{ji} \leq s_i$. In other words, the *co-slack* condition avoids the imposition of unnecessary precedence constraints for trivially solved conflicts. Condition 3 works similarly, and entails that only the $a_i \preceq a_j$ ordering is feasible. In case there is at least one activity with more than one resource option ($|R_i| > 1 \vee |R_j| > 1$), it is still possible to choose different resource assignments for a_i and a_j, and avoid posting a precedence constraint. Condition 3 works similarly, and entails that only the $a_i \preceq a_j$ ordering is feasible when $|R_i| = |R_j| = 1$.

Condition 4 designates a class of *resolvable conflicts* with more search options; in this case when $|R_i| = |R_j| = 1$ both orderings of a_i and a_j remain feasible, and it is therefore necessary to perform a *search decision*. When there is at least one activity a_i or a_j with more than one resource option ($|R_i| > 1 \vee |R_j| > 1$), then there is also the possibility of choosing different resource assignment to a_i and a_j, and avoid to post a precedence constraint.

Heuristic Analysis

Shortest path information in G_d can also be exploited to define *variable* and *value* ordering heuristics for the decision variables x_i and o_{ijr} in all cases where no mandatory decisions are deduced from the propagation phase. The idea is to evaluate both types of decision variables (x_i and o_{ijr}) and select the one (independently of type) with the minimum heuristic evaluation. The selection of the variables is based on the *most constrained first* (MCF) principle and the selection of values follows the *least constraining value* (LCV) heuristic.

Ordering decision variables. We start to analyze the case of selecting an ordering decision variables o_{ijr}, under the hypothesis that both the activity a_i and a_j use the same resource $r \in R_i \cap R_j$. As stated above, in this context $slack(e_i, s_j)$ and $slack(e_j, s_i)$ provide measures of the degree of *sequencing flexibility* between a_i and a_j. More precisely, given a variable o_{ijr}, related to the pair (a_i, a_j), its heuristic evaluation

$$VarEval(a_i, a_j) = \begin{cases} min\{\frac{slack(e_i,s_j)}{\sqrt{S}}, \frac{slack(e_j,s_i)}{\sqrt{S}}\} : \\ \quad \text{if } slack(e_i, s_j) \geq 0 \wedge slack(e_j, s_i) \geq 0 \\ slack(e_j, s_i) : \\ \quad \text{if } slack(e_i, s_j) < 0 \wedge slack(e_j, s_i) \geq 0 \\ slack(e_i, s_j) : \\ \quad \text{if } slack(e_i, s_j) \geq 0 \wedge slack(e_j, s_i) < 0 \end{cases}$$

where $S = \frac{min\{slack(e_i,s_j), slack(e_j,s_i)\}}{max\{slack(e_i,s_j), slack(e_j,s_i)\}}$ [4]. The *variable* ordering heuristic attempts to focus first on the most constrained conflict (a_i, a_j), that is, on the conflict with the least amount of temporal flexibility (i.e., the conflict that is closest to previous Condition 1.a).

As opposed to variable ordering, the *value* ordering heuristic attempts to resolve the selected conflict (a_i, a_j) by simply choosing the activity pair sequencing that retains the highest amount of temporal flexibility (least constrained value). Specifically, $a_i \preceq a_j$ is selected if $slack(e_i, s_j) > slack(e_j, s_i)$ and $a_j \preceq a_i$ is selected otherwise.

Resource decision variables. Decision variables x_i are also selected according to the MCF principle. Initially, all the pairs of activities (a_i, a_j), such that ($|R_i| > 1 \vee |R_j| > 1$ and $R_i \cap R_j \neq \emptyset$)) undergo a double-key sorting, where the primary key is a heuristic evaluation based on resource flexibility and computed as $F_{ij} = 2(|R_i| + |R_j|) - |R_i \cap R_j|$, while the secondary key is the known $VarEval(a_i, a_j)$ heuristic, based on temporal flexibility[5]. Then, we select the pair (a^*_i, a^*_j) with the lowest value of the pair $\langle F_{ij}, VarEval(a_i, a_j)\rangle$, where $VarEval(a_i, a_j)$ is computed for each possible resource $r \in R_i \cap R_j$. Finally, between x^*_i and x^*_j we select the variable whose domain of values has the lowest cardinality.

Value ordering on the decision variables x_i is also accomplished by using temporal flexibility measures. If R_i

[4] The $\sqrt{S}$ bias is introduced to take into account cases where a first conflict with the overall $min\{slack(e_i, s_j), slack(e_j, s_i)\}$ has a very large $max\{slack(e_i, s_j), slack(e_j, s_i)\}$, and a second conflict has two shortest path values just slightly larger than this overall minimum. In such situations, it is not clear which conflict has the least temporal flexibility.

[5] The resource flexibility F_{ij} increases with the size of the domains R_i and R_j, and decreases with the size of the set $R_i \cap R_j$, which is correlated to the possibility of creating resource conflicts.

PCP(*Problem,* C_{max})
1. $S \leftarrow$ InitSolution(*Problem*, C_{max})
2. **loop**
3. Propagate(S)
4. **if** UnresolvableConflict(S)
5. **then return**(nil)
6. **else**
7. **if** UniquelyResolvableDecisions(S)
8. **then** PostUnconditionalConstraints(S)
9. **else begin**
10. $C \leftarrow$ChooseDecisionVariable(S)
11. **if** ($C = nil$)
12. **then return**(S)
13. **else begin**
14. $vc \leftarrow$ ChooseValueConstraint(S, C)
15. PostConstraint(S, vc)
16. **end**
17. **end**
18. **end-loop**
19. **return**(S)

Figure 1: The PCP one-shot algorithm

is the domain of the selected decision variable x_i, then for each resource $r \in R_i$, we consider the set of activities A_r already assigned to resource r and calculate the value $F_{min}(r) = min_{a_k \in A_r}\{VarEval(a_i, a_k)\}$. Then, for each resource r we evaluate the flexibility associated with the most critical pair (a_i, a_k), under the hypothesis that the resource r is assigned to a_i. The resource $r^* \in R_i$ which maximizes the value $F_{min}(r)$, and therefore allows a_i to retain maximal flexibility, is selected.

The PCP Algorithm

Figure 1 gives the basic overall PCP solution procedure, which starts from an empty solution (Step 1) where the graphs G_d is initialized according to Section . Also, the procedure accepts a *never-exceed* value (C_{max}) of the objective function of interest, used to impose an initial *global* makespan to all the jobs. The PCP algorithm shown in Figure 1 analyses the decision variables x_i and o_{ijr}, and respectively decides their *values* in terms of imposing a duration constraint on a selected activity or a setup time constraint (i.e., $a_i \preceq a_j$ or $a_j \preceq a_i$, see Section). In broad terms, the procedure in Figure 1 interleaves the application of Dominance Conditions (Steps 4 and 7) with variable and value ordering (Steps 10 and 14 respectively) and updating of the solution graph G_d (Steps 8 and 15) to conduct a single pass through the search tree. At each cycle, a propagation step is performed (Step 3) by the function `Propagate`(S), which propagates the effects of posting a new solving decision (i.e., a setup time constraint) in the graph G_d. In particular, `Propagate`(S) updates the shortest path distances on the graph G_d. A solution S is found when the PCP algorithm finds a feasible assignment of resources $\overline{r_i} \in R_i$ to activities a_i ($i = 1 \ldots n\}$) and when none of the four dominance conditions is verified on S. In fact, when none of the four Dominance Conditions is verified (and the PCP procedure exits with success), for each resource r, the set of activities A_r assigned

IFS(S,$MaxFail$, γ)
begin
1. $S_{best} \leftarrow S$
2. $counter \leftarrow 0$
3. **while** ($counter \leq MaxFail$) **do**
4. RELAX(S, γ)
5. $S \leftarrow$PCP($S, C_{max}(S_{best})$)
6. **if** $C_{max}(S) < C_{max}(S_{best})$ **then**
7. $S_{best} \leftarrow S$
8. counter $\leftarrow$ 0
9. **else**
10. counter $\leftarrow$ counter + 1
11. **return** (S_{best})

end

Figure 2: The IFS schema

to r represents a total execution order. In addition, as the graph G_d represents a consistent Simple Temporal Problem (see Section), one possible solution of the problem is the *earliest-time solution*, such that $S = \{(-d(s_1, tp_0), \overline{r_1}), (-d(s_2, tp_0), \overline{r_2}), \ldots, (-d(s_n, tp_0), \overline{r_n})\}$.

The Optimization Metaheuristic

Figure 2 introduces the generic IFS procedure. The algorithm basically alternates relaxation and flattening steps until a better solution is found or a maximal number of iterations is executed. The procedure takes three parameters as input: (1) an initial solution S; (2) a positive integer $MaxFail$, which specifies the maximum number of consecutive non makespan-improving moves that the algorithm will tolerate before terminating; (3) a parameter γ explained in . After the initialization (Steps 1-2), a solution is repeatedly modified within the while loop (Steps 3-10) by applying the RELAX procedure (as explained in the following section), and the PCP procedure shown in Figure 1 used as flattening step. At each iteration, the RELAX step reintroduces the possibility of resource contention, and the PCP step is called again to restore resource feasibility. In the case a better makespan solution is found (Step 6), the new solution is saved in S_{best} and the *counter* is reset to 0. If no improvement is found within $MaxFail$ moves, the algorithm terminates and returns the best solution found.

Relaxation Procedure

The first part of the IFS cycle is the *relaxation step*, wherein a feasible schedule is relaxed into a possibly resource infeasible, but precedence feasible, schedule by retracting some number of scheduling decisions. Here we use a strategy similar to the one in (Godard, Laborie, and Nuitjen 2005) and called *chain-based relaxation*. Given the graph representation described above, each scheduling decision is either a *setup time constraint* between a pair of activities that are competing for the same resource capacity and/or a *resource assignment* to one activity. The strategy starts from a solution S and randomly *breaks* some total orders (or *chains*) imposed on the subset of activities requiring the same resource r. The relaxation strategy requires an input solution as a graph $G_S(A, J, X_S)$ which is a modification of the original precedence graph G that represents the input scheduling

problem. G_S contains a set of additional *general* precedence constraints X_S which can be seen as a set of *chains*. Each chain imposes a total order on a subset of problem activities requiring the same resource.

The *chain-based relaxation* proceeds in two steps. Firstly, a subset of activities a_i is randomly selected from the input solution S, according to some criteria that will be explained below. The selection process is generally driven by a parameter $\gamma \in (0, 1)$ that indicates the probability that each activity has to be selected (γ is called the *relaxing factor*). For each selected activity, the resource assignment is removed and the original set of available options R_i is re-estabilished. Secondly, a procedure similar to CHAINING – used in (Policella et al. 2007) – is applied to the set of unselected activities. This operation is in its turn accomplished in three steps: (1) all previously posted setup time constraints X_S are removed from the solution S; (2) the unselected activities are sorted by increasing earliest start times of the input solution S; (3) for each resource r and for each unselected activity a_i assigned to r (according to the increasing order of start times), a_i's predecessor $p = pred(a_i, r)$ is considered and the setup time constraint related to the sequence $p \preceq a_i$ is posted (the dummy activity a_0 is the first activity of all the chains). This last step is iterated until all the activities are linked by the correct setup time constraints. Note that this set of unselected activities still represents a feasible solution to a scheduling sub-problem, which is represented as a graph G_S in which the randomly selected activities *float* outside the solution and thus re-create *conflict* in resource usage.

As anticipated above, we implemented two different mechanisms to perform the random activity selection process, respectively called *Random* and a *Slack-based*.

Random selection According to the random selection approach, at each solving cycle of the IFS algorithm in Figure 2, a subset of activities a_i is randomly selected from the input solution S, with each activity having an uniformly distributed selection probability equal to γ. It is of great importance to underscore that according to this approach, the activities to be relaxed are randomly picked up from the solution S *with the same probability* which, as we will see shortly, entails a relaxation characterized by a greater disruption on S, compared to the following selection approach.

Slack-based selection As opposed to the random selection, at each iteration the slack-based selection approach restricts the pool of the relaxable activities to the subset containing those activities that are closer to the *critical path condition* (*critical path set*). As known, an activity a_i belongs to the critical path (i.e., meets the critical path condition) when, given a_i's end time e_i and its feasibility interval $[lb_i, ub_i]$, the condition $lb_i = ub_i$ holds. For each activity a_i, the smaller the difference $ub_i - lb_i$ (informally called *slack*) computed on e_i, the closer is a_i to the critical path condition. At each IFS iteration the critical path set is built so as to contain any activity a_i with a probability directly proportional to the γ parameter and inversely proportional to a_i's slack. For obvious reasons, the slack-based relaxation entails a smaller disruption on the solution S, as it operates on a smaller set of activities; the activities characterized by a great slack will have a minimum probability to be selected. As explained in the following section, this difference has important consequences on the experimental behaviour.

Experimental Analysis

The empirical evaluation has been carried out on a SDST-FJSSP benchmark set synthesized on purpose out of the first 20 instances of the *edata* subset of the FJSSP *HUdata* testbed from (Hurink, Jurisch, and Thole 1994), and will therefore be referred to as *SDST-HUdata*. Each one of the *SDST-HUdata* instances has been created by adding to the original *HUdata* instance one Setup-Time matrix $st^r(nJ \times nJ)$ for each present machine r, where nJ is the number of present jobs. The same Setup-Time matrix was added for each machine of all the benchmark instances. Each value $st^r_{i,j}$ in the Setup-Time matrix models the setup time necessary to reconfigure the r-th machine to switch from job i to job j. Note that machine reconfiguration times are sequence dependent: setting up a machine to process a product of type j after processing a product of type i can generally take a different amount of time than setting up the same machine for the opposite transition. The elements $st^r_{i,j}$ of the Setup-Time matrix satisfy the *triangle inequality* (Brucker and Thiele 1996; Artigues and Feillet 2008), that is, for each three activities a_i, a_j, a_k requiring the same machine, the inequality $st^r_{ij} \leq st^r_{ik} + st^r_{kj}$ holds. The 20 instances taken from *HUdata* (namely, the instances *la01-la20*) are divided in four groups of five $(nJ \times nA)$ instances each, where nJ is the number of jobs and nA is the number of activities per job for each instance. More precisely, group *la01-la05* is (10×5), group *la06-la10* is (15×5), group *la11-la15* is (20×5), and group *la16-la20* is (10×10). In all instances, the processing times on machines assignable to the same activity are identical, as in the original *HUdata* set. The algorithm used for these experiments has been implemented in Java and run on a AMD Phenom II X4 Quad 3.5 Ghz under Linux Ubuntu 10.4.1.

Results. Table 1 and Table 2 show the obtained results running our algorithm on the *SDST-HUdata* set using the *Random* or *Slack-based* procedure in the IFS relaxation step, respectively. Both tables are composed of 10 columns and 23 rows (one row per problem instance plus three data wrap-up rows). The ***best*** column lists the shortest makespans obtained in the experiments for each instance; underlined values represent the best values obtained from both tables (global bests). The columns labeled $\gamma = 0.2$ to $\gamma = 0.9$ (see Section) contain the results obtained running the IFS procedure with a different value for the *relaxing factor* γ. For each problem instance (i.e., for each row) the values in bold indicate the best makespan found among all the tested γ values (γ runs).

For each γ run, the last three rows of both tables show respectively (up-bottom): (1) the number B of best solutions

Table 1: Results with random selection procedure

inst.	*best*	γ							
		0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9
la01	**726**	772	731	728	**726**	729	**726**	729	740
la02	**749**	785	785	**749**	**749**	**749**	**749**	**749**	768
la03	**652**	677	658	658	658	**652**	**652**	658	675
la04	**673**	**673**	**673**	**673**	689	689	680	680	690
la05	**603**	613	613	**603**	605	605	606	607	632
la06	**950**	965	**950**	954	954	971	997	995	1020
la07	**916**	946	**916**	925	919	947	950	987	1000
la08	**954**	973	961	964	**954**	963	958	1000	1001
la09	**1002**	1039	**1002**	1039	1020	1042	1020	1045	1068
la10	**977**	1017	**977**	1022	**977**	1027	1008	1042	1048
la11	**1265**	**1265**	1312	1285	1282	1345	1332	1372	1368
la12	**1088**	**1088**	1114	1130	1167	1165	1199	1209	1198
la13	**1255**	**1255**	**1255**	**1255**	1300	1280	1300	1316	1315
la14	**1292**	**1292**	1315	1344	1346	1362	1351	1345	1372
la15	**1298**	**1298**	1302	1338	1355	1352	1367	1388	1429
la16	**1012**	1028	**1012**	**1012**	**1012**	**1012**	**1012**	**1012**	1023
la17	**864**	881	885	885	**864**	888	**864**	**864**	902
la18	**985**	1021	1007	1029	999	**985**	**985**	**985**	**985**
la19	**956**	1006	992	975	**956**	**956**	978	959	981
la20	**997**	1008	1010	**997**	**997**	**997**	**997**	**997**	999
B (**N**)	**12**	**6(1)**	**7(5)**	**6(4)**	**8(5)**	**6(5)**	**7(5)**	**5(3)**	**1(1)**
Av.C.		20149	17579	14767	11215	10950	9530	7782	7588
Av.MRE		19.34	18.29	18.66	18.37	19.42	19.43	20.60	22.44

found *locally* (i.e., within the current table) and, underlined within round brackets, the number **N** of best solutions found *globally* (i.e., between both tables); (2) the average number of utilized solving cycles (Av.C.), and (3) the average mean relative error ($Av.MRE$)[6] with respect to the lower bounds of the original *HUdata* set (i.e., without setup times), reported in (Mastrolilli and Gambardella 2000). For all runs, a maximum CPU time limit was set to 800 seconds.

Table 2: Results with slack-based selection procedure

inst.	*best*	γ							
		0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9
la01	**726**	739	736	**726**	**726**	**726**	**726**	**726**	**726**
la02	**749**	785	**749**	**749**	**749**	**749**	**749**	**749**	**749**
la03	**652**	658	658	658	658	658	**652**	658	658
la04	**673**	686	686	686	**673**	686	680	**673**	680
la05	**603**	613	**603**	613	605	**603**	604	**603**	605
la06	**960**	963	963	971	**960**	963	962	970	970
la07	**925**	941	966	941	**925**	931	946	972	1000
la08	**948**	983	963	**948**	964	993	967	994	973
la09	**1002**	1020	1020	**1002**	**1002**	1040	1069	1052	1042
la10	**985**	993	991	1007	1022	1022	1017	**985**	1024
la11	**1256**	**1256**	1257	1295	1295	1308	1318	1324	1332
la12	**1082**	**1082**	1097	1098	1159	1152	1188	1163	1207
la13	**1215**	1222	1240	1240	1223	**1215**	1311	1301	1311
la14	**1285**	1308	**1285**	1285	1311	1295	1335	1372	1345
la15	**1291**	1333	**1291**	1330	1302	1311	1383	1389	1412
la16	**1007**	1012	1012	1012	**1007**	1012	1012	1012	1012
la17	**858**	889	868	893	895	888	**858**	859	872
la18	**985**	1019	1025	1021	1007	**985**	**985**	**985**	**985**
la19	**956**	1006	976	987	984	**956**	980	**956**	959
la20	**997**	**997**	1033	**997**	**997**	**997**	1003	**997**	**997**
B (**N**)	**17**	**3(3)**	**4(4)**	**5(5)**	**8(6)**	**7(7)**	**5(5)**	**8(7)**	**4(4)**
Av.C.		21273	18068	15503	13007	10643	10653	8639	8575
Av.MRE		18.67	18.09	18.26	18.19	18.14	19.58	19.44	20.16

One significant result that the tables show is the difference in the average of utilized solving cycles (Av.C. row) between the *random* and the *slack-based* relaxation procedure. In fact, it can be observed that on average the slack-based approach uses more solving cycles in the same allotted time than its random counterpart (i.e., the slack-based relaxation heuristic is faster in the solving process). This is explained by observing that the slack-based relaxation heuristic entails a less severe disruption of the current solution at each solving cycle compared to the random heuristic, as the former generally relaxes a lower number of activities (given the same γ value). The lower the disruption level of the current solution in the relaxation step, the easier it is to re-gain solution feasibility in the flattening step. In addition of this efficiency issue, the slack-based relaxation approach also provides the extra effectiveness deriving from operating in the vicinity of the critical path of the solution, as demonstrated in (Cesta, Oddi, and Smith 2000).

The good performance exhibited by the slack-based heuristic can be also observed by inspecting the B(**N**) rows in both tables. Clearly, the slack-based approach finds a higher number of best solutions (17 against 12), which is confirmed by comparing the number of locally found bests (B) with the global ones (**N**), for each γ value, and for both heuristics.

Another interesting aspect can be found analyzing the γ values range where the best performances are obtained ($Av.MRE$ row). Inspecting the $Av.MRE$ values, the following can in fact be stated: (1) the slack-based heuristic finds solutions of higher quality w.r.t. the random heuristic over the complete γ variability range; (2) in the random case, the best results are obtained in the $[0.3, 0.5]$ γ range, while in the slack-based case the best γ range is wider ($[0.3, 0.6]$).

Conclusions

In this paper we have proposed the use of Iterative Flattening Search (IFS) as a means of effectively solving the SDST-FJSSP. The proposed algorithm uses as its core solving procedure an extended version of the SP-PCP procedure presented in (Oddi and Smith 1997) which introduces a new set of dominance conditions tailored to capture the SDST-FJSSP's features (i.e., alternative activity-resource assignments plus sequence dependent setup times), as well as a new relaxation strategy. The performance of the procedure has been tested on a modified version of a known FJSSP benchmark set (i.e., the Hurink edata), integrated with a series of sequence dependent setup time constraints. The new relaxation strategy has been compared against a fully random version, demonstrating a noteworthy improvement in performance. To the best of our knowledge, no similar works exist on the same problem version to allow direct comparison with different approaches. The benchmark used in this analysis will be made available on the net so as to facilitate experiment reproducibility and encourage research competition.

Acknowledgments

CNR authors are partially supported by EU under the ULISSE project (Contract FP7.218815), and MIUR under the PRIN project 20089M932N (funds 2008).

[6]The individual MRE of each solution is computed as follows: $MRE = 100 \times (C_{max} - LB)/LB$, where C_{max} is the solution makespan and LB is the instance's lower bound

References

Adams, J.; Balas, E.; and Zawack, D. 1988. The shifting bottleneck procedure for job shop scheduling. *Management Science* 34(3):391–401.

Allahverdi, A., and Soroush, H. 2008. The significance of reducing setup times/setup costs. *European Journal of Operational Research* 187(3):978–984.

Allahverdi, A.; Ng, C.; Cheng, T.; and Kovalyov, M. 2008. A survey of scheduling problems with setup times or costs. *European Journal of Operational Research* 187(3):985–1032.

Artigues, C., and Feillet, D. 2008. A branch and bound method for the job-shop problem with sequence-dependent setup times. *Annals OR* 159(1):135–159.

Balas, E.; Simonetti, N.; and Vazacopoulos, A. 2008. Job shop scheduling with setup times, deadlines and precedence constraints. *Journal of Scheduling* 11(4):253–262.

Brucker, P., and Thiele, O. 1996. A branch & bound method for the general-shop problem with sequence dependent setup-times. *OR Spectrum* 18(3):145–161.

Brucker, P.; Jurisch, B.; and Sievers, B. 1994. A branch and bound algorithm for the job-shop scheduling problem. *Discrete Applied Mathematics* 49(1-3):107–127.

Cesta, A.; Oddi, A.; and Smith, S. F. 2000. Iterative Flattening: A Scalable Method for Solving Multi-Capacity Scheduling Problems. In *AAAI/IAAI. 17^{th} National Conference on Artificial Intelligence*, 742–747.

Dechter, R.; Meiri, I.; and Pearl, J. 1991. Temporal constraint networks. *Artificial Intelligence* 49:61–95.

Gao, J.; Sun, L.; and Gen, M. 2008. A hybrid genetic and variable neighborhood descent algorithm for flexible job shop scheduling problems. *Computers & Operations Research* 35:2892–2907.

Garey, M. R., and Johnson, D. S. 1979. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. New York, NY, USA: W. H. Freeman & Co.

Godard, D.; Laborie, P.; and Nuitjen, W. 2005. Randomized Large Neighborhood Search for Cumulative Scheduling. In *ICAPS-05. Proceedings of the 15^{th} International Conference on Automated Planning & Scheduling*, 81–89.

González, M. A.; Vela, C. R.; and Varela, R. 2009. A Tabu Search Algorithm to Minimize Lateness in Scheduling Problems with Setup Times. In *Proceedings of the CAEPIA-TTIA 2009 13th Conference of the Spanish Association on Artificial Intelligence*.

Hmida, A. B.; Haouari, M.; Huguet, M.-J.; and Lopez, P. 2010. Discrepancy search for the flexible job shop scheduling problem. *Computers & Operations Research* 37:2192–2201.

Hurink, J.; Jurisch, B.; and Thole, M. 1994. Tabu search for the job-shop scheduling problem with multi-purpose machines. *OR Spectrum* 15(4):205–215.

Mastrolilli, M., and Gambardella, L. M. 2000. Effective neighbourhood functions for the flexible job shop problem. *Journal of Scheduling* 3:3–20.

Michel, L., and Van Hentenryck, P. 2004. Iterative Relaxations for Iterative Flattening in Cumulative Scheduling. In *ICAPS-04. Proceedings of the 14^{th} International Conference on Automated Planning & Scheduling*, 200–208.

Montanari, U. 1974. Networks of Constraints: Fundamental Properties and Applications to Picture Processing. *Information Sciences* 7:95–132.

Nowicki, E., and Smutnicki, C. 2005. An advanced tabu search algorithm for the job shop problem. *Journal of Scheduling* 8(2):145–159.

Oddi, A., and Smith, S. 1997. Stochastic Procedures for Generating Feasible Schedules. In *Proceedings 14th National Conference on AI (AAAI-97)*, 308–314.

Oddi, A.; Cesta, A.; Policella, N.; and Smith, S. F. 2010. Iterative flattening search for resource constrained scheduling. *J. Intelligent Manufacturing* 21(1):17–30.

Policella, N.; Cesta, A.; Oddi, A.; and Smith, S. 2007. From Precedence Constraint Posting to Partial Order Schedules. *AI Communications* 20(3):163–180.

Rossi, A., and Dini, G. 2007. Flexible job-shop scheduling with routing flexibility and separable setup times using ant colony optimisation method. *Robotics and Computer-Integrated Manufacturing* 23(5):503–516.

Ruiz, R., and Maroto, C. 2006. A genetic algorithm for hybrid flowshops with sequence dependent setup times and machine eligibility. *European Journal of Operational Research* 169(3):781 – 800.

Sethi, A. K., and Sethi, S. P. 1990. Flexibility in manufacturing: A survey. *International Journal of Flexible Manufacturing Systems* 2:289–328. 10.1007/BF00186471.

Vallada, E., and Ruiz, R. 2011. A genetic algorithm for the unrelated parallel machine scheduling problem with sequence dependent setup times. *European Journal of Operational Research* 211(3):612 – 622.

Valls, V.; Perez, M. A.; and Quintanilla, M. S. 1998. A tabu search approach to machine scheduling. *European Journal of Operational Research* 106(2-3):277 – 300.

Vela, C. R.; Varela, R.; and González, M. A. 2009. Local search and genetic algorithm for the job shop scheduling problem with sequence dependent setup times. *Journal of Heuristics*.

Solving Resource Allocation/Scheduling Problems with Constraint Integer Programming

Stefan Heinz*
Zuse Institute Berlin
Berlin, Germany
heinz@zib.de

J. Christopher Beck
Department of Mechanical & Industrial Engineering
University of Toronto
Toronto, Canada
jcb@mie.utoronto.ca

Abstract

Constraint Integer Programming (CIP) is a generalization of mixed-integer programming (MIP) in the direction of constraint programming (CP) allowing the inference techniques that have traditionally been the core of CP to be integrated with the problem solving techniques that form the core of complete MIP solvers. In this paper, we investigate the application of CIP to scheduling problems that require resource and start-time assignments to satisfy resource capacities. The best current approach to such problems is logic-based Benders decomposition, a manual decomposition method. We present a CIP model and demonstrate that it achieves performance competitive to the decomposition while outperforming the standard MIP and CP formulations.

Introduction

Constraint Integer Programming (CIP) (Achterberg 2007b; 2009) is a generalization of both finite domain constraint programming (CP) and mixed integer programming (MIP) that allows the native integration of core problem solving techniques from each area. With a CIP solver, such as SCIP (Achterberg 2009), it is possible to combine the traditional strengths of MIP such as strong relaxations and cutting planes with global constraint propagation and conflict analysis. Indeed, SCIP can be seen as a solver framework that integrates much of the core of MIP, CP, and SAT solving methodologies.

Our primary goal in this paper is to start a broad investigation of CIP as a general approach to scheduling problems. While CP tends to be very successful on a variety of scheduling problems, it is challenged by problems that exhibit weak propagation either due to their objective functions (Kovács and Beck 2011) or to the need for a cascading series of interdependent decisions such as encountered in combined resource allocation and scheduling problems (Hooker 2005). We are interested to see if CIP techniques can address these challenges.

Our investigations in this paper focus on problems which combine resource allocation and scheduling. Given a set of jobs that require the use one out of a set of alternative resources, a solution will assign each job to a resource and schedule the jobs such that the capacity of each resource is respected at all time points. We address this problem in CIP by the `optcumulative` global constraint in SCIP, extending the `cumulative` constraint to allow each job to be optional. That is, a binary decision variable is associated with each job and resource pair and the corresponding cumulative constraint includes these variables in its scope. This approach is not novel in CP, dating back to at least (Beck and Fox 2000). We handle the requirement that exactly one resource must be chosen for a job in standard MIP fashion by specifying that the sum of the binary variables for a given job, across all resources, must be one.

Our experimental results demonstrate that our preliminary implementation of the `optcumulative` constraint is sufficient to allow a CIP model to be competitive with the state-of-the-art logic-based Benders decomposition (LBBD) on two problems sets with unary and discrete resource capacity, respectively.

In the next section, we formally define the scheduling problems and provide necessary background on CIP, LBBD, and the `cumulative` global constraint which forms the basis for the `optcumulative` constraint. We then present four models of our scheduling problems: MIP, LBBD, CP, and CIP. The following section contains our empirical investigations, both initial experiments comparing the four models and a detailed attempt to develop an understanding of the impact of primal heuristics for the CIP performance. In the final section, we conclude.

Background

In this section we introduce the scheduling problem under investigation and give background on CIP, LBBD, and the `cumulative` constraint.

Problem Definition

We study two classes of scheduling problems referred to as UNARY and MULTI (Hooker 2005; Beck 2010). Both problems are defined by a set of jobs $\mathcal{J}$, and a set of resources $\mathcal{K}$. Each job has a release date, $\mathcal{R}_j$, a deadline, $\mathcal{D}_j$, a resource-specific processing time, p_{jk}, a resource assignment cost, c_{jk}, and a resource requirement, r_{jk}. Each job, j, must be assigned to one resource, k, and scheduled to start at or after

*Supported by the DFG Research Center MATHEON *Mathematics for key technologies* in Berlin.

its release date, end at or before its due date, and execute for p_{jk} time units. Each resource, $k \in \mathcal{K}$, has a capacity, C_k, and an associated resource constraint which states that for each time point, the sum of the resource requirements of the executing jobs must not exceed the resource capacity. A feasible solution is an assignment where each job is placed on exactly one resource and no resource is over capacity. The goal is to find an optimal solution, a feasible solution which minimizes the total resource assignment cost.

In the UNARY problem, the capacity of each resource and the requirement of each job is one. In the MULTI problem, capacities and requirements may be non-unary.

Constraint Integer Programming

Mixed integer programming (MIP) and *satisfiability testing (SAT)* are special cases of the general idea of *constraint programming (CP)*. The power of CP arises from the possibility to model a given problem directly with a large variety of different, expressive constraints. In contrast, SAT and MIP only allow for very specific constraints: Boolean clauses for SAT and linear and integrality constraints for MIP. Their advantage, however, lies in the sophisticated techniques available to exploit the structure provided by these constraint types.

The goal of *constraint integer programming (CIP)* is to combine the advantages and compensate for the weaknesses of CP, MIP, and SAT. It was introduced by Achterberg and implemented in the framework SCIP (Achterberg 2009). Formally a constraint integer program can be defined as follows.

Definition 1. A *constraint integer program (CIP)* $(\mathfrak{C}, I, c)$ consists of solving

$$c^\star = \min\{c^T x \mid \mathfrak{C}(x),\ x \in \mathbb{R}^n,\ x_j \in \mathbb{Z} \text{ for all } j \in I\}$$

with a finite set $\mathfrak{C} = \{\mathcal{C}_1, \ldots, \mathcal{C}_m\}$ of constraints $\mathcal{C}_i : \mathbb{R}^n \to \{0,1\}, i = \{1, \ldots, m\}$, a subset $I \subseteq N = \{1, \ldots, n\}$ of the variable index set, and an objective function vector $c \in \mathbb{R}^n$. A CIP must fulfill the following additional condition:

$$\forall \hat{x}_I \in \mathbb{Z}^I\ \exists (A', b') : \quad (1)$$
$$\{x_C \in \mathbb{R}^C \mid \mathfrak{C}(\hat{x}_I, x_C)\} = \{x_C \in \mathbb{R}^C \mid A' x_C \leq b'\}$$

with $C := N \setminus I$, $A' \in \mathbb{R}^{k \times C}$, and $b' \in \mathbb{R}^k$ for some $k \in \mathbb{Z}_{\geq 0}$.

Restriction (1) ensures that the sub-problem remaining after fixing all integer variables is a linear program. Note that this does not forbid quadratic or other nonlinear, and more involved expressions – as long as the nonlinearity only refers to the integer variables.

The central solving approach for CIP as implemented in the SCIP framework is branch-and-cut-and-propagate: as in SAT, CP, and MIP-solvers, SCIP performs a branch-and-bound search to decompose the problem into sub-problems. Also as in MIP, a linear relaxation, strengthened by additional inequalities/cutting planes if possible, is solved at each search node and used to guide and bound search. Similar to CP solvers, inference in the form of constraint propagation is used at each node to further restrict search and detect dead-ends. Moreover, as in SAT solving, SCIP uses conflict analysis and restarts. In more detail, CIP solving includes the following techniques

- *Presolving.* The purpose of presolving, which takes place before the tree search is started, is threefold: first, it reduces the size of the model by removing irrelevant information such as redundant constraints or fixed variables. Second, it strengthens the linear programming relaxation of the model by exploiting integrality information, e.g., to tighten the bounds of the variables or to improve coefficients in the constraints. Third, it extracts information from the model such as implications or cliques which can be used later for branching and cutting plane separation.
- *Propagation.* Propagation is used in the same fashion as in CP for pruning variable domains during the search.
- *Linear Relaxation.* A generic problem relaxation can be defined that includes only linear constraints. The relaxation can be solved efficiently to optimality and used in two primary ways: first to provide a guiding information for the search and second as the source of the "dual bound" a valid lower (upper) bound on the objective function for a minimization (maximization) problem.
- *Conflict Analysis.* The idea of conflict analysis is to reason about infeasible sub-problems which arise during the search in order to generate *conflict clauses* (Marques-Silva and Sakallah 1999; Achterberg 2007a) also known as *no-goods*. These conflict clauses are used to detect similar infeasible sub-problems later in the search. In order to perform conflict analysis, a bound change which was performed during the search, due to a propagation algorithm for example, needs to be explained. Such an *explanation* is a set of bounds which imply the performed bound change. The explanations are used to build up so-called conflict graph which lead to derivation of valid conflict clauses.

A global constraint in the framework of CIP can, but does not have too, contribute to all of these techniques. For example, as in CP, it can provide propagation algorithms for shrinking variable domains while also adding linear constraints to the linear programming relaxation, and supplying explanations to the conflict analysis reasoning. The minimal function of a global constraint is to "check" candidate solutions returning whether it is satisfied or not by a given variable assignment.

CIP has been applied to MIP (Achterberg 2009), mixed-integer nonlinear programming (Berthold, Heinz, and Vigerske 2009), nonlinear pseudo-Boolean programming (Berthold, Heinz, and Pfetsch 2009), the verification of chip designs (Achterberg, Brinkmann, and Wedler 2007), and scheduling (Berthold et al. 2010). The final paper is most relevant to the work here. Berthold et al. applied SCIP to resource-constrained project scheduling problems and demonstrated that CIP is competitive with the state-of-the-art in terms of finding both high quality solutions and in proving lower bounds on optimal solutions. This work forms one of our motivations for a broader investigation of CIP for scheduling problems.

Logic-based Benders Decomposition

Logic-based Benders decomposition (LBBD) (Hooker and Ottosson 2003) is a manual decomposition technique that generalizes classical Benders decomposition. A problem is modeled as a master problem (MP) and a set of sub-problems (SPs) where the MP is a relaxation of the global problem designed such that a solution to the MP induces one or more SPs. Each SP is an inference dual problem (Hooker 2005) that derives the tightest bound on the MP cost function that can be inferred from the current MP solution and the constraints and variables of the SP.

Solving a problem by LBBD is done by iteratively solving the MP to optimality and then solving each SP. If the MP solution satisfies all the bounds generated by the SPs, the MP solution is globally optimal. If not, a *Benders cut* is added to the MP by at least one violated SP and the MP is re-solved. For models where the SPs are feasibility problems, it is sufficient for correctness to solve the SPs to feasibility or generate a cut that removes the current MP solution.

In order for the MP search to be more than just a blind enumeration of its solution space, some relaxation of the SPs should be present in the MP model and the Benders cuts should do more than just remove the current MP solution.

LBBD has been successfully applied to a wide range of problems including scheduling (Beck 2010; Bajestani and Beck 2011), facility and vehicle allocation (Fazel-Zarandi and Beck 2011), and queue design and control problems (Terekhov, Beck, and Brown 2009).

Models & Solution Approaches

In this section, we define the models used for the UNARY and MULTI problems for MIP, LBBD, CP, and CIP.

Mixed Integer Programming

One of the standard MIP models for scheduling problems is the so-called *time-indexed* formulation (Queyranne and Schulz 1994). A decision variable, x_{jkt}, is defined, which is equal to 1 if and only if job j, starts at time t, on resource k. Summing over appropriate subsets of these variables can then enforce the resource capacity requirement. The model we use, taken from (Hooker 2005), is as follows:

$$\min \quad \sum_{k\in\mathcal{K}} \sum_{j\in\mathcal{J}} \sum_{t=\mathcal{R}_j}^{\mathcal{D}_j - p_{jk}} c_{jk}\, x_{jkt}$$

$$\text{s.t.} \quad \sum_{k\in\mathcal{K}} \sum_{t=\mathcal{R}_j}^{\mathcal{D}_j - p_{jk}} x_{kjt} = 1 \qquad \forall j \in \mathcal{J} \tag{2}$$

$$\sum_{j\in\mathcal{J}} \sum_{t'\in T_{jkt}} r_{jk}\, x_{jkt'} \le C_k \quad \forall k \in \mathcal{K},\ \forall t \tag{3}$$

$$x_{jkt} \in \{0,1\} \qquad \forall k \in \mathcal{K},\ \forall j \in \mathcal{J},\ \forall t,$$

with $T_{jkt} = \{t - p_{jk}, \ldots, t\}$.

The objective function minimizes the weighted resource assignment cost. Constraints (2) ensure that each job starts exactly once on one resource while Constraints (3) enforce the resource capacities on each resource at each time-point.

To solve this model, we rely on the default branch-and-bound search in the SCIP solver (Achterberg 2009). The default search has been tuned for solving MIP models and consists of a variety of modern algorithm techniques including: primal heuristics for finding feasible solutions, reliability-based branching heuristics, conflict analysis, and cutting planes. Details can be found in Achterberg and Berthold (2009).

Logic-based Benders Decomposition

The LBBD model (Hooker 2005; Beck 2010) defines two sets of decision variables: binary resource assignment variables, x_{jk}, which are assigned to 1 if and only if job j is assigned to resource k, and integer start time variables, S_j, which are assigned to the start-time of job j. The former variables are in the master problem while the latter are in sub-problems, one for each resource.

Formally, the LBBD master problem (MP) is defined as follows:

$$\min \quad \sum_{k\in\mathcal{K}} \sum_{j\in\mathcal{J}} c_{jk}\, x_{jk}$$

$$\text{s.t.} \quad \sum_{k\in\mathcal{K}} x_{jk} = 1 \qquad \forall j \in \mathcal{J} \tag{4}$$

$$\sum_{j\in\mathcal{J}} x_{jk}\, p_{jk} r_{jk} \le \hat{C}_k \quad \forall k \in \mathcal{K} \tag{5}$$

$$\sum_{j\in\mathcal{J}_{hk}} (1 - x_{jk}) \ge 1 \quad \forall k \in \mathcal{K},\ h \in [H-1]^1 \tag{6}$$

$$x_{kj} \in \{0,1\} \qquad \forall k \in \mathcal{K},\ \forall j \in \mathcal{J},$$

with $\hat{C}_k = C_k \cdot (\max_{j\in\mathcal{J}}\{\mathcal{D}_j\} - \min_{j\in\mathcal{J}}\{\mathcal{R}_j\})$.

As in the global MIP model, the objective function minimizes the total resource allocation costs. Constraints (4) ensure that each job is assigned to exactly one resource. Constraints (5) are a linear relaxation of each resource capacity constraint. They state that the area of the rectangle with height C_k and width from the smallest release date to the largest deadline must be greater than the sum of the areas of the jobs assigned to the resource.

Constraints (6) are the Benders cuts. Let H indicate the index of the current iteration and $\mathcal{J}_{hk}$ denote the set of jobs that resulted in an infeasible sub-problem for resource k in iteration $h < H$. The Benders cut, then, simply states that the set of jobs assigned to resource k in iteration h should not be reassigned to the same resource. This is a form of *no-good cut* (Hooker 2005).

Because the MP assigns each job to a resource and there are no inter-job constraints, the SPs are independent, single-machine scheduling problems where it is necessary to assign each job a start time such that its time window and the capacity of its resource are respected. The SP for resource k can be formulated as a constraint program as follows, where $\mathcal{J}_k$ denotes the set of jobs assigned to resource k:

[1] For an $n \in \mathbb{N}$ we define $[n] := \{1, \ldots, n\}$ and $[0] := \varnothing$.

$$\begin{aligned} &\texttt{cumulative}(\boldsymbol{S}, \boldsymbol{p_{\cdot k}}, \boldsymbol{r_{\cdot k}}, C_k) \\ &\mathcal{R}_j \leq S_j \leq \mathcal{D}_j - p_{jk} && \forall j \in \mathcal{J}_k \quad (7) \\ &S_j \in \mathbb{Z} && \forall j \in \mathcal{J}_k. \end{aligned}$$

$\boldsymbol{S}$, $\boldsymbol{p_{\cdot k}}$, and $\boldsymbol{r_{\cdot k}}$ are the vectors containing the start time variables and the processing times and demands for resource k with respect to subsets of jobs $\mathcal{J}_k$.

The global constraint `cumulative` (Baptiste, Pape, and Nuijten 2001) enforces the resource capacity constraint over all time-points at which a job may run. Constraints (7) enforce the time-windows for each job.

The MP and SPs are solved using the default search of SCIP. As CIP models admit global constraints, the subproblems are example of where the CIP model is solved primarily with CP technology.

Constraint Programming

As with MIP, we use the standard CP model for our problem (Hooker 2005). The start time of job j on resource k, is represented by the integer variable, S_{jk}. The model is as follows:

$$\begin{aligned} \min \quad & \sum_{k \in \mathcal{K}} \sum_{j \in \mathcal{J}} c_{jk}\, x_{jk} \\ \text{s.t.} \quad & \sum_{k \in \mathcal{K}} x_{jk} = 1 && \forall j \in \mathcal{J} \\ & \texttt{optcumulative}(\boldsymbol{S_{\cdot k}}, \boldsymbol{x_{\cdot k}}, \boldsymbol{p_{\cdot k}}, \boldsymbol{r_{\cdot k}}, C_k) \\ & && \forall k \in \mathcal{K} \quad (8) \\ & \mathcal{R}_j \leq S_j \leq \mathcal{D}_j - p_{jk} && \forall j \in \mathcal{J}_k \\ & x_{jk} \in \{0, 1\} && \forall j \in \mathcal{J},\ \forall k \in \mathcal{K} \\ & S_{jk} \in \mathbb{Z} && \forall j \in \mathcal{J},\ \forall k \in \mathcal{K}. \end{aligned}$$

Except for Constraints (8), the model components are analogous to those previously defined in the MIP and LBBD models. The `optcumulative` constraint is equivalent to the standard `cumulative` constraint with the addition that the jobs are optional: the jobs do not necessarily have to execute on this resource. The x_{jk} variable is used to indicate if job j executes on resource k and the `optcumulative` constraint include these variables in its scope. Formally, an assignment to the start time variables S_{jk} and binary choice variables x_{jk} for each job j and resource k is feasible if and only if the following condition holds at each time-point t:

$$\sum_{j \in \mathcal{J}: t \in [S_{jk}, S_{jk} + p_{jk})} x_{jk} r_{jk} \leq C_k.$$

We implement this model in IBM ILOG Solver and IBM ILOG Scheduler version 6.7. The `optcumulative` constraint is implemented by placing the set of machines in an *alternative resource set* and having each job require one resource from the set. The x_{jk} variable is implemented via the reification of the constraint stating that job j requires resource k.

To solve the problem, we use two pre-defined goals in the following order: `IloAssignAlternatives`, `IloSetTimesForward`. The first goal assigns jobs to resources in arbitrary order. When all jobs are assigned (and no dead-ends have been found via constraint propagation), the second goal implements the *schedule-or-postpone* heuristic (Pape et al. 1994) to assign start times to each job. Chronological backtracking is done when a dead-end is encountered.

Constraint Integer Programming

The CP model above is also the CIP model we use. Unlike the CP model, we implement and solve the CIP model using SCIP. As noted above, in CIP a global constraint such as `optcumulative` can contribute to the search in a number of ways. The current implementation of the `optcumulative` global constraint provides the following:

- *Presolving.* A number of problem reductions can be made in presolving, including normalization of the demands and the resource capacity and a detection of irrelevant jobs that do not influence the assignment/feasibility of remaining jobs on that resource. For example, if a job has a latest completion time which is smaller than the earliest start time of all remaining jobs then this job is irrelevant and can be ignored.
- *Propagation.* Following (Beck and Fox 2000), we adapt the standard bounds-based cumulative propagation (Baptiste, Pape, and Nuijten 2001) in a somewhat naive manner: we propagate all jobs that are known to execute on the resource. For each job j that is still optional, we perform singleton arc-consistency (SAC) (Debruyne and Bessière 1997): we assume that the job will execute on the resource and trigger propagation. If the propagation derives a dead-end, we can soundly conclude that the job cannot execute on the resource and appropriately set the x_{jk} variable. Otherwise, we retain the pruned domains for S_{jk}. In either case, the domains of all other variables are restored to their states before SAC. This propagation is stronger, but more costly, than the standard propagation of cumulative constraints with optional jobs due to Vilím, Barták, and Čepek (2005).
- *Linear Relaxation.* Each `optcumulative` constraint adds Constraint (5) as in the LBBD model to the linear programming relaxation.
- *Conflict Analysis.* Each time the `optcumulative` has to explain a bound change it first uses the `cumulative` explanation algorithm to derive an initial explanation. The explanation is extended with the the bounds of all choice variables which are (locally) fixed to one. In case of the SAC propagation, a valid explanation is the bounds of all choice variables which are fixed to one in the moment of the propagation.

The default parameters of SCIP are used to solve the CIP model. As these settings are tuned for pure MIP problems, it is likely that future work will be able to find more appropriate settings for CIPs.

Experiments

In this section we present first experimental results which indicate that the CIP model performs competitively with the LBBD model while out-performing MIP and CP.

Experimental Set-up

For all computational experiments except for the CP model we used the constraint integer programming solver SCIP (Achterberg 2009) that includes an implementation of the `cumulative` constraint (Berthold et al. 2010). As described above, we have extended SCIP by implementing the `optcumulative` global constraint. Using the same solver helps to focus on the impact of the models and algorithms used, controlling somewhat for different software engineering decisions made across different solvers. However, for the CP model, we choose to use IBM ILOG Solver and IBM ILOG Scheduler version 6.7 as they represent the commercial state-of-the-art and we did not want to unfairly penalize the CP approach due to using SCIP which has primarily been developed to this point as a MIP solver.

We used the scheduling instances introduced by Hooker (2005). Each set contains 195 problem instances. For both problem sets the number of resources ranges is from 2 to 4 and the number of jobs ranges from 10 to 38 in steps of 2. The maximum number of jobs for the instances with three and four resources is 32 while for two resources the number of maximum number of jobs is 38. For each problem size, we have five instances. For the MULTI problems the resource capacity is 10 and the job demands are generated with uniform probability on the integer interval $[1, 10]$. See Hooker (2005) for further details.

All computations reported were obtained on Intel Xeon E5420 2.50 GHz computers (in 64 bit mode) with 6 MB cache, running Linux, and 6 GB of main memory. We enforced a time limit of 7200 seconds. For all models other than CP, we use version 2.1.0.3 of SCIP integrated with SoPlex version 1.5.0.3 as the underlying linear programming solver (Wunderling 1996). Thus, we only used non-commercial software, with available source code.

Results

Tables 1 and 2 present the results for the UNARY test set and MULTI test set, respectively. The first two columns define the instance size in terms of the number of resources $|\mathcal{K}|$ and the number of jobs $|\mathcal{J}|$. For each model (for now we ignore the last four columns), we report the number of instance solved to proven optimality "opt" and the number instances for which we found a feasible solution "feas", which, of course, include the instances which are solved to optimality. We use the shifted geometric mean[2] for the number of "nodes" and for the running "time" in seconds.

The *shifted geometric mean* has the advantage that it reduces the influence of outliers. The geometric mean ensures that hard instances, at or close to the time limit, are prevented of having a huge impact on the measures. Similar shifting reduces the bias of easy instances, those solved in less than 10 seconds and/or less than 100 nodes. For a detailed discussion about different measures we refer to Achterberg (2007b).

For each resource-job combination, we display the best running time over all four models in bold face. In case one model could not solve any of the 5 instances for a particular resource-job combination, we omitted to display the shifted geometric mean of 7200.0 for the running time (instead we state "–").

UNARY On the UNARY problems, all four models are able to find feasible solutions for each instance. The CIP model finds and proves optimality for 194 out of 195 problem instances (timing-out on an instance with 30 jobs and 4 resources) followed by LBBD with 175, MIP at 161, and CP with 143. The CIP model is about three times faster than LBBD, using about half the number of nodes. However, note that the LBBD statistic includes only the nodes in the master problem search not the sub-problems. The time, however, includes both master and sub-problem solving.

The LBBD results are consistent with those of an existing implementation (not using SCIP) (Beck 2010). We found 20 time-outs for LBBD while Beck's results had 14 time-outs. We believe that this relatively small difference can be attributed to the use of different solvers and different computers.

Overall the CIP model dominates all other model for the UNARY case.

MULTI The MULTI results are somewhat different. CIP is not the dominant approach anymore. The performance of LBBD and CIP are very similar with respect to number of solved instances and overall running time. CIP manages 123 instances while LBBD solves 119. Both approach, however, are superior to the MIP and CP model using the measure of overall running time and number of solved instances to proven optimality. MIP solves 98 instances and CP solves only 62 instances.

This time, the LBBD results are not consistent with the previous implementation of Beck. He solved 175 instances which are 51 instances more than our LBBD model. We assume that using SCIP for solving the sub-problems instead of IBM ILOG Solver and IBM ILOG Scheduler leads to this differences. We plan to further investigate this issue.

The results of the CP model coincide with those of Hooker (2005) where it was shown that instances with 18 jobs or more could not be solved. The MIP results, in contrast, are substantially better than those reported by Hooker. The reason is not clear but we tentatively attribute the difference to the advance in MIP solving technology in the past six years.

Feasibility vs. Optimality It is of particular note that the MIP model was able to find feasible solutions for all instances in both problem sets. In fact, the optimality gap for MIP was usually very low: on the MULTI set the largest gap was 5.3% and 55 of the 97 instances not solved to optimality

[2] The shifted geometric mean of values $t_1, \ldots, t_n$ is defined as $\left(\prod(t_i + s)\right)^{1/n} - s$ with shift s. We use a shift $s = 10$ for time and $s = 100$.for nodes

Table 1: Results for the UNARY test set. Each resource-job combination consists of 5 instances. This adds up to a total of 195.

		MIP				LBBD				CP				CIP				CIP (optimality proof)			
$\|\mathcal{K}\|$	$\|\mathcal{J}\|$	opt	feas	nodes	time	opt	feas	nodes	time	opt	feas	nodes	time	opt	feas	nodes	time	inst	proved	nodes	time
2	10	5	5	1	**0.0**	5	5	61	0.2	5	5	96	**0.0**	5	5	25	**0.0**	5	5	1	0.0
	12	5	5	3	0.6	5	5	115	0.5	5	5	256	**0.0**	5	5	58	**0.0**	5	5	4	0.0
	14	5	5	93	2.4	5	5	566	1.5	5	5	741	**0.1**	5	5	130	**0.1**	5	5	4	0.0
	16	5	5	249	5.0	5	5	80	0.3	5	5	2 433	0.2	5	5	139	**0.1**	5	5	4	0.0
	18	5	5	221	11.0	5	5	75	0.3	5	5	6 767	0.5	5	5	216	**0.1**	5	5	8	0.0
	20	5	5	222	10.4	5	5	440	2.1	5	5	11 174	1.1	5	5	269	**0.2**	5	5	2	0.0
	22	5	5	16 898	139.5	5	5	195	1.6	5	5	47 460	4.8	5	5	117	**0.1**	5	5	1	0.0
	24	3	5	5 273	191.5	5	5	22	15.9	5	5	105 056	10.8	5	5	162	**0.1**	5	5	1	0.0
	26	5	5	13 359	174.8	4	4	300	33.7	5	5	326 528	34.8	5	5	439	**0.5**	5	5	18	0.0
	28	2	5	115 839	1813.9	5	5	510	28.9	5	5	497 455	60.0	5	5	346	**0.3**	5	5	1	0.0
	30	1	5	151 780	2432.7	4	4	1 836	74.2	5	5	4 231 229	490.2	5	5	2 139	**8.9**	5	5	114	1.2
	32	3	5	3 596	285.8	5	5	287	2.3	5	5	6 786 880	832.5	5	5	706	**0.8**	5	5	8	0.0
	34	1	5	125 588	2030.5	4	4	274	43.8	5	5	17 724 154	2239.7	5	5	897	**1.2**	5	5	9	0.0
	36	3	5	46 420	1322.9	1	1	656	2011.8	3	5	35 832 546	4939.0	5	5	1 014	**1.4**	5	5	7	0.0
	38	3	5	9 506	569.6	3	3	983	425.0	3	5	35 021 851	5151.3	5	5	1 076	**0.8**	5	5	1	0.0
3	10	5	5	1	0.1	5	5	356	0.6	5	5	584	**0.0**	5	5	73	0.1	5	5	15	0.0
	12	5	5	1	0.2	5	5	191	0.4	5	5	2 801	0.2	5	5	119	**0.1**	5	5	13	0.0
	14	5	5	7	1.6	5	5	2 759	5.0	5	5	13 431	1.0	5	5	314	**0.5**	5	5	68	0.2
	16	5	5	9	3.1	5	5	223	0.9	5	5	58 688	4.4	5	5	322	**0.4**	5	5	4	0.0
	18	5	5	7	2.6	5	5	444	**0.8**	5	5	236 227	20.3	5	5	632	1.4	5	5	24	0.1
	20	5	5	2 949	28.4	5	5	1 898	9.0	5	5	1 277 898	106.5	5	5	956	**3.0**	5	5	50	0.2
	22	5	5	602	29.6	5	5	1 106	12.2	5	5	9 746 557	873.9	5	5	1 217	**3.5**	5	5	51	0.2
	24	5	5	2 555	47.6	5	5	1 745	**5.7**	4	5	47 841 477	4432.3	5	5	1 641	6.7	5	5	48	0.3
	26	4	5	18 023	190.8	5	5	17 639	57.5	0	5	70 242 148	–	5	5	5 647	**22.6**	5	5	259	1.2
	28	4	5	2 715	100.4	5	5	3 721	**11.6**	0	5	70 336 374	–	5	5	4 591	18.5	5	5	72	0.3
	30	2	5	327 576	2904.1	3	3	11 963	133.8	0	5	68 045 013	–	5	5	18 901	**103.9**	5	5	605	12.1
	32	3	5	44 141	1209.6	4	4	6 228	95.6	0	5	70 408 953	–	5	5	10 677	**52.0**	5	5	268	7.8
4	10	5	5	1	0.1	5	5	262	0.7	5	5	1 166	0.1	5	5	29	**0.0**	5	5	4	0.0
	12	5	5	1	**0.1**	5	5	589	1.3	5	5	9 889	0.6	5	5	79	**0.1**	5	5	13	0.0
	14	5	5	1	**0.2**	5	5	2 390	5.8	5	5	44 482	3.4	5	5	211	0.4	5	5	15	0.1
	16	5	5	17	**2.4**	5	5	22 922	41.2	5	5	155 441	13.0	5	5	768	2.5	5	5	130	0.7
	18	5	5	1	**0.8**	4	4	9 857	61.4	5	5	2 006 248	163.8	5	5	904	2.9	5	5	73	0.4
	20	5	5	967	18.9	5	5	19 750	23.2	5	5	11 029 872	952.8	5	5	2 525	**8.7**	5	5	118	0.7
	22	5	5	10 364	95.9	3	3	222 873	245.1	3	5	52 404 067	5026.7	5	5	8 912	**44.4**	5	5	603	4.8
	24	5	5	2 343	55.4	4	4	43 687	70.5	0	5	81 261 130	–	5	5	7 355	**38.5**	5	5	209	2.4
	26	4	5	4 426	**119.3**	4	4	151 494	256.6	0	5	76 244 443	–	5	5	37 140	179.3	5	5	919	9.8
	28	3	5	137 997	1281.6	4	4	242 117	375.1	0	5	79 742 951	–	5	5	33 422	**175.1**	5	5	663	9.6
	30	3	5	187 957	2226.1	4	4	129 582	**129.4**	0	5	73 680 622	–	4	5	63 047	378.3	5	5	2 236	42.1
	32	2	5	15 488	831.6	4	4	526 447	**487.3**	0	5	71 281 227	–	5	5	73 517	491.8	5	5	642	5.5
		161	195	2 059	74.1	175	175	2 177	27.9	143	195	620 577	204.9	194	195	1 111	**9.9**	195	195	84	1.7

had gaps of less than 2%. From another perspective, therefore, there is an argument that MIP is performing best on the MULTI problems.

In general MIP solving, it is know that "good" primal solutions which are detected early in the search can significantly increase performance. A more detailed examination of the MIP runs shows that the primary reason for the ability of the MIP model to find feasible solutions is the use of such primal heuristics. In case of CIP, the default primal heuristics in SCIP are able to occasionally find feasible solution for the UNARY instances but never for MULTI.

To evaluate the viability of future research to develop CIP primal heuristics, we ran an additional experiment by providing the CIP model with the optimal value for all the instances in which it is known: all 195 instances for the UNARY problems and 176 instances for MULTI. The solver, then, has only to prove that the instance has no better solutions. If we are able to show that the proofs are short, we have an indication that, if we were able to develop strong primal heuristics for CIP, we would be able to substantially improve the problem solving performance. The final four columns in Tables 1 and 2 display the results. The "inst" columns is the number of instances where an optimal solution is known and the column "proved" is the number of instances that the CIP model was able to prove optimality having been given the optimal cost as an upper bound. The other two columns state again the shifted geometric mean of the number of search "nodes" and the running "time". For the UNARY instances, we are able to solve all problems, proving optimality in on average about a quarter of the time required to find and prove optimality. On the MULTI instances, however, the picture is different. Some instances (e.g., two resources and 30 or more jobs) show the same effect as for the UNARY case. On other instances it was possible to find and

Table 2: Results for the MULTI test set. Each resource-job combination consists of 5 instances. This adds up to a total of 195.

		MIP				LBBD				CP				CIP				CIP (optimality proof)			
$\|\mathcal{K}\|$	$\|\mathcal{J}\|$	opt	feas	nodes	time	opt	feas	nodes	time	opt	feas	nodes	time	opt	feas	nodes	time	inst	proved	nodes	time
2	10	5	5	99	1.4	5	5	51	0.2	5	5	2 792	0.1	5	5	152	**0.0**	5	5	35	0.0
	12	5	5	362	4.2	5	5	19	0.4	5	5	33 689	0.8	5	5	156	**0.0**	5	5	37	0.0
	14	5	5	613	13.2	5	5	6	2.7	5	5	466 774	11.5	5	5	342	**0.2**	5	5	200	1.7
	16	5	5	10 950	46.4	5	5	3	**17.0**	5	5	22 812 045	328.0	5	5	3 110	18.1	5	4	2 058	30.1
	18	5	5	55 304	153.9	5	5	7	88.1	0	3	494 257 496	–	5	5	9 951	**18.0**	5	5	2 563	29.0
	20	5	5	197 808	621.5	3	3	21	157.2	0	0	453 120 052	–	5	5	4 106	**2.4**	5	5	886	1.3
	22	3	5	971 481	3289.6	2	2	10	**702.9**	0	0	397 205 523	–	2	2	338 309	1324.1	4	2	29 718	371.6
	24	1	5	1 372 460	6912.5	0	0	1	–	0	0	339 570 055	–	3	3	353 667	**706.1**	5	2	14 242	508.7
	26	1	5	561 690	**3903.6**	1	1	1	5192.4	0	0	302 945 001	–	1	1	1 714 440	5439.1	5	2	15 689	508.5
	28	0	5	755 538	–	3	3	10	441.0	0	0	285 893 897	–	3	3	90 382	**159.6**	5	3	2 799	129.1
	30	0	5	537 011	–	1	1	1	**2971.1**	0	0	289 896 816	–	0	0	2 389 029	–	2	2	1	0.0
	32	1	5	447 812	5553.7	1	1	1	5679.1	0	0	285 549 706	–	3	3	494 763	**281.7**	3	3	1	0.0
	34	0	5	378 116	–	1	1	1	3014.7	0	0	291 878 950	–	1	1	2 729 498	**1396.4**	1	1	1	0.0
	36	0	5	459 463	–	1	1	1	2044.0	0	0	271 990 663	–	2	2	1 313 548	**699.3**	2	2	1	0.0
	38	0	5	202 726	–	1	1	1	3368.3	0	0	256 182 585	–	3	3	6 441 053	**1675.6**	3	3	1	0.0
3	10	5	5	7	0.8	5	5	49	0.2	5	5	1 508	**0.0**	5	5	85	**0.0**	5	5	19	0.0
	12	5	5	891	6.0	5	5	267	0.7	5	5	42 886	0.9	5	5	480	**0.3**	5	5	147	0.1
	14	5	5	575	8.8	5	5	94	**0.3**	5	5	536 284	10.0	5	5	1 152	1.1	5	5	234	0.2
	16	5	5	53 582	169.2	5	5	837	**9.9**	5	5	26 881 777	419.2	5	5	14 443	21.7	5	5	8 949	16.6
	18	4	5	312 463	952.5	5	5	3 196	**20.5**	0	2	485 036 163	–	4	4	201 128	138.6	5	5	75 534	53.7
	20	3	5	453 808	1629.4	5	5	1 613	**5.8**	0	1	460 840 309	–	5	5	37 498	34.8	5	5	5 612	38.3
	22	2	5	591 165	3117.1	5	5	2 254	**148.1**	0	0	431 132 614	–	2	2	632 677	1351.4	5	2	793 451	960.1
	24	3	5	577 419	5107.3	1	1	812	**2323.8**	0	0	410 966 318	–	1	1	1 660 912	3164.6	5	1	631 291	4709.2
	26	0	5	578 821	–	4	4	1 340	1350.8	0	0	370 925 295	–	3	3	650 086	**727.0**	5	2	264 430	1469.2
	28	0	5	418 861	–	0	0	8	–	0	0	337 254 114	–	2	3	725 799	**1260.3**	5	2	51 512	624.4
	30	0	5	292 470	–	0	0	49	–	0	0	288 175 928	–	0	0	1 382 511	–	4	1	55 610	1383.1
	32	0	5	95 543	–	0	0	3	–	0	0	265 357 963	–	1	1	1 747 365	**6917.5**	3	1	26 140	795.1
4	10	5	5	1	0.2	5	5	13	**0.1**	5	5	2 055	**0.1**	5	5	105	**0.1**	5	5	13	0.0
	12	5	5	544	5.1	5	5	30	**0.1**	5	5	30 179	1.0	5	5	242	0.2	5	5	32	0.0
	14	5	5	2 004	13.9	5	5	388	**1.1**	5	5	1 136 699	26.9	5	5	1 118	1.7	5	5	149	0.3
	16	5	5	1 539	29.3	5	5	251	**0.6**	5	5	6 907 787	111.8	5	5	1 094	1.3	5	5	290	0.3
	18	4	5	453 646	921.9	5	5	3 296	**3.5**	2	3	496 600 144	6994.2	5	5	28 359	20.1	5	5	2 118	4.0
	20	3	5	739 750	2188.5	5	5	1 298	**26.4**	0	1	479 264 386	–	5	5	34 834	28.6	5	5	9 271	3.9
	22	3	5	432 934	1957.6	5	5	3 363	**45.3**	0	0	471 200 720	–	4	4	77 455	166.3	5	3	7 883	135.6
	24	0	5	712 175	–	2	2	1 979	**1445.7**	0	0	413 825 956	–	2	2	1 796 900	3021.0	5	2	596 525	2190.7
	26	0	5	430 563	–	1	1	15 646	**4069.6**	0	0	399 153 422	–	0	1	2 436 109	–	5	0	2 952 736	7200.0
	28	0	5	479 513	–	1	1	679	**2803.2**	0	0	407 550 307	–	1	1	1 309 160	6575.0	5	1	158 262	1925.8
	30	0	5	237 592	–	1	1	186	**2105.0**	0	0	401 058 271	–	0	0	1 781 515	–	5	1	348 307	1982.5
	32	0	5	218 931	–	0	0	136	–	0	0	353 061 759	–	0	0	1 972 257	–	4	0	1 029 999	7200.0
		98	195	62 568	778.8	119	119	222	227.3	62	70	34 504 645	1311.1	123	125	61 323	**212.0**	176	125	5 656	83.2

prove optimality in our previous experiment, but not here when the optimal cost was given (e.g., an instances with 2 resources and 16 jobs). Clearly, the pattern of remaining open nodes and the conflict clauses learning in finding the optimal solution in the previous experiment helps in also proving it.

Overall, these results indicate that strong primal heuristics would be a clear benefit on the UNARY problems and for some of the MULTI instances. However, it is also clear from the MULTI results that research is also needed to speed-up the proofs of optimality (e.g., to strengthen the linear relaxation to improve bounding).

Summary Overall, the best performing approaches in terms of finding and proving optimality are CIP and LBBD. The former clearly dominates on the UNARY instances while their performance is similar on the MULTI instances. We conclude, therefore, that the CIP model is the best non-decomposition based approach for solving these combined resource allocation/scheduling problems.

The performance of MIP is not as strong as the two top approaches but better than both CP and the MIP results reported by Hooker (2005). However, only MIP is able to find provable good feasible solutions for all instances.

The relatively poor performance of CP is surprising given its usual success in scheduling problems. We plan to investigate more informed CP search heuristics (Beck and Fox 2000).

Conclusion

We studied four models for solving combined resource optimization and scheduling using mixed integer programming, constraint programming, constraint integer programming, and logic-based Benders decomposition. Previous results indicated the logic-based Benders decomposition was

the dominant approach. Our results demonstrated that on problems with unary capacity resources, constraint integer programming is able to solve 194 out of 195 instances to optimality compared to only 175 for logic-based Benders decomposition. However, on problems with non-unary resource capacity, the picture is not as clear as logic-based Benders and constraint integer programming showed similar performance. The results for the logic-based Benders, however, are weaker than previous results (Beck 2010). Interesting, the mixed integer programming model is able to find feasible solutions with a small optimality gap to all problems instances in both sets, unlike all other techniques. The constraint programming model is the worse performing model over both problem sets. Based on these results, we conclude that constraint integer programming represents the state-of-the-art for non-decomposition based approaches to these problems.

There are a number of avenues for future work including the investigation of primal heuristics for constraint integer programming and ways to improve the search done in constraint programming. Most significantly, we believe that our results here demonstrate that constraint integer programming may be a promising technology for scheduling in general and therefore we plan to pursue its application to a variety of scheduling problems.

References

Achterberg, T., and Berthold, T. 2009. Hybrid branching. In van Hoeve, W.-J., and Hooker, J. N., eds., *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems (CPAIOR 2009)*, volume 5547 of *LNCS*, 309–311.

Achterberg, T.; Brinkmann, R.; and Wedler, M. 2007. Property checking with constraint integer programming. ZIB-Report 07-37, Zuse Institute Berlin.

Achterberg, T. 2007a. Conflict analysis in mixed integer programming. *Discrete Optimization* 4(1):4–20. Special issue: Mixed Integer Programming.

Achterberg, T. 2007b. *Constraint Integer Programming*. Ph.D. Dissertation, Technische Universität Berlin.

Achterberg, T. 2009. SCIP: Solving Constraint Integer Programs. *Mathematical Programming Computation* 1(1):1–41.

Bajestani, M. A., and Beck, J. C. 2011. Scheduling an aircraft repair shop. In *Proceedings of the Twenty-Fifth International Conference on Automated Planning and Scheduling (ICAPS2011)*. to appear.

Baptiste, P.; Pape, C. L.; and Nuijten, W. 2001. *Constraint-based Scheduling*. Kluwer Academic Publishers.

Beck, J. C., and Fox, M. S. 2000. Constraint directed techniques for scheduling with alternative activities. *Artificial Intelligence* 121(1–2):211–250.

Beck, J. C. 2010. Checking-up on branch-and-check. In Cohen, D., ed., *Principles and Practice of Constraint Programming – CP 2010*, volume 6308 of *LNCS*, 84–98.

Berthold, T.; Heinz, S.; Lübbecke, M. E.; Möhring, R. H.; and Schulz, J. 2010. A constraint integer programming approach for resource-constrained project scheduling. In Lodi, A.; Milano, M.; and Toth, P., eds., *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems (CPAIOR 2010)*, volume 6140 of *LNCS*, 313–317.

Berthold, T.; Heinz, S.; and Pfetsch, M. E. 2009. Nonlinear pseudo-boolean optimization: relaxation or propagation? In Kullmann, O., ed., *Theory and Applications of Satisfiability Testing – SAT 2009*, volume 5584 of *LNCS*, 441–446.

Berthold, T.; Heinz, S.; and Vigerske, S. 2009. Extending a cip framework to solve MIQCPs. ZIB-Report 09-23, Zuse Institute Berlin.

Debruyne, R., and Bessière, C. 1997. Some practicable filtering techniques for the constraint satisfaction problem. In *Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence (IJCAI97)*, 412–417.

Fazel-Zarandi, M. M., and Beck, J. C. 2011. Using logic-based benders decomposition to solve the capacity and distance constrained plant location problem. *INFORMS Journal on Computing*. in press.

Hooker, J. N., and Ottosson, G. 2003. Logic-based Benders decomposition. *Mathematical Programming* 96:33–60.

Hooker, J. N. 2005. Planning and scheduling to minimize tardiness. In van Beek, P., ed., *Principles and Practice of Constraint Programming – CP 2005*, volume 3709 of *LNCS*, 314–327.

Kovács, A., and Beck, J. C. 2011. A global constraint for total weighted completion time for unary resources. *Constraints* 16(1):100–123.

Marques-Silva, J. P., and Sakallah, K. A. 1999. GRASP: A search algorithm for propositional satisfiability. *IEEE Transactions on Computers* 48(5):506–521.

Pape, C. L.; Couronné, P.; Vergamini, D.; and Gosselin, V. 1994. Time-versus-capacity compromises in project scheduling. In *Proceedings of the Thirteenth Workshop of the UK Planning Special Interest Group*.

Queyranne, M., and Schulz, A. S. 1994. Polyhedral approaches to machine scheduling problems. Technical Report 408/1994, Departement of Mathematics, Technische Universitat Berlin, Germany. Revised 1996.

Terekhov, D.; Beck, J. C.; and Brown, K. N. 2009. A constraint programming approach for solving a queueing design and control problem. *INFORMS Journal on Computing* 21(4):549–561.

Vilím, P.; Barták, R.; and Čepek, O. 2005. Extension of $O(n \log n)$ filtering algorithms for the unary resource constraint to optional activities. *Constraints* 10(4):403–425.

Wunderling, R. 1996. *Paralleler und objektorientierter Simplex-Algorithmus*. Ph.D. Dissertation, Technische Universität Berlin.

Optimization of Partial-Order Plans via MaxSAT

Christian Muise and **Sheila A. McIlraith**
Dept. of Computer Science
University of Toronto
Toronto, Canada. M5S 3G4
{cjmuise,sheila}@cs.toronto.edu

J. Christopher Beck
Dept. of Mechanical & Industrial Engineering
University of Toronto
Toronto, Canada. M5S 3G8
jcb@mie.utoronto.ca

Abstract

Partial-order plans (POPs) are attractive because of their least commitment nature, providing enhanced plan flexibility at execution time relative to sequential plans. Despite the appeal of POPs, most of the recent research on automated plan generation has focused on sequential plans. In this paper we examine the task of POP generation by relaxing or modifying the action orderings of a sequential plan to optimize for plan criteria that promote flexibility in the POP. Our approach relies on a novel partial weighted MaxSAT encoding of a sequential plan that supports the minimization of deordering or reordering of actions. We further extend the classical least commitment criterion for a POP to consider the number of actions in a solution, and provide an encoding to achieve least commitment plans with respect to this criterion. Our partial weighted MaxSAT encoding gives us an effective means of computing a POP from a sequential plan. We compare the efficiency of our approach to a previous approach for POP generation via sequential-plan relaxation. Our results show that while the previous approach is proficient at producing the optimal *deordering* of a sequential plan, our approach gains greater flexibility with the optimal *reordering*.

1 Introduction

Partial-order planning reflects a least commitment strategy (Weld 1994). Unlike a sequential plan that specifies a set of actions and a total order over those actions, a partial-order plan (POP) only specifies those action orderings necessary to achieve the goal from the initial state. In doing so, a POP embodies a family of sequential plans – a set of linearizations all sharing the same actions, but differing with respect to the order of the actions.

The flexibility afforded by POPs makes them attractive for real-time execution, multi-agent taskability, and a range of other applications that can benefit from their least commitment nature (Veloso, Pollack, and Cox 1998; Weld 1994). However, in recent years research on plan generation has shifted away from partial-order planning towards sequential planning, primarily due to the success of heuristic-based forward-search planners. To regain the least commitment nature of POPs while leveraging fast sequential plan generation, it is compeling to examine the computation of POPs via sequential planning technology. Indeed this approach has been realized in the planner POPF (Coles et al. 2010), which generated a POP by searching in a heuristic-based forward-chaining manner.

Another possibility for leveraging the strengths of sequential planning is to generate a sequential plan with a state-of-the-art planner, and subsequently relax the plan to a POP. Removing ordering constraints from the sequential plan, referred to as a *deordering*, or allowing changes in the ordering constraints, referred to as a *reordering*, are approaches that have been theoretically investigated (Bäckström 1998). Unfortunately, finding the optimal deordering or reordering is NP-hard to solve, and difficult to approximate within a constant factor. Nevertheless, with the advent of powerful optimization techniques (such as MaxSAT), we can effectively solve many problems in practice.

In this paper we focus on the optimization problem of computing the minimum deordering and minimum reordering of a sequential plan treated as a POP. The minimum deordering of a POP minimizes the number of ordering constraints between actions, so long as the POP remains valid and no two actions have their order reversed. Similarly, a minimum reordering minimizes the number of ordering constraints, but has no restriction on which ordering constraints are forbidden. These two notions cover a natural aspect of least commitment planning – minimizing the ordering constraints placed on a POP. We extend this characterization to consider the number of actions in a plan. In the spirit of least commitment planning, we argue that a POP should first commit to as few actions as possible before committing to as few ordering constraints as possible. With the various notions of least commitment planning in mind (deordering, reordering, etc), we propose a set of criteria against which we evaluate our work. These include the number of actions and ordering constraints found in the transitive closure of a POP, and the number of linearizations the POP represents. The criteria serve as a measure of the flexibility of a POP.

Our approach is to use a family of novel encodings for partial weighted MaxSAT whose solution corresponds to an optimal least commitment plan. Unlike typical SAT-based planning techniques, we represent an action occurrence once, giving us a succinct representation for use with a modern MaxSAT solver. We compare our approach to an existing algorithm for relaxing a sequential plan, due to Kambhampati and Kedar (Kambhampati and Kedar 1994), and evaluate our approach empirically. We demonstrate the

efficiency of using our MaxSAT encoding to relax a sequential plan optimally, and demonstrate the strength of Kambhampati and Kedar's algorithm in computing a deording of a sequential plan.

We find that the existing algorithm is extremely proficient at computing the minimum deordering, matching the optimal solution in every problem tested. However, we find that the minimum reordering is usually far more flexible than the minimum deordering (having fewer ordering constraints and far more linearizations). Our approach to encoding the problem gives us the first technique, to the best of our knowledge, for computing the optimal reordering of a POP. We further see a benefit in the flexibility of a POP when we use the proposed extension to least commitment planning that considers the number of actions.

In the next section we provide background on the notation we use throughout the paper. We follow with a description of least commitment planning in Section 3, and then describe our encoding for the various forms of optimization criteria in Section 4. In Section 5 we describe the prior approach to relaxing a plan, and we present the experimental results in Section 6. We finish with a brief discussion and conclusion in Sections 7 and 8.

2 Background

Propositional Planning

For the purposes of this document, we restrict ourselves to STRIPS planning problems. In STRIPS, a planning problem is a tuple $\Pi = \langle F, O, I, G \rangle$ where F is a finite set of fluents, O is the set of operators, $I \subseteq F$ is the initial state, and $G \subseteq F$ is the goal state. We refer to a *complete state* (or just *state*) as a subset of F. We interpret fluents that do not appear in a complete state as being false in that state. We characterize an operator $o \in O$ by three sets: $PRE(o)$, the fluents that must be true in order for o to be executable; $ADD(o)$, the fluents that operator o adds to the state; and $DEL(o)$, the fluents that operator o deletes from the state. An *action* refers to a specific instance of an operator, and it inherits the PRE, ADD, and DEL sets of the matching operator. We say that an action a is *executable* in state s iff $PRE(a) \subseteq s$. A sequence of actions is a *sequential plan* for the problem Π if the execution of each action in sequence, when starting in state I, causes the goal to hold in the final state.

We will make use of two further items of notation with respect to a set of actions A. We define **adders**(f) to be the set of actions in A that add the fluent f (i.e., $\{a \mid f \in ADD(a)\}$), and we define **deleters**(f) to be the set of actions in A that delete the fluent f (i.e., $\{a \mid f \in DEL(a)\}$).

Partial-Order Plans

For this paper, we adopt the notation typically used by the partial-order planning community. With respect to a planning problem $\Pi = \langle F, O, I, G \rangle$, a partial-order plan (POP) is a tuple $P = \langle \mathcal{A}, \mathcal{O}, \mathcal{C} \rangle$ where $\mathcal{A}$ is the set of actions in the plan (all of which have a corresponding operator in O), $\mathcal{O}$ is a set of orderings between the actions in $\mathcal{A}$ (e.g., $(a_1 \prec a_2) \in \mathcal{O}$), and $\mathcal{C}$ is a set of causal links between the actions in $\mathcal{A}$ (Weld 1994). A causal link is an annotated ordering constraint where the annotation of the link is a fluent from F that represents the reason for that link's existence.

For a causal link $(a_1 \overset{f}{\prec} a_2) \in \mathcal{C}$, we can assume that $f \in ADD(a_1)$ and $f \in PRE(a_2)$. The ordering constraints found in $\mathcal{C}$ will always be a subset of the ordering constraints in $\mathcal{O}$, and we assume that $\mathcal{O}$ is transitively closed. Where convenient, we will ignore the set $\mathcal{C}$ and simply use $\langle \mathcal{A}, \mathcal{O} \rangle$ to represent a POP. We refer to a total ordering of the actions in $\mathcal{A}$ that respects $\mathcal{O}$ as a *linearization* of P. A POP provides a compact representation for multiple linearizations.

Intuitively, a POP is valid for a planning problem if it is able to achieve the goal. There are two related formal notions of what a valid POP consists of. The first, only referring to $\mathcal{A}$ and $\mathcal{O}$, says that a POP P is valid for a planning problem Π iff every linearization of P is a plan for Π. While simple and intuitive, this notion is rarely used to verify the validity of a POP since there may be a prohibitively large number of linearizations represented by the POP.

The second notion is slightly more involved, and refers to open preconditions and threats. An *open precondition* is a precondition p of an action $a \in \mathcal{A}$ that does not have an associated causal link (i.e., $\nexists a' \in \mathcal{A}$ s.t. $(a' \overset{p}{\prec} a) \in \mathcal{C}$). If a precondition is not open, we say that it is *supported*, and we refer to the associated action in the causal link as the *achiever* for the precondition.

A *threat* in a POP refers to an action that can invalidate a causal link due to the ordering constraints (or lack thereof). Formally, if $(a' \overset{p}{\prec} a) \in \mathcal{C}$, we say that an action a'' threatens the causal link if the following is true:

- We can order a'' between a' and a (i.e., $\{(a'' \prec a'), (a \prec a'')\} \cap \mathcal{O} = \emptyset$)
- The action a'' can delete p (i.e., $p \in DEL(a'')$)

The existence of a threat means that a linearization may exist that is not executable because one of the preconditions of an action in the linearization is not satisfied.

We will typically add two special actions to the POP, a_I and a_G, that encode the initial and goal states through their add effects and preconditions (i.e., $PRE(a_G) = G$ and $ADD(a_I) = I$). With this modification, we say that a POP $P = \langle \mathcal{A}, \mathcal{O}, \mathcal{C} \rangle$ is a valid POP for the planning problem Π iff it has no open preconditions and no causal link in the set $\mathcal{C}$ has a threatening action in $\mathcal{A}$. The two notions of POP validity are similar in the sense that if the second notion holds, then the first follows. If the first notion holds for $\mathcal{A}$ and $\mathcal{O}$, then a set of causal links $\mathcal{C}$ exists such that the second notion holds for $P = \langle \mathcal{A}, \mathcal{O}, \mathcal{C} \rangle$. Further details on these notions of validity can be found in (Russell and Norvig 2009).

Partial Weighted MaxSAT

In boolean logic, the problem of Satisfiability (SAT) is to find a True/False setting of boolean variables such that a logical formula referring to those variables evaluates to True (Biere et al. 2009). Typically, we write problems in Conjunctive Normal Form (CNF) which is made up of a conjunction of clauses, where each clause is a disjunction of

literals. A literal is either a boolean variable or its negation. A setting of the variables satisfies a CNF formula iff every clause has at least one literal that the setting satisfies.

The MaxSAT problem is the optimization variant of the SAT problem in which the goal is to maximize the number of satisfied clauses (Biere et al. 2009, Ch. 19). Adding non-uniform weights to each clause allows for a more natural representation of the optimization problem, and we refer to this as the weighted MaxSAT problem. If we wish to force the solver to find a solution that satisfies a particular subset of the clauses, we refer to clauses in this subset as *hard*, while all other clauses in the problem are *soft*. When we have a mix of hard and soft clauses, we have a partial weighted MaxSAT problem (Biere et al. 2009, Ch. 19.6).

In a partial weighted MaxSAT problem, only the soft clauses are given a weight, and a valid solution corresponds to any setting of the variables that satisfies the hard clauses in the CNF. An optimal solution to a partial weighted MaxSAT problem is any valid solution that maximizes the sum of the weights on the satisfied soft clauses.

3 Least Commitment Criteria

The aim of least commitment planning is to find flexible solutions that allow us to defer decisions regarding the execution of the plan. Considering only the ordering constraints of a POP, two appealing notions for least commitment planning are the *deordering* and *reordering* of a POP. Following (Bäckström 1998), we define these formally:

Definition 1. Let $P = \langle \mathcal{A}, \mathcal{O} \rangle$ and $Q = \langle \mathcal{A}', \mathcal{O}' \rangle$ be two POPs, and Π a planning problem. Then,

1. Q is a reordering of P wrt. Π iff P and Q are valid POPs for Π, and $\mathcal{A} = \mathcal{A}'$
2. Q is a deordering of P wrt. Π iff P and Q are valid POPs for Π, $\mathcal{A} = \mathcal{A}'$, and $\mathcal{O}' \subseteq \mathcal{O}$.

Recall that we assume the ordering constraints of a POP are transitively closed. We define the optimal deordering and optimal reordering as follows:

Definition 2. Let $P = \langle \mathcal{A}, \mathcal{O} \rangle$ and $Q = \langle \mathcal{A}', \mathcal{O}' \rangle$ be two POPs, and Π a planning problem. Then,

1. Q is a minimum deordering of P wrt. Π iff
   (a) Q is a deordering of P wrt. Π, and
   (b) there is no deordering $\langle \mathcal{A}'', \mathcal{O}'' \rangle$ of P wrt. Π s.t. $|\mathcal{O}''| < |\mathcal{O}'|$
2. Q is a minimum reordering of P wrt. Π iff
   (a) Q is a reordering of P wrt. Π, and
   (b) there is no reordering $\langle \mathcal{A}'', \mathcal{O}'' \rangle$ of P wrt. Π s.t. $|\mathcal{O}''| < |\mathcal{O}'|$

Note that we use $<$ rather than $\subset$ for 1(b) and 2(b) since the orderings in $\mathcal{O}'$ and $\mathcal{O}''$ may only partially overlap. We will equivalently refer to a minimum deordering (resp. reordering) as an *optimal* deordering (resp. reordering). In both cases, we prefer a POP that has the smallest set of ordering constraints. In other words, no POP exists with the same actions and fewer ordering constraints while remaining valid with respect to Π. The problem of finding the minimum deordering or reordering of a POP is NP-hard, and cannot be approximated within a constant factor unless $NP \in \text{DTIME}(n^{poly\ log\ n})$ (Bäckström 1998).

While the notion of a minimum deordering or reordering of a POP addresses the commitment of ordering constraints, in the spirit of least commitment planning we would like to commit to as few actions as possible. To this end, we provide an extended criterion of what a least commitment POP (LCP) should satisfy:

Definition 3. Let $P = \langle \mathcal{A}, \mathcal{O} \rangle$ and $Q = \langle \mathcal{A}', \mathcal{O}' \rangle$ be two POPs valid for Π. Q is a *least commitment POP* (LCP) of P iff Q is the minimum reordering of itself and there is no valid POP $\langle \mathcal{A}'', \mathcal{O}'' \rangle$ for Π such that $\mathcal{A}'' \subseteq \mathcal{A}$ and $|\mathcal{A}''| < |\mathcal{A}'|$.

Intuitively, we can compute the LCP of an arbitrary POP by first minimizing the number of actions, and then minimizing the number of ordering constraints.

It may turn out that preferring fewer actions causes us to commit to more ordering constraints, simply due to the interaction between the actions we choose. However, in practice we usually place a much greater emphasis on minimizing the number of actions in a POP, as, in the standard interpretation of a POP, every every action must be executed.

Following the above criteria we will evaluate the quality of a POP by the number of actions and ordering constraints it contains, as these metrics give us a direct measure of the least commitment nature of a POP. Another property of interest is a POP's *flexibility*, which provides a measure of the robustness inherent in the POP. We measure the flexibility, whenever computationally feasible, as the number of linearizations a POP represents.

As we have discussed earlier, verifying a POP's validity by way of the linearizations is not always practical. As such, we will not attempt to compute POPs that maximize the number of linearizations, but rather we will compute POPs that adhere to one of the above criteria: minimum deordering, minimum reordering, or LCP.

4 A Partial Weighted MaxSAT Encoding

To generate a POP, we encode the problem of finding a minimum deordering or reordering as a partial weighted MaxSAT problem. Solutions to the default encoding correspond to a LCP. That is, no POP exists with a proper subset of the actions, or with a proper subset of the ordering constraints. We add further clauses to produce encodings that correspond to optimal deorderings or reorderings.

We use two types of propositional variables: action variables and ordering variables. For every action $a \in \mathcal{A}$, the variable x_a indicates whether or not the action a appears in the POP. For every pair of actions $a_1, a_2 \in \mathcal{A}$, the variable $\kappa(a_1, a_2)$ indicates an ordering constraint between action a_1 and a_2 in the POP.

In a partial weighted MaxSAT encoding there must be a distinction between hard and soft clauses. We first present the hard clauses of the encoding as boolean formulae which we subsequently convert to CNF, and later describe the soft clauses with their associated weight. We define the formulae that ensure the POP generated is acyclic, and the ordering constraints produced include the transitive closure (here,

actions are universally quantified, and for formula (4) we assume $a_I \neq a_i \neq a_G$):

$$(\neg\kappa(a,a)) \quad (1)$$
$$(x_{a_I}) \wedge (x_{a_G}) \quad (2)$$
$$\kappa(a_i,a_j) \rightarrow x_{a_i} \wedge x_{a_j} \quad (3)$$
$$x_{a_i} \rightarrow \kappa(a_I,a_i) \wedge \kappa(a_i,a_G) \quad (4)$$
$$\kappa(a_i,a_j) \wedge \kappa(a_j,a_k) \rightarrow \kappa(a_i,a_k) \quad (5)$$

(1) ensures that there are no self-loops; (2) ensures that we include the initial and goal actions; (3) ensures that if we use an ordering variable, then we include both actions in the POP; (4) ensures that an action cannot appear before the initial action (or after the goal); and (5) ensures that a solution satisfies the transitive closure of ordering constraints. Together, (1) and (5) ensure the POP will be acyclic, while the remaining formulae tie the two types of variables together and deal with the initial and goal actions.

In contrast to the typical SAT encoding for planning problems, we do not require the actions to be placed in a particular layer. Instead, we represent each action only once and handle the ordering between actions through the κ variables.

Finally, we include the formulae needed to ensure that every action has its preconditions met, and there are no threats in the solution:

$$\Upsilon(a_j,a_i,p) \equiv \bigwedge_{a_k \in \textbf{deleters}(p)} x_{a_k} \rightarrow \kappa(a_k,a_j) \vee \kappa(a_i,a_k) \quad (6)$$

$$x_{a_i} \rightarrow \bigwedge_{p \in PRE(a_i)} \bigvee_{a_j \in \textbf{adders}(p)} \kappa(a_j,a_i) \wedge \Upsilon(a_j,a_i,p) \quad (7)$$

Intuitively, $\Upsilon(a_j,a_i,p)$ ensures that if a_j is the achiever of precondition p for action a_i, then no deleter of p will be allowed to occur between the actions a_j and a_i. Υ ensures that every causal link remains unthreatened in a satisfying assignment. Formula (7) ensures that if we include action a_i in the POP, then every precondition p of a_i (the conjunction) must be satisfied by at least one achiever a_j (the disjunction). $\kappa(a_j,a_i)$ orders the achiever correctly, while $\Upsilon(a_j,a_i,p)$ removes threats.

In order to achieve a POP that is least commitment, we prefer solutions that first minimize the actions, and then minimize the ordering constraints. We achieve this by adding a soft unit clause for every variable in our encoding. We weight the κ variables with a unit cost and weight the action variables high enough for the solver to focus on satisfying them first:[1]

- $w(\neg\kappa(a_i,a_j)) = 1, \ \ \forall a_i,a_j \in \mathcal{A}$
- $w(\neg x_a) = 1 + |\mathcal{A}|^2, \ \ \forall a \in \mathcal{A} \setminus \{a_I,a_G\}$

Note that the weight of any single action clause is greater than the weight of all ordering constraint clauses. Since the soft clauses are all unit clauses, we are able to use negation and solve the encoding with a MaxSAT procedure. A violation of any one of the unit clauses means that the solution includes the action or ordering constraint corresponding to the variable in the violated clause.

[1] In domains with non-uniform action cost we could replace the weight of 1 in the action clause with the cost of the action, allowing us to minimize the total cost of the actions in the POP.

Proposition 1. Given a planning problem Π and a valid POP $P = \langle \mathcal{A}, \mathcal{O} \rangle$, any variable setting that satisfies the formulae (1)-(7) will correspond to a valid POP for Π where the ordering constraints are transitively closed.

Proof sketch. We have already seen that the POP induced by a solution to the hard clauses will be a acyclic and transitively closed (due to formulae (1)-(5)). We can further see that there will be no open preconditions since we include a_G, and the conjunction of (7) ensures that every precondition will be satisfied when the POP includes an action. Additionally, there are no threats in the final solution because of formula (6), which will be enforced every time a precondition is met by formula (7). Since the POP corresponding to any solution to the hard clauses will have no open preconditions and no threats, the second notion of POP validity allows us to conclude that the POP will be valid for Π. □

Proposition 2. Given a planning problem Π and a valid POP $P = \langle \mathcal{A}, \mathcal{O} \rangle$, any valid POP $Q = \langle \mathcal{A}', \mathcal{O}' \rangle$, where $\mathcal{A}' \subseteq \mathcal{A}$ and $\mathcal{O}'$ is transitively closed, has a corresponding variable setting that satisfies formulae (1)-(7).

Proof sketch. The proposition follows from the direct encoding of the POP Q where $x_a = True$ iff $a \in \mathcal{A}'$ and $\kappa(a_i,a_j) = True$ iff $(a_i \prec a_j) \in \mathcal{O}'$. If Q is a valid POP, then it will be acyclic, include a_I and a_G, have all actions ordered after a_I and before a_G, and be transitively closed (satisfying (1)-(5)). We further can see that (6) and (7) must be satisfied: if (7) did not hold, then there would be an action a in the POP with a precondition p such that *every* potential achiever of p has a threat that could be ordered between the achiever and a. Such a situation is only possible when the POP is invalid, which is a contradiction. □

Proposition 3. Given a planning problem Π and a valid POP $P = \langle \mathcal{A}, \mathcal{O} \rangle$, a partial weighted MaxSAT solver will find a solution to the soft clauses and formulae (1)-(7) that minimizes the number of actions in the corresponding POP, and subsequently minimizes the number of ordering constraints.

Proof sketch. With $|\mathcal{A}|$ actions, there can only be $|\mathcal{A}|^2$ ordering constraints. Since every soft clause that corresponds to an ordering constraint has weight 1, the total sum of satisfying every ordering constraint clause will be $|\mathcal{A}|^2$. Since the weight of satisfying any action clause is greater than $|\mathcal{A}|^2$, the soft clauses corresponding to actions dominate the optimization criteria. As such, there will be no valid POP for Π which has a subset of the actions in P and has fewer actions than a solution that satisfies formulae (1)-(7) while maximizing the weight of the satisfied soft clauses. □

Theorem 1 (Encoding Correctness)**.** Given a planning problem Π, and a valid POP P for Π, a solution to our partial weighted MaxSAT encoding is a LCP for P.

Proof sketch. This follows from propositions 1, 2, and 3. □

Observe that (1)-(7) never use the sequential plan. An optimal solution to the encoding will correspond to a LCP. To enforce solutions that are minimum deorderings or reorderings, we introduce two sets of hard clauses.

All Actions For optimal deorderings and reorderings, we require every action to be a part of the POP. We consider a formula that ensures we use every action (and so the optimization works only on the ordering constraints). To achieve this, we simply need to add each action as a hard clause:

$$(x_a), \quad \forall a \in \mathcal{A} \tag{8}$$

Deordering For a deordering we must forbid any explicit ordering that contradicts the sequential plan. Assume our sequential plan is $[a_0, \cdots, a_k]$. We ensure that the computed solution is a deordering by adding the following family of hard unit clauses:

$$(\neg\kappa(a_j, a_i)), \quad 0 \leq i < j \leq k \tag{9}$$

Intuitively, (9) simply forbids any ordering that contradicts the orderings found in the transitive closure of the sequential plan, thus ensuring the solution is a deordering.

Due to space limitations, we refrain from proving the correctness of the two encoding extensions (8) and (9).

5 Relaxer Algorithm

We investigate the efficiency of an existing algorithm for relaxing a sequential plan to produce a deordering. Originally due to Kambhampati and Kedar (1994), the algorithm operates by removing ordering constraints from a sequential plan in a systematic manner. A heuristic guides the procedure, and as pointed out in (Bäckström 1998), the process does not provide any guarantee that the resulting POP is minimally constrained (that is, we may be able to remove further ordering constraints and the POP remains valid). Despite this lack of theoretical guarantee, we show later that the algorithm produces excellent results.

The intuition behind the algorithm is to remove any ordering $(a_i \prec a_k)$ from the sequential plan where a_i does not contribute to a precondition of a_k and a_i does not threaten a precondition of a_k (and vice versa). For example, consider the case where our sequential plan is $[a_1 \cdots, a_i, \cdots, a_k, \cdots, a_n]$ and $p \in PRE(a_k)$. The algorithm will keep the ordering $(a_i \prec a_k)$ only if leaving it out would create a threat for a precondition of one of the actions, or if a_i is the earliest action in the sequence where the following holds:

1. $p \in ADD(a_i)$: a_i is an achiever for p
2. $\forall a_j, i < j < k, p \notin DEL(a_j)$: p is not threatened.

Algorithm 1, which we will refer to as the *Relaxer Algorithm*, presents this approach formally. We use **index**(a,$\vec{a}$) to refer to the index of action a in the sequence $\vec{a}$.

If $\vec{a}$ is a valid plan, line 11 will evaluate to true before either line 8 evaluates to true or the for-loop at line 6 runs out of actions. That is, we know that an unthreatened achiever exists and the earliest such one is found.

The achiever is then added to the POP as a new causal link (line 14), and the for-loop at line 17 adds all of the necessary ordering constraints so the achiever remains unthreatened. Note that for any deleter found in this for-loop, either line 18 or 20 must evaluate to true.

Algorithm 1: Relaxer Algorithm

Input: Sequential plan, $\vec{a}$, including a_I and a_G
Output: Partial-order plan, $\langle \mathcal{A}, \mathcal{O}, \mathcal{C} \rangle$

1 $\mathcal{A} = set(\vec{a})$;
2 $\mathcal{O} = \mathcal{C} = \emptyset$;
3 **foreach** $a \in \mathcal{A}$ **do**
4 **foreach** $f \in PRE(a)$ **do**
5 $ach = Null$;
6 **for** $i = (index(a, \vec{a}) - 1) \cdots 0$ **do**
7 // Stop if we find a deleter of f
8 **if** $f \in DEL(\vec{a}[i])$ **then**
9 break;
10 // See if we have an earlier achiever
11 **if** $f \in ADD(\vec{a}[i])$ **then**
12 $ach = \vec{a}[i]$;
13 // Add the appropriate causal link
14 $\mathcal{C} = \mathcal{C} \cup \{(ach \overset{f}{\prec} a)\}$;
15 $\mathcal{O} = \mathcal{O} \cup \{(ach \prec a)\}$;
16 // Add orderings to avoid threats
17 **foreach** $a' \in$ ***deleters***$(f) \setminus \{a\}$ **do**
18 **if** $index(a', \vec{a}) < index(ach, \vec{a})$ **then**
19 $\mathcal{O} = \mathcal{O} \cup \{(a' \prec ach)\}$;
20 **if** $index(a', \vec{a}) > index(a, \vec{a})$ **then**
21 $\mathcal{O} = \mathcal{O} \cup \{(a \prec a')\}$;
22 return $\langle \mathcal{A}, \mathcal{O}, \mathcal{C} \rangle$;

After going through the outer loop at line 3, every action in the newly formed POP has an unthreatened causal link for each of its preconditions. We are left with a valid POP, as there are no open preconditions or causal threats.

6 Evaluation

We evaluate the effectiveness of using the partial weighted MaxSAT solver, minimaxsat1.0 (Heras, Larrosa, and Oliveras 2008), to optimally relax a plan using our proposed encoding. To measure the quality of the POPs we generate, we consider the number of actions, ordering constraints, and linearizations (whenever feasible to compute). Further, we investigate the effectiveness of the Relaxer Algorithm to produce a minimally constrained deordering.

For our analysis, we use six domains from the International Planning Competition (IPC)[2] that allow for a partially ordered solution: Depots, Driverlog, Logistics, TPP, Rovers, and Zenotravel. These domains demonstrate both the strengths and weaknesses of our approach. We conducted the experiments on a Linux desktop with an eight-core 3.0GHz processor. Each run was limited to 30 minutes and 2GB of memory.

We generated an initial sequential plan by using the FF planner (Hoffmann and Nebel 2001). The encodings provided in Section 4 were converted to CNF using simple distributive rules so they may be used with the minimaxsat1.0 solver. We investigated the possibility of using a starting solution from the POPF planner (Coles et al. 2010), but

[2] `http://ipc.icaps-conference.org/`

Domain	Num Probs	FF Solved	Successfully Encoded
Depots	22	22	11
Driverlog	20	16	15
Logistics	35	35	30
TPP	30	30	7
Rovers	20	20	19
Zeno	20	20	15
ALL	147	143	97

Table 1: Number of instances successfully encoded. We indicate the number problems per domain, the number of problems for which FF finds a sequential plan, and the number of problems that our approach can encode successfully.

found that POPF failed to solve as many of the problems as FF (albeit POPF handles a far richer set of planning problems). Of the problems that were mutually solved by FF and POPF, we found that the POPs produced by the POPF planner were quite over-constrained, having many more ordering constraints than necessary. Once relaxed (by way of a similar encoding), the optimal deorderings, optimal reorderings, and LCPs of both FF and POPF plans were very similar. That is to say, there is usually little difference between the POPs generated by relaxing a sequential FF plan and the POPs generated by relaxing the POP found by POPF. For this reason, we only present the results for FF.

With two extensions for the encoding (**A**ll **A**ctions and **D**e**O**rdering) we have 4 possible variations for every instance. We will be primarily concerned with the following combinations for the settings:

- $\neg$ **AA**, $\neg$ **DO**: No additional clauses correspond to the default encoding where an optimal solution gives us a least commitment POP. We denote this setting as **LCP**.
- **AA**, $\neg$ **DO**: When we require all of the actions, solutions correspond to a minimum reordering of the sequential plan. We denote this setting as **MR**.
- **AA**, **DO**: When we require all of the actions, and a deordering, solutions correspond to a minimum deordering of the sequential plan. We denote this setting as **MD**.

In the following evaluation, we only report on the problems where FF was able to find a sequential plan. We found that many problems in the TPP and Depots domains are too large to encode in CNF. In these cases, we found that the combinatorial explosion for converting formulae (6) and (7) to CNF caused the encoding size to become too large. In Table 1 we present the number of problems per domain, solved by FF, and successfully encoded. Below we discuss a potential solution to dealing with this drawback.

POP Quality We begin by examining the relative quality of the POPs produced with different optimization criteria (LCP, MR, and MD), as well as the Relaxer Algorithm (abbreviated as RX). We report the number of actions and ordering constraints in the generated POP. Since the number of actions for RX, MR, and MD are equal to the number of actions in the sequential plan, we report the value only for RX and LCP. Table 2 shows the results for all six domains on

Domain	$\lvert\mathcal{A}\rvert$		$\lvert\mathcal{O}\rvert$			
	RX	LCP	RX	MD	MR	LCP
Depots (10)	34.2	30.9	451.8	451.8	407.9	339.1
Driverlog (15)	27.5	26.5	332.6	332.6	326.9	297.3
Logistics (25)	59.8	59.3	906.3	906.3	883.5	894.0
TPP (5)	13.4	13.4	74.8	74.8	74.8	74.8
Rovers (17)	30.6	30.1	214.3	214.3	208.8	200.2
Zeno (15)	19.8	19.8	137.1	137.1	136.6	136.6
ALL (87)	36.0	35.2	439.5	439.5	425.8	414.1

Table 2: Mean number of actions and ordering constraints for the various approaches. Numbers next to the domain indicate the number of instances solved by all methods (and included in the mean).

the problems for which every approach succeeded in finding a solution (87 of the 97 successfully encoded problems).

There are a few items of interest to point out. First, columns 4 and 5 coincide perfectly. It turns out, perhaps surprisingly, that the Relaxer Algorithm is able to produce the optimal deordering in every case, even though it is not guaranteed to do so. Since the algorithm can only produce deorderings, this is the best we could hope for from the algorithm. Second, we see the number of ordering constraints for the LCP approach is greater than those for the MR approach (on average) in the Logistics domain. The reason for this is because POPs in the Logistics domain require more ordering constraints for a solution with slightly fewer actions. For example, in prob15 the MR solution has 100 actions and 2278 ordering constraints, while the LCP solution reduces the number of actions in the POP to 96 at the expense of requiring 2437 ordering constraints.

In general, the LCP has fewer actions and ordering constraints than the optimal reordering, which in turn has fewer ordering constraints than the optimal deordering. If the LCP has the same number of actions as the sequential plan, then the LCP and minimum reordering coincide. We can see this effect in the TPP and Zenotravel where the number of actions and ordering constraints for LCP and MR are equal.

Finally, we note that in 4 problems (1 from Logistics, and 3 from Rovers), we found that the number of actions and constraints for either the LCP or MR POP matched that of the Relaxer POP, but the number of linearizations differed. Further, the differences in linearizations were not in the same direction (some better, and some worse). While the number of ordering constraints in a POP (for a given number of actions) usually gives an indication of the number of linearizations for that POP, these 4 problems indicate that this is not always the case.

Encoding Difficulty To measure the difficulty of solving the encoded problems, we computed the average time minimaxsat1.0 required to find an optimal solution (since the timing results had such a high standard deviation, we include both the mean and median values). It should be noted that an initial solution was consistently produced almost immediately by minimaxsat1.0's pre-processing step (a stochastic local search that satisfies every hard clause and serves as a lower bound on the optimal solution). Table 3 shows the average solving time for each domain given a particular set-

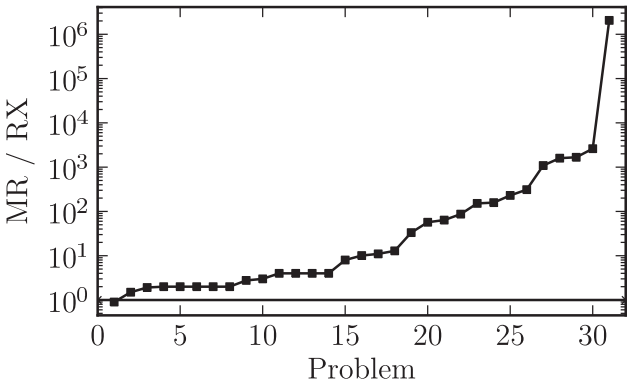

Figure 1: Ratio of Linearizations. The y-axis represents the number of linearizations induced by the POP for the optimal reordering divided by the number of linearizations induced by the POP for the optimal deordering. The x-axis ranges over all problems where the number of linearizations differed (∼40%), and is sorted based on the y-axis value.

ting. The largest solving time recorded was just under 500 seconds for a problem in the Rovers domain with the LCP setting. For comparison, we additionally include the average time the Relaxer Algorithm required to find a deordering.

For at least one of MR and LCP, 10 of the 97 problems successfully encoded proved too difficult for minimaxsat1.0 to solve, causing the solver to time out. We found that MD was on average easier to solve, but overall the majority of the problems were readily handled by minimaxsat1.0: 74% being solved in under 5 seconds. Being a polynomial algorithm, Relaxer consistently found a solution very quickly. The maximum time for a problem was just over 4 seconds.

Reordering Flexibility We have already seen that the Relaxer Algorithm is capable of producing the minimum deordering. To further evaluate the flexibility of the optimal deordering, we compare the number of linearizations induced by the optimal deordering with the number of linearizations induced by the optimal reordering. We found that of the 78 problems we could successfully compute the linearizations for, approximately 40% of the problems exhibited a difference between the optimal deordering and optimal reordering. Figure 1 shows the number of linearizations for the optimal reordering divided by the number of linearizations for the optimal deordering. For readability, we omit the 47 instances where the linearizations matched.

The ratio of linearizations ranges from 0.9 (an anomaly discussed below) to over two million. While the Relaxer Algorithm is proficient at finding the optimal deordering, this result demonstrates that there are still significant gains to be had in terms of flexibility by using the optimal reordering.

7 Discussion

The results paint an overall picture of how the optimization criteria compare to one another. We find that the Relaxer Algorithm is extremely adept at finding the optimal deordering, despite its lack of theoretical guarantee. In contrast, in many of the domains we see gains in terms of flexibility of the POP if we compute the optimal reordering or a least commitment POP. The encoding for the optimal deordering is easier for minimaxsat1.0 to solve compared to the optimal reordering or LCP, but the majority of problems for all optimization criteria were readily handled by minimaxsat1.0.

Whenever possible, we used the number of linearizations a POP represents as a measure of the POP's flexibility (this may not always be possible to compute due to the structure and size of the POP). We found that there are problems where the number of actions and ordering constraints in two POPs are equal, while the number of linearizations is not. Since the objective function of the encoding includes only the number of actions and ordering constraints, there is no guarantee on the number of linearizations that will result from a computed POP. An optimal reordering may even have fewer linearizations than an optimal deordering (which was observed in one case, as seen in Figure 1).

For a concrete example, consider two POPs on four actions $\mathcal{A} = \{a_1, a_2, a_3, a_4\}$. Ignoring causal links, Figure 2 shows the structure of the POPs P_1 and P_2. Both POPs have the same number of actions and ordering constraints, but the number of linearizations differ: P_1 has 6 linearizations while P_2 only has 5. These POPs serve as a basic example of how the LCP criteria does not fully capture the notion of POP flexibility. However, we should point out that fewer ordering constraints usually indicates more linearizations.

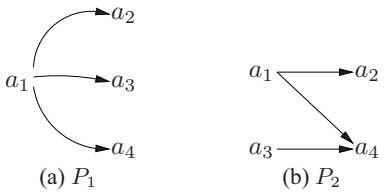

Figure 2: Two POPs with the same number of actions and ordering constraints, but different number of linearizations.

For the Depots and TPP domains, the encoding size became prohibitive. The larger formulae (i.e., (6) and (7)) are not too large in themselves, but when converting to CNF the theory becomes large. In future work, we plan to use the Tseitin encoding to convert the theory (Tseitin 1970). The Tseitin encoding will allow us to avoid the exponential blow-up in theory size, at the expense of introducing more variables. We also hope to investigate versions of partial weighted MaxSAT solvers tailored to problems in which only unit clauses are soft (as is the case with our encoding). There are other optimization techniques we plan on investigating, including constraint programming encodings, mixed-integer programming models, and a restricted form of partial-order causal link (POCL) search.

The encoding technique we have presented differs significantly from the standard SAT-based planning encodings. In particular, we avoid the need to encode an action for every layer in a planning graph by appealing to the fact that we already know the (superset of) actions that will be in the solution. The core of our encoding follows an approach similar

	Mean Time				Median Time				Mean	Mean # Clauses		
Domain	RX	MD	MR	LCP	RX	MD	MR	LCP	# Vars	MD	MR	LCP
Depots	0.54	89.63	55.63	19.72	0.44	1.26	1.55	1.71	1346.55	398205.91	397566.36	397532.64
Driverlog	0.58	1.35	38.36	6.02	0.55	0.77	0.70	0.87	943.87	39082.60	38638.20	38610.67
Logistics	1.16	9.40	127.66	105.02	0.80	3.13	5.69	4.98	7378.00	927265.59	923648.79	923576.59
TPP	0.87	3.58	171.71	171.68	0.61	0.44	0.50	0.50	1024.57	110964.86	110477.00	110452.57
Rovers	1.14	3.88	72.78	81.21	1.00	1.90	1.97	2.34	1428.74	111052.00	110371.11	110337.63
Zeno	0.60	0.86	0.85	0.87	0.46	0.42	0.53	0.43	541.73	18728.53	18477.47	18457.67
ALL	0.89	14.48	77.99	63.65	0.61	1.16	1.28	1.38	2972.67	364842.46	363397.60	363356.12

Table 3: Average time for the MaxSAT encoding to be solved by minimaxsat1.0, average time for the Relaxer Algorithm to compute a deordering, and average number of clauses and variables in the encoding. Mean and median values of run time are given in seconds. The mean is used to compute the average number of variables and clauses.

to Variant-II of Robinson *et al.* (Robinson et al. 2010). We similarly encode the ordering between any pair of actions as a variable ($\kappa(a_i, a_j)$), but rather than encoding a relaxed planning graph, we encode the formulae that need to hold for a valid POP. There are also similarities between our work and that of (Do and Kambhampati 2003). In particular, the optimization criteria for minimizing the number of ordering constraints coincide, as does the optional use of constraints to force a deordering. However, while Do and Kambhampati focus on temporal relaxation in the context of action ordering, we take the orthogonal view to minimize the number of actions required.

It is natural to consider the impact the choice of initial plan has on the final POP. As was mentioned earlier, the choice of initial solution between FF and POPF makes little difference in the quality of the optimally relaxed POP. The question remains open, however, on how to best compute an initial set of actions for our encoding.

8 Conclusion

In this paper we proposed a practical method for computing the optimal deordering and reordering of a sequential plan. Despite the theoretical complexity of computing the optimal deordering or reordering being NP-hard, we are able to compute the optimal solution by leveraging the power of modern MaxSAT solvers. We further propose an extension to the classical least commitment criterion that considers the number of actions in a solution, and demonstrate the added flexibility of a POP that satisfies this criterion.

Our approach uses a family of novel encodings for partial weighted MaxSAT where a solution corresponds to an optimal POP satisfying one of the three criteria we investigate (minimal deordering, minimal reordering, and our proposed least commitment POP). We solve the encoding with a state-of-the-art partial weighted MaxSAT solver, minimaxsat1.0, and find that the majority of problems are readily handled by minimaxsat1.0. We do, however, find that two domains present a problem for the encoding phase of our approach. In TPP and Depots, we find that the encoding size becomes too large to handle, which limits the applicability of our current encoding technique. In the future, we hope to employ the Tseiten encoding to limit this drawback.

We also investigated an existing algorithm for deordering sequential plans, and discovered that it successfully computes the optimal deordering in every problem we tested (despite its lack of theoretical guarantee). Since the algorithm is polynomial, and quite fast in practice, it is well suited for relaxing a POP if we require a deordering. However, if a reordering or least commitment POP is acceptable, then we can produce a far more flexible POP by using one of the proposed encodings.

Acknowledgements

The authors gratefully acknowledge funding from the Ontario Ministry of Innovation and the Natural Sciences and Engineering Research Council of Canada (NSERC). We would also like to thank the anonymous referees for useful feedback on earlier drafts of the paper.

References

Bäckström, C. 1998. Computational aspects of reordering plans. *Journal of Artificial Intelligence Research* 9(1):99–137.

Biere, A.; Heule, M.; van Maaren, H.; and Walsh, T. 2009. Handbook of Satisfiability, Frontiers in Artificial Intelligence and Applications, vol. 185.

Coles, A.; Coles, A.; Fox, M.; and Long, D. 2010. Forward-Chaining Partial-Order Planning. In *Twentieth International Conference on Automated Planning and Scheduling (ICAPS 10)*.

Do, M., and Kambhampati, S. 2003. Improving the temporal flexibility of position constrained metric temporal plans. In *AIPS Workshop on Planning in Temporal Domains (AIPS 03)*.

Heras, F.; Larrosa, J.; and Oliveras, A. 2008. MiniMaxSAT: An efficient weighted Max-SAT solver. *Journal of Artificial Intelligence Research* 31(1):1–32.

Hoffmann, J., and Nebel, B. 2001. The FF planning system: Fast plan generation through heuristic search. *Journal of Artificial Intelligence Research* 14(1):253–302.

Kambhampati, S., and Kedar, S. 1994. A unified framework for explanation-based generalization of partially ordered and partially instantiated plans. *Artificial Intelligence* 67(1):29–70.

Robinson, N.; Gretton, C.; Pham, D. N.; and Sattar, A. 2010. Partial weighted maxsat for optimal planning. In *Proceedings of the 11th Pacific Rim International Conference on Artificial Intelligence, Daegu, Korea, August 30 - September 02, 2010*.

Russell, S., and Norvig, P. 2009. *Artificial intelligence: a modern approach*. Prentice hall.

Tseitin, G. 1970. On the complexity of proofs in propositional logics. In *Seminars in Mathematics*, volume 8, 1967–1970.

Veloso, M.; Pollack, M.; and Cox, M. 1998. Rationale-based monitoring for planning in dynamic environments. In *Artificial Intelligence Planning Systems*, 171–179.

Weld, D. 1994. An introduction to least commitment planning. *AI Magazine* 15(4):27.

Exploiting MaxSAT for Preference-Based Planning

Farah Juma and **Eric I. Hsu** and **Sheila A. McIlraith**
Department of Computer Science
University of Toronto
{fjuma, eihsu, sheila}@cs.toronto.edu

Abstract

In this paper, we explore the application of partial weighted MaxSAT techniques for preference-based planning (PBP). To this end, we develop a compact partial weighted MaxSAT encoding for PBP based on the SAS$^+$ formalism. Our encoding extends a SAS$^+$ based encoding for SAT-based planning, SASE, to allow for the specification of simple preferences. To the best of our knowledge, the SAS$^+$ formalism has never been exploited in the context of PBP. Our MaxSAT-based PBP planner, MSPLAN, significantly outperformed a STRIPS-based MaxSAT approach for PBP with respect to running time, solving more problems in a few cases. Interestingly, when compared to SGPlan$_5$ and HPLAN-P, two state-of-the-art heuristic search planners for PBP, MSPLAN consistently generated plans with comparable quality, slightly outperforming at least one of these two planners in almost every case. For some problems, MSPLAN was competitive with HPLAN-P with respect to running time. Our results illustrate the effectiveness of our SASE based encoding and suggests that MaxSAT-based PBP is a promising area of research.

1. Introduction

Many real-world planning problems consist of both a set of mandatory goals and an additional set of desirable plan properties. The degree of satisfaction of these desirable properties, or plan *preferences*, determines the quality of a plan. Preference-based planning (PBP) (e.g., (Baier and McIlraith 2008)) extends the well-known classical planning problem in order to generate plans that achieve problem goals while maximizing the satisfaction of other preferred properties of the plan. In so doing, they allow a planner to generate plans of high quality, often under situations with conflicting preferences.

PBP has been the subject of substantial research in recent years. The 2006 International Planning Competition (IPC-2006) initiated a track on this topic which resulted in the extension of the standardized Planning Domain Description Language (PDDL) to support the specification of preferences (Gerevini et al. 2009). In PDDL3, desirable properties of a plan are expressed as preference formulae. These formulae may describe properties of the final state as well as properties that hold over intermediate states visited during plan execution. The relative importance associated with not violating these preference formulae is reflected in a metric function, a weighted linear combination of preferences whose violation is minimized by the planner. At IPC-2008, this family of planning problems was extended to include action costs. The objective of these so-called *net benefit* problems is to maximize the sum of the utilities of the goals and preferences that have been achieved, minus total costs. Action costs can be incorporated into our partial weighted MaxSAT-based PBP approach and it is something that we are exploring, but do not address it in this paper.

To date, the most effective techniques for PBP have been based on heuristic search (e.g., *Yochan*^{PS} (Benton, Kambhampati, and Do 2006), SGPlan$_5$ (Hsu et al. 2007), and HPLAN-P (Baier, Bacchus, and McIlraith 2009)). There have also been several planners that have used SAT, CSP, or Answer Set solvers (e.g., SATPLAN(P) (Giunchiglia and Maratea 2007), PREFPLAN (Brafman and Chernyavsky 2005), CPP (Tu, Son, and Pontelli 2007)). Most recently, Giunchiglia and Maratea (2010) explored a partial weighted MaxSAT-based approach to solving PBP problems by modifying a version of SATPLAN (Kautz 2006). For the purposes of this paper, we refer to this as GM. While all of these latter systems show promise, performance has generally been inferior to heuristic search.

In this paper, we characterize the problem of computing preference-based plans as a partial weighted MaxSAT problem. A major focus of our work is on how to construct an effective encoding. To this end, we propose a SAS$^+$ based (Bäckström and Nebel 1995) encoding of PBP as MaxSAT that is both compact and correct. Our encoding builds on the success of Huang, Chen, and Zhang (2010)'s SASE, a SAS$^+$ based encoding recently developed for SAT-based planning. To the best of our knowledge, the SAS$^+$ formalism has never been used in the context of PBP despite its effectiveness in classical planning. Exploiting our characterization of PBP as a partial weighted MaxSAT problem, we develop a system called MSPLAN that employs our SASE based encoding.

We experimentally evaluated our system by comparing it to Giunchiglia and Maratea's GM on *Simple Preferences* problems from IPC-2006. MSPLAN consistently outperformed GM with respect to running time, in some cases by an order of magnitude, solving some problems that GM could not solve. In all cases, plan quality was comparable. We also compared the performance of MSPLAN, run with two different MaxSAT solvers, to state-of-the-art heuris-

tic search planners for PBP. MSPLAN generated plans of comparable plan quality, sometimes slightly out- or underperforming the best of the heuristic search planners we employed. However, consistent with our expectations, MSPLAN was not able to solve as many problems as the heuristic search planners. While in some instances, one of the MSPLAN systems significantly outperformed HPLAN-P with respect to running time, the heuristic search planners were generally faster. Analysis showed that a significant part of MSPLAN's running time was attributed to the incremental construction of makespans, rather than the search for a solution. Given the consistent quality of MSPLAN solutions, consistent superior performance to GM, and the variability in the performance of all of these systems, we deem our SASE based encoding to be effective compared to a STRIPS encoding, and the use of MaxSAT and SAS$^+$ encodings for PBP to be promising areas of research.

2. PBP as Partial Weighted MaxSAT

In this section, we overview the correspondence between PBP and partial weighted MaxSAT. For the purposes of this paper, we restrict our attention to simple preferences – preferences over properties of the final state of the plan.

Definition 1 (Planning domain) Let X be a set of state variables. A planning domain is a tuple $D = (S, A, \gamma)$, where S is the set of all possible states; A is the set of actions; and γ is the transition function. A state $s \in S$ is a set of assignments to all of the variables in X. An action $a \in A$ is described by a tuple $(\mathit{pre}(a), \mathit{eff}(a))$, which denotes the action's preconditions and effects. A transition $\gamma(s, a)$ modifies the values of the state variables mentioned in $\mathit{eff}(a)$.

Definition 2 (Simple PBP problem) A simple PBP problem is a tuple $P = (D, s_I, s_G, \mathit{Pref}, W)$, where $D = (S, A, \gamma)$ is the planning domain; $s_I \in S$ is the (complete) initial state; s_G represents the goal and is a partial set of assignments to the state variables; and *Pref*, the preferences, is a partial set of assignments to the state variables. Optionally, each element in *Pref* may have a positive weight associated with it, W: *Pref* $\rightarrow \mathbb{R}_{\geq 0}$, to capture the relative importance of preferences.

We now define the partial weighted MaxSAT problem. Let ϕ denote a CNF propositional formula over a set V of boolean variables and let $\{C_1, \ldots, C_m\}$ denote the clauses of ϕ.

Definition 3 (MaxSAT and (Partial) Weighted MaxSAT) The *MaxSAT problem* is to find an assignment of values for V that maximizes the number of satisfied clauses in ϕ. Given a weight w_i for each clause C_i in ϕ, the *weighted MaxSAT problem* is to find an assignment of values for V that maximizes the total weight of the satisfied clauses in ϕ. When some clauses in ϕ are designated as *hard* clauses and other clauses in ϕ are designated as *soft* clauses and we are given a weight w_i for each *soft* clause C_i in ϕ, the *partial weighted MaxSAT problem* is to find an assignment of values for V that satisfies all of the designated *hard* clauses in ϕ and maximizes the total weight of the satisfied *soft* clauses in ϕ.

If *Pref*, the preferences in a simple PBP problem, are encoded as soft clauses, and s_I, s_G, and D are encoded as hard clauses, and weights drawn from W are assigned to the soft clauses to indicate the relative importance of the preferences, then from Definition 3, it follows that finding a solution to the resulting partial weighted MaxSAT problem is equivalent to finding a plan for the original PBP problem that achieves all of the hard constraints while maximizing the weight of the satisfied preferences. How we actually encode these clauses is a challenge and is one of the contributions of this paper.

3. Preference-Based Planning

3.1 A SAS$^+$ Based Encoding

The SAS$^+$ formalism (Bäckström and Nebel 1995) has been increasingly exploited in the context of *classical* planning. Unlike a STRIPS-based encoding, which consists of actions and facts and represents a state using a set of facts, a SAS$^+$ based encoding consists of transitions and multi-valued state variables and represents a state using a set of assignments to all of the state variables. A transition represents a change in the value of a state variable. For example, consider a transportation domain in which trucks can move packages between locations with certain restrictions. To represent the possible locations of a truck, a STRIPS-based encoding might include a fact for each such location (e.g., `(at truck1 loc1)`, `(at truck1 loc2)`). On the other hand, a SAS$^+$ based encoding might include a state variable `truck1` whose domain consists of all of the possible truck locations. Moving `truck1` from `loc1` to `loc2` would be represented by an action in a STRIPS-based encoding. However, in a SAS$^+$ based encoding, this would be represented by a transition that changes the value of the state variable `truck1` from `loc1` to `loc2`. In general, an action might result in changes in the values of multiple state variables. In a SAS$^+$ based encoding, this would be represented by multiple transitions. Because the number of transitions in an encoding based on the SAS$^+$ formalism is normally much less than the number of actions in a STRIPS-based encoding, the search space in a transition-based encoding is smaller than the search space in an action-based encoding (Huang, Chen, and Zhang 2010). The compactness of the SAS$^+$ formalism along with the rich structural information that can be derived from it are the major advantages of this formalism. As demonstrated by its use in (Helmert 2006), many state-of-the-art planners have been adopting SAS$^+$ based encodings.

The major limitation of previous STRIPS-based SAT encodings for classical planning problems has been the large number of clauses that are produced. The SAS$^+$ formalism was recently used for the first time in the context of a SAT-based classical planning approach with an impressive reduction in the size of the SAT encoding (Huang, Chen, and Zhang 2010). However, to the best of our knowledge, the SAS$^+$ formalism has never been used in the context of PBP despite its effectiveness in classical planning. We explore this here.

Translating PBP Problems to SAS$^+$ We consider PBP problems from the *Simple Preferences* track of IPC-2006. STRIPS problems can be compiled into the SAS$^+$ representation using the parser created for the Fast Downward plan-

ner (Helmert 2006). Since IPC-2006 *Simple Preferences* problems are non-STRIPS problems, we first need to compile them into STRIPS problems. We do so using the compilation technique developed in (Giunchiglia and Maratea 2010). The compilation proceeds as follows. For each simple preference, a so-called *dummy* action is created. The precondition of this dummy action is the simple preference itself and the effect of this action is a *dummy* literal that represents the simple preference. This dummy literal is then added to the definition of the goal for the problem instance and the definition of the simple preference is then removed from the problem. The execution of a dummy action indicates that the corresponding simple preference has been satisfied. For example, consider the following preference that is expressed in PDDL3:

```
(preference time
  (or (delivered pck1 loc3 t1)
      (delivered pck1 loc3 t2)))
```

The `time` preference specifies that `pck1` should be delivered to `loc3` at `t1` or `t2`. The compilation technique replaces this preference with the following dummy action:

```
(:action dummy-time
 :parameters ()
 :precondition (or (delivered pck1 loc3 t1)
                   (delivered pck1 loc3 t2))
 :effect (dummy-goal-time))
```

The literal `dummy-goal-time` represents the satisfaction of the `time` preference. After introducing dummy actions for each simple preference, the resulting problems are compiled into STRIPS using Gerevini and Serina's ADL2STRIPS tool. Note that we must also maintain the information about which goals in the resulting STRIPS problems correspond to simple preferences. These STRIPS problems can then be translated into the SAS$^+$ formalism.

After translating these STRIPS problems into SAS$^+$, there is a SAS$^+$ goal variable that corresponds to each dummy literal (i.e., simple preference). We will refer to these SAS$^+$ variables as preference variables. We will use s_P to denote the set of desired assignments to preference variables in the final state. As such, $s_P(x) = p$ indicates that in the final state, the desired value of the preference variable x is p.

Encoding the Clauses To encode a SAS$^+$ based PBP problem as a partial weighted MaxSAT problem, we modify the SAS$^+$ based SAT encoding, SASE, first proposed in (Huang, Chen, and Zhang 2010). SASE is made up of transition variables U, action variables V, and eight classes of clauses. We show these classes of clauses below. Note that N denotes the number of time steps in the plan, $\tau(x)$ denotes the set of possible transitions for x, $M(a)$ denotes the transition set of action a, R is the set of all prevailing transitions, $\delta_{f \rightarrow f'}$ represents a change in the value of x from f to f', $U_{x,f,f',t}$ is a transition variable that indicates that variable x changes from the value of f to a value of f' at time step t, and $V_{a,t}$ is an action variable that indicates that the action a is executed at time t.

1. Initial state - $(\forall x, s_I(x) = f)$: $\bigvee_{\forall \delta_{f \rightarrow g} \in \tau(x)} U_{x,f,g,1}$
2. Goal - $(\forall x, s_G(x) = g)$: $\bigvee_{\forall \delta_{f \rightarrow g} \in \tau(x)} U_{x,f,g,N}$
3. Transition's progression - $(\forall \delta^x_{f \rightarrow g} \in \tau$ and $t \in [2, N])$: $U_{x,f,g,t} \rightarrow \bigvee_{\forall \delta^x_{f' \rightarrow f} \in \tau(x)} U_{x,f',f,t-1}$
4. Transition mutex - $(\forall \delta_1 \forall \delta_2$ such that δ_1 and δ_2 are transition mutex): $U_{\delta_1,t} \rightarrow \neg U_{\delta_2,t}$
5. Existence of transitions - $(\forall x \in X)$: $\bigvee_{\forall \delta \in \tau(x)} U_{\delta,t}$
6. Composition of actions - $(\forall a \in O)$: $V_{a,t} \rightarrow \bigwedge_{\forall \delta \in M(a)} U_{\delta,t}$
7. Action's existence - $(\forall \delta \in \tau \backslash R)$: $U_{\delta,t} \rightarrow \bigvee_{\forall a, \delta \in M(a)} V_{a,t}$
8. Non-interference of actions - $(\forall a_1 \forall a_2$ such that $\exists \delta$, $\delta \in T(a_1) \cap T(a_2)$ and $\delta \notin R) : V_{a_1,t} \rightarrow \neg V_{a_2,t}$

These clauses encode the initial state, the goal state, the transitions that are allowed to occur at various time steps, the relationship between transitions and actions, and the fact that mutually exclusive actions cannot be executed simultaneously. After finding a sequence of transitions that achieves the goal, from the initial state, a corresponding action plan is found.

The key challenge is determining how to encode the preferences. We modify SASE to handle preference variables (i.e., variables that we would *like* to achieve a certain value in the final state) in addition to goal variables (i.e., variables that *must* achieve a certain value in the final state). To this end, we create a new class of clauses that represents the preferences. Specifically, for each preference variable x, we create a clause that is the disjunction of all transitions which result in the desired value for x in the final state. Using the notation described above, this new class of clauses can be defined as follows:

Preferences - $(\forall x, s_P(x) = p)$: $\bigvee_{\forall \delta_{f \rightarrow p} \in \tau(x)} U_{x,f,p,N}$

It is important to note that once a dummy action is executed, the dummy goal literal it produces persists indefinitely, i.e., no action deletes a dummy goal literal. Because the execution of a dummy action is meant to indicate that a simple preference is satisfied in the *final* state of the plan, we must ensure that once a dummy action has been executed, no other action that can invalidate the precondition of this dummy action can occur after it. As such, we restrict the execution of dummy actions, denoted by O_{dummy}, to the final time step of the plan by only defining dummy action variables that correspond to this time step, i.e.,

Dummy action variables - $V_{a,N}, \forall a \in O_{dummy}$

Because preference variables do not necessarily have to achieve their desired value in the final state, unlike hard goal variables, we treat all preference clauses as soft clauses. We treat the clauses that encode the initial state, goal state, and the planning domain, as hard clauses.

Weighting the Clauses We assign weights to all of the soft clauses using the PDDL3 metric function. This metric function is a linear function defined over simple preferences and is used to determine the quality of a plan. An example of

a PDDL3 metric function over two preferences named `time` and `loc` is shown below:

```
(:metric minimize(+
                  (* 10 (is-violated time))
                  (*  5 (is-violated loc))))
```

Note that the `is-violated` function returns 1 if the preference with the given name does not hold in the final state of the plan and returns 0 otherwise. This metric function indicates that satisfying the `time` preference is twice as important as satisfying the `loc` preference. An IPC-2006 *Simple Preferences* task can thus be viewed as the problem of finding a plan that satisfies all of the hard goals while minimizing the total weight of the preferences that are not satisfied (i.e., the task is to minimize the metric function). Since each simple preference is represented by one soft clause in our encoding, we assign a weight to this soft clause based on the weight assigned to this preference in the PDDL3 metric function from the original IPC-2006 problem instance. A similar approach was used to weight clauses in (Giunchiglia and Maratea 2010).

3.2 Planning with MaxSAT

With our SAS$^+$ based partial weighted MaxSAT encoding for PBP in hand, the next step is to determine how to find a plan using such an encoding. As in SAT-based planning, an incremental construction of makespans is required, i.e., for a specific value n for the makespan (the number of time steps in the plan), we must encode a given PBP problem into our SAS$^+$ based partial weighted MaxSAT encoding with a makespan of n, attempt to solve the problem using a partial weighted MaxSAT solver, and continue on in this manner, trying increasing values of n, until a solution is found. However, determining when a solution is found is not trivial, as discussed in the next subsection.

Stopping Conditions Because the task of PBP involves finding a high-quality plan, and not just a plan with the minimum number of time steps (as in SAT-based planning), determining when to stop trying increasing values for the makespan is more difficult. One possibility we consider is to stop trying increasing values for the makespan after the first satisfiable partial weighted MaxSAT formula is found. The solution to this formula corresponds to a plan with optimal makespan and optimal plan quality for that particular makespan. However, this plan is not guaranteed to be globally optimal. Consequently, there could still be a plan with a larger number of time steps but with better plan quality. We also consider the case where a time limit is given and an incremental construction of makespans is carried out until the allotted time expires, at which point the plan returned is the plan with the best quality that was found during the given amount of time. We will refer to this algorithm as the BEST algorithm.

Properties of the Plan We can show that for any fixed makespan, our approach is guaranteed to return a solution with optimal plan quality with respect to that particular makespan and furthermore, we can show that when restricted to plans with makespan bounded by k, our approach is guaranteed to return a solution that is k-optimal.

Lemma 1 *For any fixed makespan n, the solution to the partial weighted MaxSAT problem encoded with makespan n yields a plan with optimal quality with respect to the set of all plans with makespan n, if such a plan exists.*

Proof: Follows directly from the definition of the partial weighted MaxSAT problem (Definition 3). □

From Lemma 1, we can conclude that any plan P that is returned by our approach has optimal quality with respect to the set of all plans with the same makespan as P.

In certain cases, we may want to restrict our attention to plans with a makespan that is bounded by some value.

Definition 4 (k-Optimality) We can say that a partial weighted MaxSAT-based PBP algorithm is k-optimal if it is always able to find a plan that is optimal, in terms of quality, with respect to the set of all plans with makespan $i \leq k$.

Theorem 1 *If the search is restricted to plans with makespan bounded by k and the* BEST *algorithm is run long enough for the PBP problem to be encoded into a partial weighted MaxSAT formula using each makespan $i \leq k$, then the* BEST *algorithm is k-optimal.*

Proof: For each makespan $i \leq k$, the PBP problem will be encoded into a partial weighted MaxSAT formula with makespan i and if a solution to this formula is found, the quality of the resulting plan will be determined. From Lemma 1, each such plan will have optimal quality with respect to the set of all plans with makespan i. Now, the result follows since the BEST algorithm returns the plan with the best quality among the plans that were found. □

4. Implementation and Evaluation

We implemented our planner, MSPLAN, by extending the SASE planner. In order to compare the performance of MSPLAN to GM, we tried two of the partial weighted MaxSAT solvers which GM was evaluated with, namely, MiniMaxSAT v1.0 (Heras, Larrosa, and Oliveras 2007) and SAT4J v2.1 (Berre and Parrain 2010). We also attempted a comparison using MSUncore (Marques-Silva 2009), another partial weighted MaxSAT solver, but were unable to get a version of the system from the developers that would run on our hardware.

Most partial weighted MaxSAT solvers do not allow real-valued weights to be assigned to clauses but the PDDL3 metric function does allow for real-valued weights to be assigned to preferences. Thus, we must multiply real-valued weights in our encoding by an appropriate power of 10 in order to remove decimals from the weights. Additionally, many partial weighted MaxSAT solvers require a special weight to be specified for hard clauses in addition to weights for soft clauses. Following the convention used to specify this special weight, we assign a weight to each hard clause that exceeds the sum of the weights of all of the soft clauses.

Our evaluation of MSPLAN was motivated by two objectives. Specifically, we wanted to: (1) compare the performance of our planner to a previous partial weighted MaxSAT-based approach; and (2) compare our planner to state-of-the-art heuristic search planners for PBP. In doing so, we also hoped to gain some insight into the impact of the

underlying MaxSAT partial weighted MaxSAT solver on the performance of MSPLAN.

To support comparison with GM, the domains we evaluated were limited to those used in (Maratea 2010). Four domains from the IPC-2006 *Simple Preferences* track were used in our evaluation: trucks, storage, pathways, and openstacks. The trucks and openstacks domains contain both simple preferences and hard goals. However, the pathways and storage domains contain only simple preferences and it is generally not possible to satisfy all of these preferences. MSPLAN requires STRIPS-encodings of these problem instances as input. Since such encodings were also used in (Maratea 2010), we were able to obtain the problems from the *Simple Preferences* track which Giunchiglia and Maratea have been able to compile into STRIPS. The trucks and storage domains consist of 7 problems, the pathways domain consists of 20 problems, and the openstacks domain consists of 1 problem.

We compared the performance of MSPLAN to GM. GM uses a STRIPS-based partial weighted MaxSAT encoding as opposed to a SAS^+ based encoding. We also compared the performance of MSPLAN to that of the top heuristic search planners for PBP from IPC-2006, namely, $SGPlan_5$ (Hsu et al. 2007) and HPLAN-P (Baier, Bacchus, and McIlraith 2009). $SGPlan_5$ was the winner of all of the IPC-2006 *Simple Preferences* tracks. HPLAN-P did not formally compete in this track and came in 2nd place in the *Qualitative Preferences* track. Nevertheless, experiments performed in (Baier, Bacchus, and McIlraith 2009) show that HPLAN-P would have outperformed all entrants in the *Simple Preferences* track, other than $SGPlan_5$.

All of our experiments were performed on an AMD Opteron 1GHz processor. The memory usage in our experiments did not exceed 1GB. We ran our experiments with a timeout of 60 minutes. We were not able to obtain a copy of GM. As such, the results for these experiments are taken from (Maratea 2010). Different machines have thus been used in our comparison. In (Maratea 2010), experiments were performed on a Pentium IV 3.2GHz processor with 1GB of RAM, a faster machine than ours. Note that (Maratea 2010) gives only the time required for different partial weighted MaxSAT solvers to find a solution to the first satisfiable partial weighted MaxSAT formula. The total amount of time required to find a solution plan does not appear in (Maratea 2010).

Table 1 shows the performance of MSPLAN compared to GM when evaluating the time required for the solver to find a solution to the first satisfiable partial weighted MaxSAT formula when using MiniMaxSAT v1.0 and SAT4J v2.1 as the underlying solvers. The results show that for all problem instances in the trucks and storage domains which could be solved using both GM and MSPLAN, the time required to find a solution to the first satisfiable partial weighted MaxSAT formula was less, by an order of magnitude in many cases, when using MSPLAN than when using GM, regardless of the solver used. The value of the plan quality metric does not appear in (Maratea 2010) and we were unable to obtain the corresponding quality values from the authors. However, from a different encoding given to us by the authors, we were able to generate an upper bound on the quality values. This information indicated that the plan quality was comparable in all cases. This information also indicated that the number of clauses in our encoding was often significantly smaller, by a large constant factor. While this comparison is not precise, it is the best that could be done with the available data and gives an indication of the positive properties of our approach. There were problems in both the trucks and storage domains which could be solved by MSPLAN with a particular solver but could not be solved by GM when using the same solver. In fact, MSPLAN was able to optimally solve three problems in the trucks domain which GM could not solve using any of the partial weighted MaxSAT solvers evaluated in (Maratea 2010). Neither MSPLAN nor GM could generate a plan for the one problem instance in the openstacks domain. Finally, we were not able to do a direct comparison between these two planners for the pathways domain because we were not able to obtain information about the running time of GM on these problems.

	MiniMaxSAT		SAT4J	
Instance	GM	MSPLAN	GM	MSPLAN
trucks1	7.7	1.18	359.17	1.95
trucks2	308.92	44.803	-	24.868
trucks3	-	89.15*	-	446.578*
trucks4	-	128.904*	-	-
trucks5	-	652.877*	-	-
trucks6	-	-	-	-
trucks7	-	-	-	-
storage1	0.21	0.008	0.32	0.171
storage2	0.44	0.032	0.65	0.21
storage3	0.59	0.032	1.45	0.503
storage4	0.71	0.084	2.8	0.667
storage5	58.79	13.721	16.35	1.228
storage6	-	43.059	70.6	2.564
storage7	-	-	365.53	6.232

Table 1: Performance of MSPLAN compared to GM. Entries indicate the time required by MiniMaxSAT and SAT4J to find a solution to the first satisfiable partial weighted MaxSAT formula, in seconds. Dash entries indicate that the problem could not be solved during the given time. Starred entries indicate that the plan generated was optimal in terms of quality.

Table 2 shows the performance of MSPLAN compared to the two top heuristic search PBP planners from IPC-2006. In this set of experiments, we ran MSPLAN with the BEST algorithm that was described in Section 3.2i.e., the plan returned by MSPLAN was the highest quality plan found during a 60 minute period. We evaluated the total running time of all three planners and the quality of the plans found. The results show that for all but one of the problems that all three planners were able to solve, the quality of the plan generated by MSPLAN was equal to or was superior to the quality of the plan returned by at least one of $SGPlan_5$ and HPLAN-P, regardless of the partial weighted MaxSAT solver that was used. As expected, $SGPlan_5$ generated plans significantly faster than MSPLAN in all cases. However, MSPLAN solved some problems more quickly than HPLAN-P, while returning a plan of equal or better quality.

The significance of the $SGPlan_5$ comparison needs to be evaluated carefully. SGPlan has been shown to have in-

consistent performance on domains, depending on the encoding (Haslum 2007). Further, the version of SGPlan that participated in IPC-2008 was hand-tuned to the IPC problem encodings[1] and it is believed that previous versions of SGPlan were similarly hand-tuned. If this is the case, then the comparison between MSPLAN and SGPlan$_5$ is best interpreted as a comparison between a domain-independent MaxSAT-based preference-based planner and a manually domain-tuned heuristic search preference-based planner, and the generally (but not universally) superior performance of the manually domain-tuned system is to be expected. As such, a more reliable comparison is between MSPLAN and HPLAN-P since this heuristic search preference-based planner outperformed all other IPC-2006 Simple Preferences track competitors (Baier, Bacchus, and McIlraith 2009). Our comparison shows that MSPLAN is competitive with HPLAN-P (what we believe to be the top domain-independent heuristic search preference-based planner). We are currently undertaking a comparison with LAMA (Richter, Helmert, and Westphal 2008), a cost-optimal planner, by exploiting a translation of soft goals into action costs (Keyder and Geffner 2009).

As in SAT-based planning, a bottleneck in partial weighted MaxSAT-based planning appears to be the iteration required to determine the makespan for the solution. In our experiments, we encoded the problem using each possible makespan until the plan with the best quality was found. This is clearly a worst-case scenario. Many SAT-based planners first generate an estimate for the appropriate makespan and use this as a starting point for incremental search. As seen in Table 2, the time required for MSPLAN to generate a plan when given, by an oracle, the makespan that yields the plan with the best quality was typically much smaller than the time required by MSPLAN to generate a plan when this makespan was not known *a priori*. If an incremental approach could be created, whereby the partial weighted MaxSAT encoding of a problem with makespan k is extended, instead of being encoded from scratch, in order to generate the partial weighted MaxSAT encoding with makespan $k+1$, the running time of MSPLAN could likely be improved. A similar incremental approach has recently been investigated in the context of compiling PDDL planning problems into answer set programs (Knecht 2009).

Our planner appeared to be sensitive to the underlying MaxSAT solver. With MiniMaxSAT, MSPLAN was able to solve 12 of the 20 problems in the pathways domain but when SAT4J was used instead, 17 of the 20 problems were solved. Most of the pathways problem instances which could not be solved by MiniMaxSAT had a relatively large number of simple preferences (usually between 35 to 50 simple preferences). Since partial weighted MaxSAT solvers are believed to be very sensitive to the ratio of soft constraints versus hard constraints (Ansótegui, Bonet, and Levy 2009), the relatively larger proportion of soft constraints in these problems might explain the superior performance of SAT4J in the pathways domain. In contrast to the branch-and-bound framework of MiniMaxSAT, SAT4J forgoes the overhead of attempting to prune the search space by computing lower bounds. This decision pays off on underconstrained problems where such prunings are unlikely to trigger.

[1] See for example the function `search_ops_modal` in the file `Parser/inst_utils.c` in the SGPlan code located at `http://ipc.informatik.uni-freiburg.de/Planners` which contains hand-tuning for IPC domains.

5. Concluding Remarks

In this paper, we characterized the PBP problem as a partial weighted MaxSAT problem. We developed a compact encoding of PBP as partial weighted MaxSAT by building on the success of a SAS$^+$ based SAT encoding. To the best of our knowledge, this is the first time that the SAS$^+$ formalism has been used in the context of PBP. Our experimental evaluation showed that our MSPLAN system (with our SAS$^+$ encoding), consistently outperformed an existing MaxSAT-based planner (with a STRIPS encoding) with respect to running time, while generating plans of comparable quality. Remarkably, when run with two different MaxSAT solvers, MSPLAN generated plans with comparable quality to those generated by state-of-the-art heuristic search planners for PBP. Although the heuristic search planners were generally faster, there were problem instances for which at least one of the two MSPLAN systems ran significantly faster than HPLAN-P. The impressive performance of MSPLAN serves to illustrate the effectiveness of our SAS$^+$ encoding and suggests that both MaxSAT and SAS$^+$ encodings for PBP are worthy areas of continued exploration.

In this paper, we focused on simple preferences to support comparison with GM. Following the compilation technique described in (Baier and McIlraith 2006), it is possible to translate the full suite of PDDL3 qualitative preferences, including temporally extended preferences, into simple preferences. It is also possible to extend our work to net benefit problems by creating negated unary clauses that represent that an action is not executed at a particular time step and associating the satisfaction of these clauses with a weight corresponding to an action's cost. These are topics of current investigation. A current limitation of MSPLAN is that it is k-optimal rather than optimal. (SGPlan$_5$ is neither optimal nor k-optimal whereas HPLAN-P has the capacity to be k-optimal with respect to plan length and optimal if run to completion with certain restrictions on metric functions (Baier and McIlraith 2008).) Optimality may be achievable with a MaxSAT-based approach by exploiting a translation of soft goals into action costs (Keyder and Geffner 2009) and a cost-optimal planner based on partial weighted MaxSAT (e.g., (Robinson et al. 2010)).

Acknowledgements

We thank Enrico Giunchiglia and Marco Maratea for providing us with their STRIPS-encodings of the IPC-2006 *Simple Preferences* problems. We gratefully acknowledge funding from the Natural Sciences and Engineering Research Council of Canada (NSERC), the Ontario Ministry of Innovation, and MITACS.

Instance	MSPLAN: MiniMaxSAT				MSPLAN: SAT4J				HPLAN-P		$SGPlan_5$	
	Time (s)				Time (s)							
	Soln	Oracle	Total	M	Soln	Oracle	Total	M	Time (s)	M	Time (s)	M
trucks1	2.16	5.08	14.01	0	1.78	5.18	21.22	0	5.58	0	0.01	1
trucks2	36.51	41.3	165.91	0	36.04	37.66	173.63	0	92.28	0	0.04	0
trucks3	89.15	111.54	304.1	0	527.21	553.62	861.07	0	660.1	0	0.05	0
trucks4	128.9	426.26	596.18	0	-	-	-	-	563.93	3	0.08	0
trucks5	652.88	675.59	2553.01	0	-	-	-	-	1015.36	0	0.13	0
trucks6	-	-	-	-	-	-	-	-	-	-	0.27	0
trucks7	-	-	-	-	-	-	-	-	-	-	0.63	8
storage1	0.01	1.31	1.66	3	0.17	2.15	2.37	3	0.07	3	0.0	8
storage2	4.9	27.62	34.18	5	4.3	27.15	43.58	5	2.93	5	0.01	16
storage3	708.84	730.15	1192.33	18	1061.36	1083.39	2769.18	7	38.06	6	0.03	41
storage4	23.71	34.51	47.34	38	532.2	637.1	1063.3	24	183.37	9	0.06	49
storage5	70.08	123.29	214.77	107	677.02	705.96	937.9	76	76.57	94	0.13	136
storage6	43.06	71.49	73.19	173	1573.75	1605.23	1978.41	150	459.93	141	0.22	189
storage7	-	-	-	-	127.5	168.81	183.76	277	1280.43	160	0.32	242
pathways1	0.05	1.48	1.6	2	0.18	2.56	3.09	2	3.88	2	0.01	2
pathways2	0.07	0.99	1.29	3	0.46	1.99	3.34	3	186.34	4	0.0	3
pathways3	0.36	3.12	4.95	3	0.59	5.0	7.42	3	115.6	3.7	0.03	3
pathways4	0.36	5.33	5.85	2	0.78	6.25	8.66	2	324.82	2	0.03	2
pathways5	6.47	36.59	45.26	6	2.08	35.39	42.73	6	413.42	9	0.14	6.5
pathways6	434.02	797.73	1468.57	6.4	114.92	103.81	296.1	6.4	0.04	12.9	1.92	7
pathways7	1251.16	1466.24	1710.57	11.5	1378.79	1406.88	2325.46	10.3	0.05	12.5	1.93	10.4
pathways8	201.349	229.51	823.23	18.2	993.28	1020.99	1370.69	18	0.05	20.2	0.92	12.9
pathways9	-	-	-	-	2.33	31.50	32.31	15.7	0.07	15.7	1.4	10.6
pathways10	1182.02	1209.2	1744.95	12.9	775.77	803.84	1692.19	10.1	0.06	16.8	10.94	13.4
pathways11	5.56	32.88	37.85	11.8	1302.28	1332.93	2062.62	9.6	0.0	12.5	2.27	9
pathways12	37.29	47.72	62.02	18.8	2.81	28.77	29.88	18.8	0.04	18.8	14.93	15.4
pathways13	-	-	-	-	-	-	-	-	0.03	22	12.6	16
pathways14	55.28	62.81	80.5	20.7	942.9	1008.96	1092.89	20	0.03	20.7	7.15	15.6
pathways15	-	-	-	-	-	-	-	-	0.06	20.9	0.57	14.5
pathways16	-	-	-	-	660.60	693.78	1060.91	25.7	0.11	25.7	18.4	18.5
pathways17	-	-	-	-	582.82	603.99	615.12	22.3	0.1	22.3	41.94	20.3
pathways18	-	-	-	-	2.26	30.35	30.75	22.8	0.1	22.8	27.68	20
pathways19	-	-	-	-	-	-	-	-	0.05	26.5	41.24	22
pathways20	-	-	-	-	8.31	40.08	41.64	24.7	0.08	24.7	6.53	15
openstacks1	-	-	-	-	-	-	-	-	128.14	6	0.13	13

Table 2: Performance of MSPLAN when run with the BEST algorithm compared to the top PBP heuristic search planners from IPC-2006. Let B be the makespan that results in the plan with the best quality within the time limit. Soln denotes the time required for MiniMaxSAT and SAT4J to find a solution to the partial weighted MaxSAT formula with makespan B. This includes only the solver time to find the solution to the formula. Oracle denotes the total running time of MSPLAN when given, *a priori*, the best makespan B. This time includes the time for the translation to SAS^+, the time for encoding the partial weighted MaxSAT formula with makespan B, and the solver time. Total denotes the total running time when this best makespan B is not known ahead of time and must be determined through an incremental construction of makespans. This time includes the time for the translation to SAS^+, the time for repeatedly encoding a partial weighted MaxSAT formula with increasing makespans, and the total solver times. The total running time of HPLAN-P and $SGPlan_5$ is denoted by Time. M is the value of the plan metric, the total value of the violated preferences in the plan found. (Low is good.) A dash indicates timeout.

References

Ansótegui, C.; Bonet, M. L.; and Levy, J. 2009. Solving (weighted) partial MaxSAT through satisfiability testing. In *SAT*, 427–440.

Bäckström, C., and Nebel, B. 1995. Complexity results for SAS^+ planning. *Computational Intelligence* 11(4):625–655.

Baier, J. A., and McIlraith, S. A. 2006. Planning with first-order temporally extended goals using heuristic search. In *AAAI*, 788–795.

Baier, J. A., and McIlraith, S. A. 2008. Planning with preferences. *AI Magazine* 29(4):25–36.

Baier, J. A.; Bacchus, F.; and McIlraith, S. A. 2009. A heuristic search approach to planning with temporally extended preferences. *AIJ* 173(5-6):593–618.

Benton, J.; Kambhampati, S.; and Do, M. B. 2006. YochanPS: PDDL3 simple preferences and partial satisfaction planning. In *IPC-2006*, 54–57.

Berre, D. L., and Parrain, A. 2010. The SAT4J library, release 2.2. In *Journal on Satisfiability, Boolean Modeling and Computation*, volume 7, 59–64.

Brafman, R., and Chernyavsky, Y. 2005. Planning with goal preferences and constraints. In *ICAPS*, 182–191.

Gerevini, A.; Haslum, P.; Long, D.; Saetti, A.; and Dimopoulos, Y. 2009. Deterministic planning in the fifth international planning competition: PDDL3 and experimental evaluation of the planners. *Artificial Intelligence* 173(5-6):619–668.

Giunchiglia, E., and Maratea, M. 2007. Planning as satisfiability with preferences. In *AAAI*, 987–992.

Giunchiglia, E., and Maratea, M. 2010. A pseudo-boolean approach for solving planning problems with IPC simple preferences. In *COPLAS*, 23–31.

Haslum, P. 2007. Quality of solutions to IPC5 benchmark problems: Preliminary results. In *ICAPS*.

Helmert, M. 2006. The Fast Downward planning system. *JAIR* 26:191–246.

Heras, F.; Larrosa, J.; and Oliveras, A. 2007. MiniMaxSAT: a new weighted Max-SAT solver. In *SAT*, 41–55.

Hsu, C.-W.; Wah, B.; Huang, R.; and Chen, Y. 2007. Constraint partitioning for solving planning problems with trajectory constraints and goal preferences. In *IJCAI*, 1924–1929.

Huang, R.; Chen, Y.; and Zhang, W. 2010. A novel transition based encoding scheme for planning as satisfiability. In *AAAI*, 89–94.

Kautz, H. A. 2006. Deconstructing planning as satisfiability. In *AAAI*, 1524–1526.

Keyder, E., and Geffner, H. 2009. Soft goals can be compiled away. *JAIR* 36:547–556.

Knecht, M. 2009. Efficient domain-independent planning using declarative programming. Master's thesis, Hasso Plattner Institute, University of Potsdam.

Maratea, M. 2010. An experimental evaluation of Max-SAT and PB solvers on over-subscription planning problems. In *RCRA*, volume 616.

Marques-Silva, J. 2009. The MSUncore MaxSAT solver.

Richter, S.; Helmert, M.; and Westphal, M. 2008. Landmarks revisited. In *AAAI*, 975–982.

Robinson, N.; Gretton, C.; Pham, D. N.; and Sattar, A. 2010. Partial weighted MaxSAT for optimal planning. In *PRICAI*, 231–243.

Tu, P. H.; Son, T. C.; and Pontelli, E. 2007. CPP: A constraint logic programming based planner with preferences. In *LPNMR*, volume 4483 of *LNCS*, 290–296. Springer.

A SAT Compilation of the Landmark Graph

Vidal Alcázar
Universidad Carlos III de Madrid
Avenida de la Universidad, 30
28911 Leganés, Spain
valcazar@inf.uc3m.es

Manuela Veloso
Carnegie Mellon University
5000 Forbes Avenue
Pittsburgh, PA 15213
veloso@cmu.edu

Abstract

Landmarks are subgoals formed by sets of propositions or actions that must be achieved at some point in a planning task. These landmarks have a series of ordering relations between them that form what is known as the landmark graph. Previous works have used information from this graph with several purposes; however, few have addressed some of the shortcomings of the current representation of the graph, like landmarks having to be true at several time steps. In this work we propose a SAT encoding of the landmarks graph whose solution represents a more informative version of the original graph.

Introduction

Landmarks are disjunctive sets of propositions of which at least one component must be achieved or executed at least once in every solution plan to a problem (Hoffmann, Porteous, and Sebastia 2004). Currently landmarks are a very prominent line of research in automated planning, as the success of landmark-based planners like LAMA (Richter and Westphal 2010) shows. Most approaches have focused on using landmarks to partition the problem (Hoffmann, Porteous, and Sebastia 2004; Sebastia, Onaindia, and Marzal 2006) and to derive heuristics used in forward search planners (Richter and Westphal 2010; Karpas and Domshlak 2009; Bonet and Helmert 2010).

Finding the complete set of landmarks or even just proving that a proposition is actually a landmark is PSPACE-complete (Hoffmann, Porteous, and Sebastia 2004). However, current methods can efficiently compute a subset of the landmarks of the task based on a delete-relaxation representation of the problem. A series of partial orders between propositions can be computed with similar techniques too, which applied to landmarks are used to build the landmark graph. The landmark graph is the base of many of the techniques that employ landmarks, as the order in which landmarks should be achieved is often as important as finding the landmarks themselves.

Even though the landmark graph provides important information it still has several shortcomings. First, the partial orderings may not be enough to come up with a reasonable total order, which is critical for things such as partitioning the problem into smaller subproblems. Second, landmarks appear only once, even when it is clear that they must be achieved several times, like when they are causal preconditions of other landmarks that cannot be true at the same time. For example, in the well-known *Blocksworld* domain, *(arm-empty)* appears only once despite being necessary before every possible *(holding x - block)*, which cannot be true at the same time as any other proposition derived from the predicate *holding*. Third, the exploitation of the cycles and the unsound orderings in the graph is still unclear. In fact, most techniques that use landmarks are designed to be applied to acyclic graphs, so cycles are usually removed discarding unsound edges first before any kind of search begins.

An important remark is that the concept of time in a total order setting is absent in the landmark graph. In order to introduce time, landmarks must be able to appear as needed in different time steps. For this, they must be labeled with some sort of time stamp. The planning graph (Blum and Furst 1997) that is often used to represent the planning task does specifically this, as every proposition and action is represented several times by nodes labeled with the level they appear in. Hence, landmarks can be represented as several nodes, one per different possible time step. There is a similar relationship between the orderings in the landmark graph and the different edges of the planning graph: orderings are constraints that represent causal relationships of precedence between landmarks, and edges in the planning graph are constraints of either action-proposition causality or mutual exclusivity.

One of the most popular ways of exploiting the information contained in the planning graph is encoding it into a SAT problem (Kautz and Selman 1999). Using a SAT solver to find a solution to the encoding allows to find a corresponding solution plan to the problem or to prove that there is none for a given number of parallel time steps. The encoding consists in creating a variable per node in the planning graph and a clause per constraint between nodes. Given the similarity between the planning graph and the time-stamped landmark graph, this can also be done for the latter in order to obtain an enhanced version of the landmark graph that allows the possibility of landmarks being required at different time steps and that represents a more consistent total order than the one represented by the original graph. In addition, binary mutual exclusive relationships (mutex) will be introduced explicitly as another way of enforcing temporal constraints.

In this work we propose a SAT compilation for the land-

mark graph. First, some background regarding automated planning and landmarks will be given. Afterwards previous related work will be analyzed making emphasis on how the information of landmark graph has been exploited. The process of encoding the landmark graph into a SAT problem will be described next, going into detail for every feature relevant to the compilation. Some experimentation will be done to demonstrate the viability of the approach and finally some conclusions will be drawn and a few remarks on future lines of research will be presented.

Background

Automated planning can be described as the task of finding an ordered sequence of actions (commonly referred to as a plan) that achieves a set of goals from a given initial state. In this work only propositional planning is considered. A standard formalization of a planning problem is a tuple $P=(S,A,I,G)$ where S is a set of propositions, A is the set of grounded actions derived from the operators of the domain, $I \subseteq S$ is the initial state and $G \subseteq S$ the set of goal propositions. The actions that are applicable depend on several propositions being true and their effects can make propositions true of false, which is commonly known as *adding* and *deleting* the propositions, respectively. This way an action would be defined as a triple $\{pre(a), add(a), del(a)\}$ in which $a \in A$ and $pre(a), add(a), del(a) \in S$. Actions can have an associated cost; in this work only non-negative cost actions are used.

A planning graph is a directed leveled graph used to represent the constraints of a propositional planning problem. There are two kind of levels, *proposition levels* and *action levels*. Both kind of levels are alternated. Every node in the graph is labeled with the proposition or action it represents and the level (which actually represents a time step) it appears in. The first level is composed by the propositions $s \in I$. Subsequent levels contain propositions and actions that could be true at that level. For instance, the actions in level 2 are those applicable from the initial state, and the propositions in level 3 are the propositions that appeared in level 1 plus those achieved by the actions in level 2. Edges in the graph are derived from the triple $\{pre(a), add(a), del(a)\}$ that defines actions. There are also binary mutual exclusivity (mutex) relationships between nodes in the same level, which represent that both nodes cannot be true at the same time. Edges and mutexes represent the constraints of the planning problem.

Landmarks were initially defined as propositions that had to be true at some point in every solution plan to a problem. This concept was extended later to disjunctive sets of propositions and actions that had be respectively achieved or executed at some point (Richter and Westphal 2010) and more recently also to conjunctive sets of propositions (Keyder, Richter, and Helmert 2010). A general definition of landmark follows:

Definition 1 *A landmark is a logical formula* L *over either* S *(proposition landmark) or* A *(action landmark). Every solution plan must satisfy every action landmark. For each proposition landmark, at least one state in every sequence of states generated by a solution plan must satisfy it.*

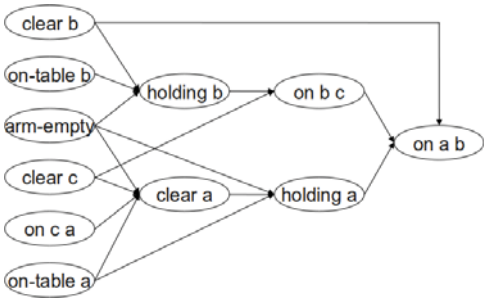

Figure 1: Simplified landmark graph of the Sussman's anomaly.

Orderings between facts are relations between two sets of propositions which represent the order in which they should be achieved in a given problem. There are the following orderings:

- Natural ordering: A proposition *a is naturally ordered before b* if *a* must be true at some time before *b* is achieved
- Necessary ordering: A proposition *a is necessarily ordered before b* if *a* must be true one step before *b* is achieved
- Greedy-necessary ordering: A proposition *a is greedy-necessarily ordered before b* if *a* must be true one step before *b* when *b* is first achieved
- Reasonable ordering: A proposition *a is reasonably ordered before b* if, whenever *b* is achieved before *a*, any plan must delete *b* on the way to *a*, and re-achieve *b* after or at the same time as *a*

Previous works classify reasonable orders as unsound (Hoffmann, Porteous, and Sebastia 2004), as not every solution plan has to achieve *a* before *b* if *b* is ever achieved. Obedient-reasonable orderings are a special case of reasonable orderings which arise when all previously computed reasonable orders are assumed to hold, although they will not be considered to this work.

The landmark graph is the directed graph composed by the proposition landmarks of a problem and the orderings between them. It is not acyclic, as necessary and reasonable orderings can induce cycles. Figure 1 shows the landmark graph of the Sussman's anomaly slightly simplified for the sake of clarity.

Related Work

In this section a series of relevant previous works that have dealt with the landmark graph will be discussed.

Partitioning using Disjunctive Landmarks

The first proposed way of using landmarks was partitioning the problem into several smaller ones (Hoffmann, Porteous, and Sebastia 2004). This has the advantage of potentially obtaining an exponential gain by reducing the depth of the

problem. It works by taking the leaves (landmarks not preceded by other unachieved landmarks) of an acyclic landmark graph and turning them into a disjunctive set of goals. When this goal is achieved, the landmark graph is updated and the search begins from the last state with a new subjunctive goal until all landmarks have been achieved.

Although this achieves important speedups in some cases, it does not take into account interactions between goals and is forced to eliminate cycles from the landmark graph. Besides, the total order is guessed in a rather random way, as the search will almost always achieve the closest landmark even if others belonging to the goal must come first.

Landmark Layering

A more elaborated way of partitioning the problem is computing layers of conjunctive landmarks, as done in the planning system STELLA (Sebastia, Onaindia, and Marzal 2006). In this case the motivation is the same, but mutexes are taken into account along with the orderings to build sets of conjunctive goals. Basically layers are built delaying landmarks that are mutex with other landmarks depending on whether their causal preconditions have been achieved or not. Besides, cycles are dealt with by breaking them and assuming that one landmark must be achieved twice, guessing which one and at which time through some intuitions.

This method is more informed, but it is not without disadvantages. First, its computation is substantially more complex, to the point that in some instances the planner may time out even before beginning the search. Besides, it is an ad hoc approach based on a series of intuitions that make its generalization complex.

Temporal Landmarks

With the extensions that appeared in PDDL2.1 (Fox and Long 2003) and later versions time can also be characterized in a planning task. This lead to research on the discovery of temporal landmarks (L. Sebastia 2007). In this case a temporal planning graph is built after computing the regular STRIPS landmarks and new temporal landmarks are found for a given horizon. The most interesting point is that in the same process both types of landmarks get ”activated” at some point taking into account mutexes and orders, which gives an intuition of when they may be needed for the first time and even allows to prove that there is no solution for a given horizon if some landmark could not be activated in time.

In this case the novel concept of landmarks being required at some time step regarding the problem and its possible solution plans is introduced. However, landmarks still appear as required only once, cycles and unsound orderings are ignored and the whole computation of the activation times depends on a given horizon that is chosen by hand and whose viability is unknown.

Landmarks in a CSP to prove Solvability

With the deadline constraints introduced in PDDL3.0 (Gerevini et al. 2009) a hard constraint on the horizon of the planning task can be imposed. A different way of using the landmarks was proposed to detect unsolvable problems due to these time constraints (Marzal, Sebastia, and Onaindia 2008). In this case the landmark graph was encoded as a CSP, in which the landmarks were the variables, the time steps where the landmarks might be needed for the first time the variables and the orderings, mutexes and deadlines the constraints. Then, if the CSP had no solution, the problem was deemed as unsolvable.

In this case a more general approach is taken. Even though the goal is proving unsolvability, probably the most interesting part is the new landmark graph obtained from a solution assignment.

A SAT Compilation of the Landmark Graph

The planning graph and the landmark graph are two conceptually close concepts. In fact, the planning graph is the core of the most promising landmark discovery methods so far (Keyder, Richter, and Helmert 2010). However, the relationship between the two has not been further analyzed. Landmarks are sets of propositions and actions, so they can be represented in the planning graph as well. The main difference is that landmarks do not contain information regarding time. When carrying landmarks over to the planning graph, it is clear that a landmark is a proposition or action that is true at at least one level out of all the possible levels in which that proposition or action may appear. Hence, the question of discovering at which levels they have to be true involves finding both when and how many times.

Another related concept are the constraints that constitute the edges in both graphs. These edges are causal relationships between the nodes of the graph that encode time constraints. Since time steps are not represented in the landmark graph, its edges are of a more general nature. However, they share most of their properties and can be exploited in similar ways. Following this intuition, it is interesting to analyze whether some of these commonalities allow applying techniques used in the planning graph to the landmark graph.

An interesting approach commonly employed in automated planning is the encoding of the planning graph as a SAT problem. This is done by planners like Blackbox (Kautz and Selman 1999), which use a SAT solver to find an assignment that corresponds to a valid solution plan. Despite being a relatively old concept, these planners still represent the state of the art in parallel-length optimal planning as they effectively take advantage of the techniques developed by the SAT community. The classical way of translating encodes every proposition and action at every level as a variable, and every constraint as a clause. Since the number of parallel steps that the optimal solution has is unknown, the initial number of levels is set to the minimum required for the goal propositions to appear without being mutex. Then, the planning graph is converted into a SAT problem and solved using a SAT solver. If no solution is found, the planning graph is extended by one level and the process is repeated until a solution is finally found or the planning graph levels off, also known as the ”ramp-up” method.

Inspired by this approach a SAT compilation of the landmark graph is proposed. In this case, the variables represent the landmarks being true at every time step, and the clauses their respective ordering constraints. Additionally

londex constraints (Chen, Xing, and Zhang 2007), which are a generalization of static binary mutexes, are added to the problem. This is because the landmark graph does not contain explicit information regarding mutual exclusion, as opposed to the planning graph. The previous computation of h^{max} from the initial state for every landmark is also required for two reasons: first, landmarks should not be encoded as variables for levels in which they can not appear, as there must be a minimum of parallel time steps before some propositions can be achieved; and second, to obtain a minimum horizon (number of levels) from which begin the method, which is the maximum h^{max} among all the landmarks.

The same method as with SAT-based planners is used. Once the landmark graph has been computed, a SAT compilation for a given number of time steps is computed. Then, a SAT solver is used to find a solution assignment, and, if none is found, the horizon is increased. An important difference is that in this case the "ramp-up" method is not complete in the sense that it is only the compilation and not the landmark graph which varies from level to level and hence there is no equivalent concept to the planning graph leveling off.

Clause Encoding

Clauses can be divided in three types: existential clauses, ordering clauses and londex clauses. In these clauses, *ini* represents the time step at which a given proposition can be true for the first time. Existential clauses represent the fact that the landmarks must be true at some time at least once. They have the following form:

- Existential clauses: Every landmark must be true at at least one time step ($a_{ini} \vee ... \vee a_n$)

In the particular case of propositions that are true in the initial state and goals these clauses are not necessary, as they always appear at least in the first and last level respectively. The variables that represent these landmarks must be introduced in the problem, though, as they may be necessary at different time steps. These propositions however can be used to simplify the problem by setting the variables whose value is known to true and applying unit propagation.

Ordering clauses are actually the edges of the landmark graph. There is a different clause per type of sound ordering:

- Natural orderings: *a* must be true at some time step before *b* is true ($a_{ini} \vee ... \vee a_{t-1} \vee \neg b_t$)
- Necessary orderings: Either *a* or *b* must be true at the time step before *b* is true ($a_{t-1} \vee b_{t-1} \vee \neg b_t$)
- Greedy-necessary orderings: Either *a* or *b* must be true at some time step before *b* is true ($a_{ini} \vee ... \vee a_{t-1} \vee b_{ini} \vee ... \vee b_{t-1} \vee \neg b_t$)

In this case a clause is needed for every edge and time step in which a proposition can appear as the supported proposition, unless it is not possible to create the clause. The latter can happen when the precondition propositions can not appear at the required time steps for being those lower than their h^{max} value, in which case the supported proposition is assigned the value of false. For example, if *a* is naturally ordered before *b*, for every time step in which *b* can appear a clause must be created; however, if $h^{max}(a) \geq t$ then *a* can not be true before b_t and so b_t gets automatically assigned the value of false.

Necessary orderings are worth of mention, as they can induce cycles in the landmark graph. In this case though cycles are not undesirable, as they are resolved implicitly when finding a solution assignment. This allows to find important structural information in the planning graph, such as loops or a producer-consumer relationship between propositions. For instance, in the aforementioned Blocksworld domain necessary orders allow to discover that *(arm-empty)* is required in every even time step, as it is necessarily ordered before every *(holding x - block)*.

Reasonable orders behave differently from the rest of the edge constraints. Reasonable orders are said to be unsound as they do not have to be respected in every solution plan. However, they can be considered sound when regression on the propositions is done. Reasonable orders hold among goal propositions (Koehler and Hoffmann 2000). This means that if *a* and *b* are goal propositions and *a* is reasonably ordered before *b*, the last time *a* is made true must come before the last time *b* is made true. When doing regression, this means that b must be supported first whenever both a and b are true if both must be true until the final state. In fact, this is true for every pair of reasonable ordered propositions whenever they are true in the same state. In terms of setting a constraint, this means that if *a* and *b* are true at the same time and *a* is reasonably ordered before *b*, at least *a* must be true in the previous time step, because if an action would have added either proposition it should have been *b*. Both propositions staying as true until the last level is represented by forcing them to be true after *t* in every level. They only case in which this is not true is when there exists an action that adds both propositions at the same time, but in that case current methods would not report a reasonable order between them (Richter and Westphal 2010). Their representation as a SAT clause in the encoding of a landmark graph with *n* time steps is the following:

- Reasonable orderings: If *a* and *b* are true at the same level, *a* must be true at the time step before that level ($a_{t-1} \vee \neg a_t \vee \neg b_t \vee \neg a_{t+i} \vee ... \vee \neg a_n \vee \neg b_{t+i} \vee ... \vee \neg b_n$)

Before defining londex clauses, some clarifications must be done. Londexes are a generalization of mutexes that introduce a relation of mutual exclusivity among several time steps. For example, *(holding a)* and *(holding b)* in the Blocksworld domain would form a londex of distance 2, as at least 2 actions are needed to achieve one of the propositions from a state in which the other is true. Seen as a constraint, this means that both propositions could not be true neither in the same time step nor in consecutive time steps. A fact that is not mentioned in the definition of londex is that londex are not necessarily symmetrical, in the sense that if *a* and *b* are mutex the distance to achieve *b* from a state in which *a* is true may not be the same as the distance to achieve *a* from a state in which *b* is true. For example, in an instance of the Sokoban domain with a single block and in which the grid is a n×n one with no obstacles, a proposition

that represents the block being at some corner of the grid and another one that represents the block being somewhere at the center are mutex. However, the minimum number of actions to move the block from the center to the corner is finite, whereas moving the block from the corner to the center is not possible at all (in which case the distance could be considered as infinite). The clauses derived from the londex must take into account this fact, so they are based on the distance between mutex propositions rather than in a single londex constraint:

- Distance between londexes: *a* cannot be true at a time step *t'* if *b* is true at *t* such that $t - distance(a, b) > t' \leq t$ $(\neg a_{t-(distance-1)} \vee \neg b_t) \wedge ... \wedge (\neg a_t \vee \neg b_t)$

Disjunctive and Conjuctive Sets of Landmarks and Action Landmarks

Landmark discovery methods are not limited to single proposition landmarks. Disjunctive and conjuctive sets of landmark propositions and action landmarks may also be found. Regarding action landmarks, it is easy to see that their preconditions and effects are proposition landmarks themselves. Therefore, actions can be easily encoded with an additional variable with constraints over those landmarks. These constraints can be represented by clauses equivalent to those used when encoding the planning graph as a SAT problem. Two differences exist: first, there are no action levels in the landmark graph, so we must assume that either the preconditions or the effects are true at the same time step than the action; second, preconditions and added propositions are landmarks, but deleted propositions are not, so the only case in which these constraint would be represented is when the action deletes a proposition that is a landmark itself and so appears in the landmark graph independently from the action. In this case a regular mutex clause would be used. These are the clauses if it is assumed that actions occur at the same time their effects become true:

- Existential action clauses: Every action landmark must be true at at least one time step $(a_{ini} \vee ... \vee a_n)$
- Precondition clauses: Precondition *p* must be true one time step before action *a* is true $(p_{t-1} \vee \neg a_t)$
- Add clauses: Added proposition *p* must be true at the same time step than action *a* whenever *a* is true $(p_t \vee \neg a_t)$
- Delete clauses: Deleted proposition *p* must be false if it is a landmark at the same time step than action *a* whenever *a* is true $(\neg p_t \vee \neg a_t)$

Landmark actions can introduce non-causal landmark propositions in the landmark graph, this is, propositions that are not ordered before any other landmark and are added only as a side effect in every solution plan. Although other approaches ignore these landmarks in this case they are not harmful and the additional londex constraints they introduce may prove useful to add additional restrictions to the SAT encoding.

The sets of conjunctive and disjunctive propositions can be represented with auxiliary variables. Existential, natural and greedy-necessary ordering clauses are built for these auxiliary variables, whereas necessary and reasonable orderings and londex clauses are built for the propositions that compose the set. Auxiliary variables and their composing propositions are represented in the following way:

- Disjunctive landmarks: At least one proposition p^i from those that compose the set *s* must be true whenever *s* is true $(p_t^0 \vee ... \vee p_t^n \vee \neg s_t)$
- Conjunctive landmarks: Every propositions p^i from those that compose the set *s* must be true whenever *s* is true $(p_t^0 \vee \neg s_t) \wedge ... \wedge (p_t^n \vee \neg s_t)$

Exploiting the Solution of the SAT Encoding

As described before, the SAT encoding is iteratively generated by increasing its horizon from an initial value. A SAT solver is used for every resulting subproblem until a solution assignment is found. First of all, it should be noted that the solution may not be unique. First, there may be several solutions for the same landmark graph encoding; second, if there is a solution for an encoding with a given number of levels *n*, there will be solutions for every encoding with a number of levels greater than *n*. This means that the obtained graph is not sound in the sense that the total order obtained does not have to be respected by every solution plan. Besides, in the case of disjunctive landmarks the propositions that appear as true are not necessarily those that support the disjunctive set for every solution plan.

On the other hand, the use of a Max-SAT solver that is able to minimize the number of true variables is encouraged in order to prevent unnecessary assignments to true. These unnecessary assignments to true may occur when there are variables whose values do not affect the satisfiability of the solution. Nevertheless, Max-SAT is harder than regular SAT and so not using it or settling with a local minimum on the number of true variables is in most cases enough to get a representative solution. Another possibility is computing the backbone of the SAT problem; this is more informative but it is also harder to compute and may leave some landmarks with no value at all if they can appear as true at different time steps in different solution assignments.

By solving the problem, a time-stamped landmark graph is obtained. This assignment can be exploited in several ways. The first relevant conclusion is that the number of levels of the graph is a lower bound on the parallel length of the original problem. When regular SAT-based planners that employ the ramp-up method are used this does not represent a great advantage, as their running time is dominated by proving that there is no solution at the level *n-1* when the optimal solution has *n* levels. However, for other approaches that are based on guesses over the minimal parallel length of the problem (Xing, Chen, and Zhang 2006; Rintanen 2010) this can be used to reduce the range of horizons considered.

Another advantage of the time-stamped landmark graph is that an intuition about the times steps at which a landmark may be necessary is obtained. This allows the construction of some sort of roadmap that can be used to guide the search. Closely related with this idea is the aforementioned partitioning of the problem by using layers of conjunctive land-

marks (Sebastia, Onaindia, and Marzal 2006), although in this case the layers can be built just by choosing the landmarks that appear as true at every particular level with no additional computation.

An important difference with the original graph is that here landmarks may appear as true several times. This can mean two things: first, a landmark must not only be achieved but also maybe kept as true across different levels; second, it is often the case that a landmark *l* appears as true at non-consecutive time steps t_i, t_j and there is another landmark *l'* mutex with *l* which appears as true at some level t' such that $t_i < t' < t_j$. If this can be proven for every solution plan this means that *l* must be achieved, deleted and the achieved again. In conjunction with landmark-based heuristics this would allow to obtain more informative values; in particular, admissible landmark-based heuristics would keep their admissibility while not being limited by the upper bound that h^+ imposes to admissible delete-effect relaxation heuristics.

An Example of Time-Stamped Landmark Graph

In Figure 1 a simplified landmark graph of the Sussman's anomaly is shown. Although the graph contains the most relevant propositions and orderings, it gives little insight on what the structure of the task may be like. In particular, the cycles induced by the necessary orders on *(arm-empty)* and which represent the concept of the arm as a common resource in Blocksworld are absent, apart from the lack of minimum number of levels. For this reason this problem has been chosen as an example of the proposed approach.

Table 1 represents the output of the encoding process. LAMA (Richter and Westphal 2010) was used to generate the landmark graph and domain transitions graphs were used for the londex computation. Changes were done in its algorithm, as LAMA does not compute necessary orders. Besides, cycles in the landmark graph were not removed. The table shows at which levels landmarks must be true to satisfy the constraints. The opposite is not true; a landmark does not have to be false when it appears as not required. That a landmark must be false may be useful in some cases, although this is trivially computable using the original constraints along with the solution.

This solution is a good example of how some of the shortcomings of the landmark graph can be overcome. First, the number of levels is the minimum required; second, trivial landmarks like *(arm-empty)* being required on every even level but the last one are detected; third, the positions at which landmarks appear as needed offer a great deal of information with regards to possible solutions. In this notable case, the optimal solution plan always respects the required landmarks at the exact times; the other way around, the only sequence of actions that would comply with the solution of the compilation of the landmarks graph is the optimal plan.

Experimentation

Although this work is centered around an alternative method of exploiting the information contained in the landmark graph as a departure point for novel approaches, an experimental part has been added for the sake of completeness.

Level:	0	1	2	3	4	5	6
(arm-empty)	x	-	x	-	x	-	-
(on a b)	-	-	-	-	-	-	x
(on b c)	-	-	-	-	x	x	x
(on c a)	x	-	-	-	-	-	-
(clear a)	-	x	x	x	x	-	-
(clear b)	x	-	x	-	-	x	-
(clear c)	x	-	x	x	-	-	-
(on-table a)	x	-	-	-	-	-	-
(on-table b)	x	-	-	-	-	-	-
(holding a)	-	-	-	-	-	x	-
(holding b)	-	-	-	x	-	-	-
(holding c)	-	x	-	-	-	-	-

Table 1: Output of the solution of the SAT problem generated from the landmark graph

In this experiments the landmark layering approach used by STELLA (described in subsection *Landmark Layering*) has been emulated in the following way:

- The landmark graph is compiled into a SAT problem.
- A solution for the resulting SAT encoding is found using the "ramp-up" method employed by conventional SAT solvers.
- The problem is partitioned by creating a series of ordered subgoals. Subgoals are sets of conjunctive landmarks that appear as needed in a given layer of the new landmark graph. These subgoals are ordered by the level they were extracted from.
- A base planner is used to solve the different subproblems. The initial state for every subproblem is the final state from the previous one. The final solution is the concatenation of the solution plans of all the subproblems.

The base planner is Fast-Downward (Helmert 2006) with greedy best-first search as the search algorithm, the FF heuristic and preferred operators with boosting enabled. Experiments have been done on a Dual Core Intel Pentium D at 3.40 Ghz running under Linux. The maximum available memory for the planner was set to 1GB, and the time limit was 1800 seconds. Only non-disjunctive propositional landmarks were used in the encoding of the landmark graph. No parameter setting was necessary.

Experiments were done using the same domains that were used when STELLA was compared to other planners: Blocksworld, Elevator, Freecell, Logistics, Depots, Driverlog, Satellite and Zenotravel from the second and the third international planning competitions. Mutexes were computed using the invariant discovery process of Fast-Downward. Since this method is unable to detect important mutexes in some domains, three domains were left out: Elevators, Depots and Freecell.

Regarding coverage, the results were exactly the same, that is, both approaches were able to solve the same instances. This is due to the fact that the base planner is able to solve all the instances and that the domains have no dead-ends, which may make the partitioning approach unable to

solve some problems due to its greediness. Therefore, we will focus on quality and number of expanded nodes to compare both approaches.

Table 2 shows the comparison between the base planner and the partitioning approach in terms of evaluated states and solution quality as length in the Blocksworld domain. Instances that were solved by both approaches expanding fewer than 100 were left out, as they are deemed to easy to contribute with relevant information. Results show that in many cases partitioning leads to an increase in quality. Instances in which the partitioning approach finds shorter plans usually require fewer nodes expansions for this approach as well. The opposite phenomenon can be observed too: when the base planner finds shorter plans the number of states is also smaller. In this case, the differences are greater, which explains why the average number of expanded states by the partitioning problem is considerably bigger.

Problem:	Base-S	Part-S	Base-Q	Part-Q
Prob-9-0	145	112	72	28
Prob-9-1	141	108	60	28
Prob-9-2	73	104	44	26
Prob-10-0	228	136	60	34
Prob-10-1	148	128	60	32
Prob-10-2	145	136	52	34
Prob-11-0	98	2994	54	72
Prob-11-1	327	628	104	64
Prob-11-2	150	3038	72	62
Prob-12-0	269	30836	74	125
Prob-12-1	131	248	102	52
Prob-13-0	678	284	108	68
Prob-13-1	391	674744	108	120
Prob-14-0	238	5968	88	140
Prob-14-1	268	390732	94	342
Prob-15-0	1504	11052	184	144
Prob-15-1	498	604	124	114
Prob-16-1	455	514	102	96
Prob-16-2	2006	29034	160	118
Prob-17-0	2424	190	280	48
Geometric Mean	299.91	1419.31	87.32	68.30

Table 2: Comparison between the base planner and the partitioning approach in terms of evaluated states (columns "Base-S" and "Part-S") and solution quality as plan length (columns "Base-Q" and "Part-Q").

In the other domains the results are not so conclusive. The geometric mean of the number of evaluated nodes and the plan length was computed for every domain and the ratio (the base planner mean divided by the partitioning approach mean) displayed. Table 3 shows these results. On average the number of expanded is greater, whereas quality remains the same. This is explained by two factors: first, the base planner is already very competitive and so there is little margin for improvement, and second the usage of preferred operators with the FF heuristic already guides the search towards unachieved landmarks, which overlaps with the usage of the landmark graph. A partitioning scheme with a planner based in other paradigms like LPG(Gerevini and Serina 2002) and VHPOP(Younes and Simmons 2003) would probably yield more positive results.

Problem:	Mean States	Mean Quality
Driverlog	0.89	1.06
Logistics	0.8	0.87
Satellite	0.45	1.01
Zenotravel	0.54	1

Table 3: Comparison between the base planner and the partitioning approach for the rest of the evaluated IPC-2 and IPC-3 domains

Conclusions and Future Work

In this work an alternative representation of the landmark graph has been proposed. We have discussed the shortcomings of the original landmark graph and analyzed previous related work that has exploited the information contained in it. The points in common between the planning graph and the landmark graph have been brought up and a parallelism between the SAT encoding of the constraints of both graphs has been presented. Finally, we have described the SAT encoding of the landmark graph and analyzed its characteristics.

We have also proposed several ways of exploiting the new landmark graph that can lead to future lines of research. Some of the information obtained after solving the SAT problem generated by the landmark graph, like the minimum number of levels, can be straightforwardly used with current techniques. The structure of the graph itself can also be exploited as it has been done in the experimentation section or in alternative ways, for example by using it as the seed for local search planners like LPG. Also the fact that landmarks can appear several times may lead to more accurate landmark-based heuristics for forward search planners both in satisfying and optimal search. Finally, generalizing the SAT encoding of the landmark graph for temporal and numeric domains by transforming it into a constraint satisfaction problem is also a promising continuation of this work.

Acknowledgments

This work has been partially supported by a FPI grant from the Spanish government associated to the MICINN project TIN2008-06701-C03-03.

References

Blum, A., and Furst, M. L. 1997. Fast planning through planning graph analysis. *Artif. Intell.* 90(1-2):281–300.

Bonet, B., and Helmert, M. 2010. Strengthening landmark heuristics via hitting sets. In *ECAI*, 329–334.

Chen, Y.; Xing, Z.; and Zhang, W. 2007. Long-distance mutual exclusion for propositional planning. In *IJCAI*, 1840–1845.

Fox, M., and Long, D. 2003. Pddl2.1: An extension to PDDL for expressing temporal planning domains. *J. Artif. Intell. Res. (JAIR)* 20:61–124.

Gerevini, A., and Serina, I. 2002. LPG: A planner based on local search for planning graphs with action costs. In *AIPS*, 13–22. AAAI.

Gerevini, A.; Haslum, P.; Long, D.; Saetti, A.; and Dimopoulos, Y. 2009. Deterministic planning in the fifth international planning competition: PDDL3 and experimental evaluation of the planners. *Artif. Intell.* 173(5-6):619–668.

Helmert, M. 2006. The Fast Downward planning system. *J. Artif. Intell. Res. (JAIR)* 26:191–246.

Hoffmann, J.; Porteous, J.; and Sebastia, L. 2004. Ordered landmarks in planning. *J. Artif. Intell. Res. (JAIR)* 22:215–278.

Karpas, E., and Domshlak, C. 2009. Cost-optimal planning with landmarks. In *IJCAI*, 1728–1733.

Kautz, H. A., and Selman, B. 1999. Unifying SAT-based and graph-based planning. In *IJCAI*, 318–325. Morgan Kaufmann.

Keyder, E.; Richter, S.; and Helmert, M. 2010. Sound and complete landmarks for and/or graphs. In *ECAI*, 335–340.

Koehler, J., and Hoffmann, J. 2000. On reasonable and forced goal orderings and their use in an agenda-driven planning algorithm. *J. Artif. Intell. Res. (JAIR)* 12:338–386.

L. Sebastia, E. Marzal, E. O. 2007. Extracting landmarks in temporal domains. In *International Conference on Artificial Intelligence (ICAI'07)*, volume 2, 520–525.

Marzal, E.; Sebastia, L.; and Onaindia, E. 2008. Detection of unsolvable temporal planning problems through the use of landmarks. In Ghallab, M.; Spyropoulos, C. D.; Fakotakis, N.; and Avouris, N. M., eds., *ECAI*, volume 178 of *Frontiers in Artificial Intelligence and Applications*, 919–920. IOS Press.

Richter, S., and Westphal, M. 2010. The LAMA planner: Guiding cost-based anytime planning with landmarks. *J. Artif. Intell. Res. (JAIR)* 39:127–177.

Rintanen, J. 2010. Heuristics for planning with SAT. In Cohen, D., ed., *CP*, volume 6308 of *Lecture Notes in Computer Science*, 414–428. Springer.

Sebastia, L.; Onaindia, E.; and Marzal, E. 2006. Decomposition of planning problems. *AI Commun.* 19(1):49–81.

Xing, Z.; Chen, Y.; and Zhang, W. 2006. Optimal strips planning by maximum satisfiability and accumulative learning. In Long, D.; Smith, S. F.; Borrajo, D.; and McCluskey, L., eds., *ICAPS*, 442–446. AAAI.

Younes, H. L. S., and Simmons, R. G. 2003. VHPOP: Versatile heuristic partial order planner. *J. Artif. Intell. Res. (JAIR)* 20:405–430.

A Constraint-based Approach for Planning and Scheduling Repeated Activities

Irene Barba and **Carmelo Del Valle**
Departamento de Lenguajes y Sistemas Informáticos
Avda Reina Mercedes s/n, 41012 Sevilla (Spain)
{irenebr,carmelo}@us.es

Abstract

The main purpose of this work entails the development of a constraint-based proposal, in order to solve problems which deal with the planning and scheduling of activities which can be executed several times and are related by high-level constraints. The proposed constraint-based approach includes new filtering rules for the definition of the high-level relations between the repeated activities, and thereby facilitates the specification of the problem through global constraints at the same time as it enables the efficiency in the search for solutions to increase. As an application of the current approach, the considered high-level constraints can be used, in a declarative way, to specify the relations given between the activities involved in business process (BP) environments. In fact, the high-level constraints considered in this work are taken and extended from an existing declarative language for BP, named ConDec. The main objective of this application is to provide business analysts with assistance in the generation of optimized BP models from declarative specifications, using AI P&S techniques. This interesting and innovative topic has yet to be addressed. Moreover, the proposed constraint-based approach can be used to efficiently solve further planning and scheduling problems unrelated to business process environments, which include similar relations between repeated activities.

1 Introduction

Constraint Programming (CP) has been evolved to a mature field because, among others, of the use of different generic and interchangeable procedures for inference and search, which can be used for solving different types of problems (Rossi, Van Beek, and Walsh 2006). The constraint-based algorithms can work together with filtering algorithms (propagators) for the constraints, which are in charge of removing inconsistent values. CP supplies a suitable framework for modelling and solving problems involving planning and scheduling (P&S) aspects (Salido 2010).

The main purpose of the current work entails the development of a constraint-based approach, in order to solve problems which deal with the P&S of activities which can be executed several times and which are related by high-level constraints. The proposed constraint-based approach includes new filtering rules for the definition of the high-level relations between the repeated activities, and thereby facilitates the specification of the problem through global constraints at the same time as it enables the efficiency in the search for solutions to increase. It should be emphasized that the developed propagators deal with a combination of several aspects, and, in most cases, result in new complex propagators.

A business process (BP) can be defined as a set of related structured activities whose execution produces a specific service or product required by a particular customer. ConDec (Pesic and van der Aalst 2006) is a declarative language which is used to specify dynamic BP models through a graphical notation which can be mapped to formulas in Linear Temporal Logic (LTL).

In the current work, the problem definition is specified in a new language (ConDec-R), based on ConDec, which allows to specify constraints which restrict the selection and the ordering of activities which can be executed several times, together with the requirements of resources. For the definition of relationships between the repeated activities, ConDec-R proposes an open set of high-level templates which must be satisfied. Once the problem is specified in ConDec-R, is then translated into a CSP so that a constraint-based solver can be used to obtain an optimal execution plan for the specified problem. The resolution of the CSP problems entails the selection and the ordering of the activities to be executed (planning), and the resource allocation involving temporal reasoning (scheduling), both of which consider the optimization of some objective functions.

Currently, there exists an increasing interest in the effective management of BP (BP Management, BPM). Related to this, BPM Systems (BPMS) are responsible for goal specification, design, implementation, enactment, monitoring, and evaluation of BP (Muehlen and Ting-Yi Ho 2005). In general, for the execution of activities, the use of shared resources is necessary, which must be managed in an effective way to optimize some aspects of the BP enactment. As a practical application of the current approach, ConDec-R can be used to specify the relations given between the activities which are involved in BP environments, in a declarative way. In fact, the high-level constraints considered in this work are taken and extended from a declarative language for BP.

The proposed new filtering rules, together with the related high-level global constraints, can be used to efficiently solve

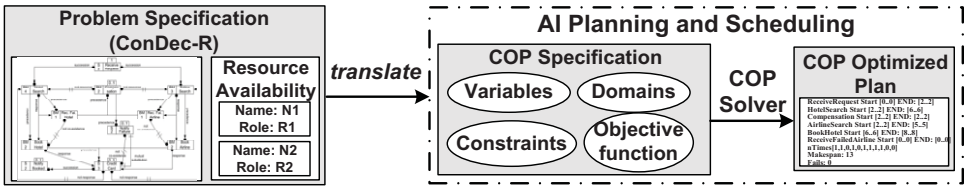

Figure 1: AI P&S techniques for generating optimized plans for the execution of repeated activities.

further P&S problems unrelated to business process environments, which include similar relations between repeated activities.

In Fig. 1, a graphical representation of the current approach is shown. First, the input problem information is specified through ConDec-R language, which includes constraints which restrict the selection and the ordering of activities, and the requirements of resources, for activities which can be executed several times. This information is translated into a constraint-based specification, which in turn is used as input for a solver in order to obtain a feasible optimized execution plan. As stated, the constraint-based approach includes propagators for treating the high-level relationships between the activities.

In short, the main contributions of this work are:

- The development of a constraint-based approach in order to solve problems which deal with the P&S of activities which can be executed several times and are related by high-level constraints. This approach includes new filtering rules for the definition of the high-level relations between the repeated activities.
- An application of the proposed approach in a BP environment in order to automatically generate optimized BP enactment plans from declarative BP specifications.

This paper is organized as follows: Section 2 explains the ConDec-R language, Section 3 details the proposed constraint-based approach, Section 4 presents an application of the proposed approach in BP environments, Section 5 shows some experimental results, Section 6 summarizes the most related work, and, finally, Section 7 presents some conclusions and future work.

2 ConDec-R Language

ConDec (Pesic and van der Aalst 2006) is a declarative language which proposes an open set of constraints based on Linear Temporal Logic (LTL) for the definition of high-level relations between activities which can be executed several times. ConDec was proposed in order to specify dynamic BP models using a graphical notation.

Since ConDec does not allow direct reasoning about resources, a new language based on ConDec, named ConDec-R is defined in this work. The main contribution of ConDec-R with respect to ConDec is the reasoning about resources, since in ConDec-R the execution of activities requires resources of a specific role.

In short, a ConDec-R problem specification must include:

- As in ConDec, the activities which can be executed.
- As in ConDec, the relations between the activities through high-level templates.
- As extensions of ConDec, for each activity, the role of the required resource for its execution.
- As extensions of ConDec, the estimated duration of each activity.
- As extensions of ConDec, the available resources with the competences defined by a role.

One of the most important aspects when modelling with ConDec-R is that a ConDec-R activity represents multiple executions of a P&S activity. Several executions of the same ConDec-R activity can be modelled as a sequence of single P&S activities. In this work, the high-level relations between the ConDec-R activities are translated into constraints between the corresponding P&S activities.

3 Constraint-based Proposal

3.1 From ConDec-R to CSP Model

In the current proposal, repeated activities are modelled as a sequence of optional scheduling activities for the definition of the propagators. Each ConDec-R activity can be executed several times. Each time one of these activities is executed, it can be seen as a P&S-activity execution, that *precedes* the next execution of the same activity. Bearing this in mind, each ConDec-R activity is modelled as a linear sequence which is composed of several P&S activities.

In ConDec-R, the number of times each activity is to be executed can be unknown, and hence each P&S activity can potentially be included. A type representing the ConDec-R activities, named RepeatedActivity, is presented in Fig. 2 (UML diagram). The attributes of RepeatedActivity are: *duration*, which represents the estimated duration of the ConDec-R activity; *role*, which indicates the role of the required resource for the activity execution; and *nt*, which shows the actual number of times which the ConDec-R activity is executed. The RepeatedActivity type is composed of *nt* SchedulingActivities. Each SchedulingActivity includes the *st* and the *et* attributes, which indicate the start and the end times of the activity execution, respectively. The fact that each P&S activity can potentially be included, is modelled through the *sel* variable of the SchedulingActivity type, equal to 0 in the case that it is not executed, and equal to 1 otherwise.

Henceforth, $nt(A)$ refers to the CSP variable that represents the number of times that the ConDec-R activity A is executed; $act(A,i)$ represents the i-th execution of A; and $st(act(A,i))$ and $et(act(A,i))$ represent the start and the end times of the i-th execution of A, respectively. Furthermore, the constraints $\forall i : 1 \leq i \leq nt(A) : sel(A,i) = 1$ and $\forall i > nt(A) : sel(A,i) = 0$ hold for each activity A.

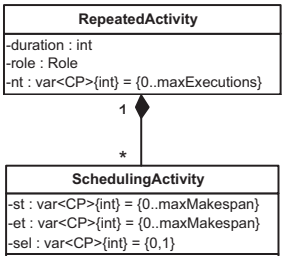

Figure 2: RepeatedActivity and SchedulingActivity types

It should be clarified that the number of times which each activity is executed is commonly unknown in advance (it is a CSP variable of the model which must be instantiated). Hence, it is necessary to select and to order the suitable activities in order to reach the established goal. This fact provides the considered problem with some characteristics of a planning problem.

Some representative examples of ConDec-R templates are graphically represented in Fig. 3, where three precedence relations between two repeated activities, A and B, are shown. As stated earlier, several executions of the same repeated activity can be modelled as a sequence of single P&S activities. In this figure, the P&S activity A_i represents the i-th execution of the repeated activity A ($act(A,i)$). In this figure the arrow represents:

- A precedence relation between two P&S activities A_i and B_j, when it appears between two activities ($et(A_i) \leq st(B_j)$).
- A precedence relation between a P&S activity A_i and a set S of P&S activities, when it appears between an activity and a dotted rectangle which encloses a set of activities ($\exists B_j \in S : et(A_i) \leq st(B_j)$).
- A precedence relation between a set S of P&S activities and a P&S activity B_j, when it appears between a dotted rectangle which encloses a set of activities, and an activity ($\exists A_i \in S : et(A_i) \leq st(B_j)$).

In a similar way, a special arrow (wider than the other arrows and with a big dot in its origin) which appears between two P&S activities, A and B, shows that A must be executed **immediately** before B ($et(A) = st(B)$). In a similar way, this can be defined for a set of activities. More details about Fig. 3 are shown in the definition of the related templates.

It should be clarified that the constraints $\forall i : 1 \leq i < nt(A) : et(act(A,i)) \leq st(act(A,i+1))$ and $\forall i : i > nt(A) : st(act(A,i)) = et(act(A,i))$ hold for each repeated activity.

The ConDec-R templates are listed as follows.

- $Existence(A,N)$: A must be executed more than or equal to N times, $nt(A) \geq N$.
- $Absence(A,N)$: A must be executed fewer than N times, $nt(A) < N$.
- $Exactly(A,N)$: A must be executed exactly N times, $nt(A) = N$.
- $RespondedExistence(A,B)$: If A is executed, then B must also be executed either before or after A, $nt(A) > 0 \Rightarrow nt(B) > 0$.

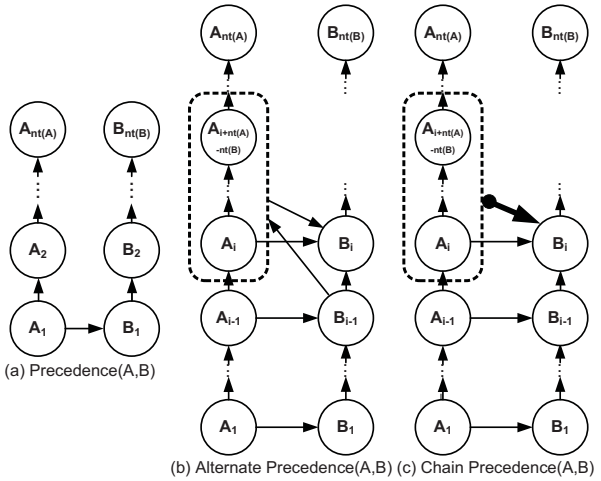

Figure 3: Precedence templates when $nt(B) > 0$.

- $CoExistence(A,B)$: The execution of A forces the execution of B, and vice versa, $nt(A) > 0 \iff nt(B) > 0$.
- $Precedence(A,B)$: Before the execution of B, A must have been executed, $nt(B) > 0 \Rightarrow (nt(A) > 0) \wedge (et(act(A,1)) \leq st(act(B,1)))$. As can be seen in Fig. 3(a), this relation implies that A_1 must precede B_1 in the case that $nt(B) > 0$.
- $Response(A,B)$: After the execution of A, B must be executed, $nt(A) > 0 \Rightarrow (nt(B) > 0) \wedge (st(act(B,nt(B))) \geq et(act(A,nt(A))))$.
- $Succession(A,B)$: Relations $Precedence(A,B)$ and $Response(A,B)$ hold.
- $AlternatePrecedence(A,B)$: Before the execution of B, A must have been executed, and between each two executions of B, A must be executed. It implies that:
  a. The number of times that A is executed must be greater than or equal to the number of times that B is executed: $nt(A) \geq nt(B)$.
  b. Between each two executions of B, A must be executed at least once. Specifically, between the $(i-1)$-th and the i-th execution of B, the earliest execution of A that can exist is i, and hence A_{i-1} must precede B_{i-1} (as can be seen in Fig. 3(b)). In a similar way, between the $(i-1)$-th and the i-th execution of B, the latest execution of A that can exist is $i + nt(A) - nt(B)$, and hence B_i must precede $A_{i+nt(A)-nt(B)+1}$. This can also be seen in Fig. 3(b), where the possible activities to be executed between the $(i-1)$-th and the i-th execution of B are framed within the dotted rectangle. $\forall i : 2 \leq i \leq nt(B) : \exists j : i \leq j \leq i + nt(A) - nt(B) :$
  $st(act(A,j)) \geq et(act(B,i-1)) \wedge$
  $et(act(A,j)) \leq st(act(B,i))$.
  c. Before B, A must be executed: $st(act(B,1)) \geq et(act(A,1))$.
- $AlternateResponse(A,B)$: After the execution of A, B must be executed, and between each two executions of A, there must be at least one execution of B. It implies:

a. The number of times that B is executed must be greater than or equal to the number of times that A is executed: $nt(B) \geq nt(A)$.

b. Between each two executions of A, B must be executed at least once. Specifically, between the i-th and the $(i+1)$-th execution of A, the earliest execution of B that can exist is i, and hence B_{i-1} must precede A_i. In a similar way, between the i-th and the $(i+1)$-th execution of B, the latest execution of A that can exist is $i+nt(B)-nt(A)-1$, and hence A_i must precede $B_{i+nt(B)-nt(A)}$. $\forall i: 1 \leq i < nt(A) : \exists j : i \leq j \leq i+nt(B)-nt(A)-1 : st(act(B,j)) \geq et(act(A,i)) \wedge et(act(B,j)) \leq st(act(A,i+1))$.

c. After A, B must be executed: $st(act(B,nt(B))) \geq et(act(A,nt(A)))$.

- $AlternateSuccession(A,B)$: Relations $AlternatePrecedence(A,B)$ and $AlternateResponse(A,B)$ hold.
- $ChainPrecedence(A,B)$: **Immediately** before B, A must be executed. It implies that:

  a. The number of times that A is executed must be greater than or equal to the number of times that B is executed: $nt(A) \geq nt(B)$.

  b. Immediately before each execution of B, A must be executed. Specifically, before the i-th execution of B, the earliest execution of A that can exist is i. In a similar way, before the i-th execution of B, the latest execution of A that can exist is $i+nt(A)-nt(B)$. $\forall i: 1 \leq i \leq nt(B) : \exists j : i \leq j \leq i+nt(A)-nt(B) : et(act(A,j)) = st(act(B,i))$.

  This is shown in Fig. 3(c), where a special arrow (wider than the other arrows and with a big dot in its origin) shows that A must be executed **immediately** before B.

- $ChainResponse(A,B)$: **Immediately** after A, B must be executed. It implies:

  a. The number of times that B is executed must be greater than or equal to the number of times that A is executed: $nt(B) \geq nt(A)$.

  b. Immediately after each execution of A, B must be executed. Specifically, before the i-th execution of A, the earliest execution of B that can exist is i. In a similar way, after the i-th execution of A, the latest execution of B that can exist is $i+nt(B)-nt(A)-1$. $\forall i: 1 \leq i \leq nt(A) : \exists j : i \leq j \leq i+nt(B)-nt(A)-1 : st(act(B,j)) = et(act(A,i))$.

- $ChainSuccession(A,B)$: Relations $ChainPrecedence(A,B)$ and $ChainResponse(A,B)$ hold.
- $RespondedAbsence$ and $NotCoExistence(A,B)$: If B is executed, then A cannot be executed, and vice versa, $((nt(A)>0)\cdot(nt(B)>0)) == 0$.
- $NegationResponse$, $NegationPrecedence$, $NegationSuccession(A,B)$: After the execution of A, B cannot be executed, $(nt(A)>0 \wedge nt(B)>0) \Rightarrow st(act(B,nt(B))) \leq et(act(A,1))$.
- $NegationAlternatePrecedence(A,B)$: Between two executions of B, A cannot be executed, $nt(B) \geq 2 \Rightarrow \forall i: 1 \leq i \leq nt(A) : et(act(A,i)) \leq st(act(B,1)) \vee st(act(A,i)) \geq et(act(B,nt(B)))$.
- $NegationAlternateResponse(A,B)$: Between two executions of A, B cannot be executed, $nt(A) \geq 2 \Rightarrow \forall 1 \leq i \leq nt(B) : et(act(B,i)) \leq st(act(A,1)) \vee st(act(B,i)) \geq et(act(A,nt(A)))$.
- $NegationAlternateSuccession(A,B)$: Relations $NegationAlternatePrecedence(A,B)$ and $NegationAlternateResponse(A,B)$ hold.
- $NegationChainSuccession(A,B)$: B cannot be executed immediately after the execution of A, $\forall i: 1 \leq i \leq nt(B) : \neg\exists j : 1 \leq j \leq nt(A) : et(act(A,j)) = st(act(B,i))$.

```
Existence(A,N) is added ->
    If N > LB(nt(A)) then
        LB(nt(A)) <- N

Precedence(A,B) is added OR bounds of nt(B) changed OR bounds of
st(act(B,1)]) changed OR bounds of et(act(A,1)) changed ->
    If LB(nt(B)) > 0 then
        nt(A) <- nt(A) - {0}
    If LB(et(act(A,1))) > LB(st(act(B,1))))then
        LB(st(act(B,1)))) <- LB(et(act(A,1)))
    If UB(et(act(A,1))) > UB(st(act(B,1))) then
        UB(et(act(A,1))) <- UB(st(act(B,1)))
```

Figure 4: Propagators for some ConDec-R Templates

3.2 Filtering Rules

In a constraint-based environment, the filtering rules (propagators) are responsible for removing values which do not belong to any solution. In this work, a propagator for each ConDec-R template has been developed from the expressions stated in Section 3.1 in order to increase the efficiency in the search of solutions.

As an example, three representative propagators corresponding to templates of varying levels of difficulties are shown in Figs. 4 and 5, where the propagator that describes the pruning of domains appears after the symbol $\rightarrow$. $UB(v)$ and $LB(v)$ represent the upper and lower bounds of the domain of v, respectively. The proposed reasoning for the optional activities is similar to the proposal presented in (Laborie et al. 2009), and hence a scheduling activity can be modelled as a time-interval variable.

The *Existence* rule (Fig. 4) is invoked when the template is added to the constraint model, hence its trigger "Existence(A,N) is added". Moreover, the *Precedence* (Fig. 4) and *Alternate Precedence* rules (Fig. 5) are invoked when the templates are added or when the domain bounds of some variables are updated. The fact that the relations $\forall i: 1 \leq i \leq LB(nt(A)) : sel(A,i) = 1$ and $\forall i > UB(nt(A)) : sel(A,i) = 0$ hold is relevant in the development of the propagators.

Proposition 1: If implemented properly, the time complexity of the rule *Existence*, which includes all possible recursive calls, is $\Theta(1)$.

Proof: The *Existence* rule is fired only when the constraint is added, and the time complexity of its execution is constant, hence the complexity of this rule is $\Theta(1)$.

```
Alternate Precedence (A,B) is added OR
(bounds of nt(A) changed) OR (bounds of nt(B) changed) OR (bounds
of st(act(A,i)) for any i changed) OR (bounds of et(act(A,i)) for
any i changed) OR (bounds of st(act(B,i)) for any i changed) OR
(bounds of et(act(B,i)) for any i changed) ->
   if (LB(nt(A)) < LB(nt(B))) {LB(nt(A)) <- LB(nt(B))}
   if (UB(nt(B)) > UB(nt(A))) {UB(nt(B)) <- UB(nt(A))}
   for (int i = 1; i <= UB(nt(B)); i++){
      SchedulingActivity a = act(A,i);
      SchedulingActivity b = act(B,i);
      if (LB(et(a)) > LB(st(b))) {LB(st(b)) <- LB(et(a))}// Ai -> Bi
      if (UB(st(b)) < UB(et(a))) {UB(et(a)) <- UB(st(b))}// Ai -> Bi
   }
   for (int i = 1; i < LB(nt(B)); i++){
      int dif = UB(nt(A)) - LB(nt(B));
      SchedulingActivity a = act(A,i+dif+1);
      SchedulingActivity b = act(B,i);
      // Bi -> Ai+dif+1
      if (LB(et(b)) > LB(st(a))) {LB(st(a)) <- LB(et(b))}
      if (UB(st(a)) < UB(et(b))) {UB(et(b)) <- UB(st(a))}
   }
   // force exists A between Bi-1 and Bi
   for (int i = 2; i <= UB(nt(B)); i++){
      int dif = UB(nt(A)) - max(i,LB(nt(B));
      SchedulingActivity b1 = act(B,i-1);
      SchedulingActivity b2 = act(B,i);
      // Candidate As between Bi-1 and Bi
      int j = i;
      bool forcedValue = false;
      while (j <= (i + dif) && !forcedValue) {
         SchedulingActivity aFor = act(A,j);
         int k = i;
         bool force = true;
         while (k <= (i + dif) && force){
            if (k != j){
               SchedulingActivity a = act(A,k);
               //If Bi-1->Ak->Bi possible, not force Bi-1->Aj->Bi
               if (UB(st(a))>=LB(et(b1))&&LB(et(a))<=UB(st(b2)))
                  force = false;
            }
            k++;
        }
        if (force){ // force Bi-1 -> Aj -> Bi
             forcedValue = true;
             // Bi-1 -> Aj
             if (LB(et(b1))>LB(st(aFor))){LB(st(aFor)) <- LB(et(b1))}
             if (UB(st(aFor))<UB(et(b1)) {UB(et(b1)) <- UB(st(aFor))}
             // Aj -> Bi
             if (LB(et(aFor))>LB(st(b2))){LB(st(b2) <- LB(et(aFor))}
             if (UB(st(b2))<UB(et(aFor))){UB(et(aFor)) <- UB(st(b2))}
        } // end if
        j++;
     }}// end while j, end for i
```

Figure 5: Propagator for the Alternate Precedence Template

Proposition 2: If implemented properly, the worst-case time complexity of the rule *Precedence*, which includes all possible recursive calls, is $O(n)$, where n is the number of Repeated (ConDec-R) Activities of the problem.

Proof: The *Precedence* rule can be fired, at most, n times. This is due to the fact that only a change in the first execution of an activity (Act_1) can fire this rule. Moreover, the time complexity of the *Precedence* rule execution is constant, and hence the worst-case complexity of this rule is $O(n)$.

Proposition 3: If implemented properly, the worst-case time complexity of the rule *AlternatePrecedence*, which includes all possible recursive calls, is $O(n \times nt^4)$, where n is the number of Repeated Activities of the problem, and nt is the upper bound of the variable $nt(Act)$ domain. This upper bound obtains the same value for all the ConDec-R activities.

Proof: The *AlternatePrecedence* rule can be fired, at most, $n \times nt$ times. This is due to the fact that a change in any execution of any activity can fire this rule. Moreover, the time complexity of the *AlternatePrecedence* rule execution is $O(nt^3)$, and hence the worst-case time complexity of this rule is $O(n \times nt^4)$.

It should be emphasized that the developed propagators deal with a combination of several aspects, and, in most cases, result in new complex propagators. For example, the Alternate Precedence propagator (Fig. 5) deals with repeated activities which must be executed in an alternating way (this aspect has not been previously treated by a constraint propagator, to the best of our knowledge), while the number of times each activity is executed is a constraint variable, resulting in a complex and new propagator. The development of these filtering rules allows problems to be easily specified through these global constraints, and also to be solved in an efficient way due to the strong constraint propagation.

There can be several objectives to be pursued in P&S problems. We have considered minimizing the makespan, but the CSP model presented can be extended to consider further objectives.

4 From Declarative Specifications to Optimized BP Enactment Plans

In recent years, interest has grown in the integration of P&S techniques with BPMS (R-Moreno et al. 2007; González-Ferrer, Fernández-Olivares, and Castillo 2009; Hoffmann, Weber, and Kraft 2010). However, from our point of view, several connections between these two disciplines remain to be exploited.

In business process (BP) environments, in most cases, the model phase is manually carried out by business analysts, who must deal with several aspects, such as resource allocation, activity properties and the relations between them, and even the optimization of several objectives. The manual specification of BP models can form a very complex problem, since it can consume a great quantity of time and human resources, cause some failures, or lead to non-optimized models. Furthermore, in most cases, BP information is provided through imperative languages.

An application of the presented constraint-based approach entails the generation of optimized execution plans for BP which are specified in a declarative way through ConDec-R language. The specification of process properties in a declarative way is an important step towards the (automatic) generation of BP models. ConDec (Pesic and van der Aalst

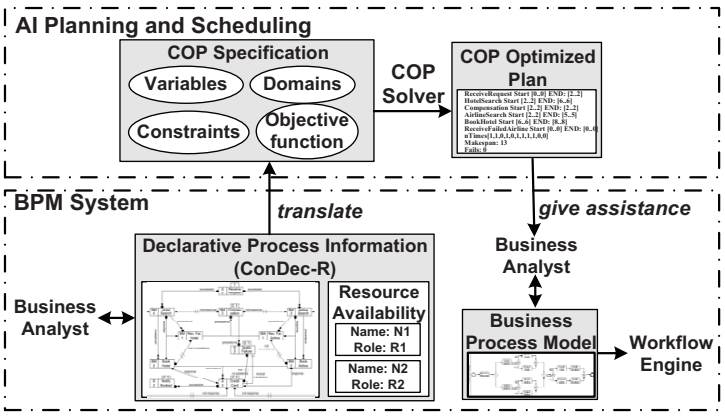

Figure 6: AI P&S techniques for the generation of BP models.

2006) is a declarative language to specify dynamic BP models using a graphical notation.

As stated, in this work, the BP specification is provided in a declarative way through ConDec-R. We consider ConDec-R as a suitable declarative language for BP, since it allows the specification of business activities together with the constraints which must be satisfied, including the objective to be pursued and resource requirements.

Fig. 6 shows the application of the presented approach in BP environments. First, the business analyst provides the declarative BP information through a ConDec-R specification, which includes constraints which restrict the selection and the ordering of activities which can be executed several times, together with the resource requirements. This information is translated into the proposed constraint-based specification, which in turn is used as input for a solver to obtain a feasible optimized BP execution plan. As stated, the constraint-based specification includes suitable propagators for treating the high-level relationships between the activities. Moreover, this optimized enactment plan provides assistance for the generation of an optimized BP model.

In a typical BPM life cycle, P&S techniques can be applied at different stages in a coordinated way to improve overall system functionality. Our work is based on automating the model generation in the BP design and system configuration phases, by considering estimated values. It should be clarified that in a coordinated way, in the enactment phase, P&S techniques can, where necessary, be applied to replan the activities by considering the actual values of the parameters.

5 Empirical Evaluation

Our proposal is in a preliminary step from a commercial point of view, and hence a limited evaluation is presented. It focuses on problems with different kinds of precedence relations between activities that can be executed several times.

5.1 Experimental Design

In the empirical evaluation, due to the fact that the analyzed relations are introduced in this paper for the first time, there has not been possible to find a previously developed solver which deals with these relations. Also, due to the use of a specific and new language, it has been impossible to find a set of public benchmarks for ConDec-R problems. As far as the purpose of the experimental evaluation is to determine the efficacy of using our own propagators to improve the efficiency, it has been considered suitable to compare the results obtained by: first, the proposal with propagators and the related high-level global constraints; and secondly, with neither propagators nor global constraints (in this case, the equivalent local constraints are included).

In order to solve the COPs, COMET (Dynadec 2010) is used, which is able to swiftly generate high-quality solutions for highly constrained problems. It provides a Scheduling module that offers high-level modelling and search abstraction specific to scheduling. In this work, an adaptation from the P&S problem under consideration into a scheduling problem is proposed in order to take advantage of the efficient COMET mechanisms and high-level modelling.

ConDec-R problems can be modelled as an extension of scheduling problems, and hence a set of well-known scheduling benchmarks (FT06 (6x6), FT10 (10x10), ABZ06 (10x10), LA21 (15x10), LA36 (15x15)) have been extended for use in the empirical evaluation in the following way:

- In a scheduling problem, the activities are executed only once. For this evaluation, the cardinality of some activities (a random percentage, $p\%$) has been set at $c > 1$, column $Card = c(p\%)$ in Table 1. In order to minimize the influence of the random component, 50 random instances are generated for each problem.
- In a scheduling problem, all the relations between activities are Precedence. For the current empirical evaluation, the alternate and chain precedence relations are also considered: in the original problem P, all the precedence relations between the activities of the same job are changed to alternate precedence relations (*PAlt*. in Table 1) and chain precedence relations (*PChain* in Table 1).

Setting the cardinality of $p\%$ activities to a value $v > 1$, can result in a solution where there are $p'\%$ activities ($p' > p$) with cardinality v. For example, each activity a with $nt(a) = v$ implies that for all b where $AlternatePrecedence(b,a)$ or $ChainPrecedence(b,a)$ hold, then $nt(b) \geq nt(a)$ must be satisfied (the same logic applies for all the activities c such that $AlternatePrecedence(c,b)$ or

Table 1: Results on a set of ConDec-R problems from JSS instances

	Card = 2 (5%)		Card = 2 (10%)		Card = 3 (5%)		Card = 3 (10%)	
Problem	B_P^M (B_P^T)	B^M (B^T)	B_P^M (B_P^T)	B^M (B^T)	B_P^M (B_P^T)	B^M (B^T)	B_P^M (B_P^T)	B^M (B^T)
FT06 Prec.	0	0 (1)	0	0 (2)	0	0	0	0 (2)
FT06 Alt.	6 (1)	20 (22)	7 (2)	36 (4)	37	0 (3)	50	0
FT06 Chain	7 (1)	23 (19)	6 (2)	37 (4)	37	0 (3)	50	0
FT10 Prec.	0	0	0	0	0	0	0	0
FT10 Alt.	11	36	5	44	50	0	50	0
FT10 Chain	13	35	3	46	50	0	50	0
ABZ06 Prec.	0	0	0	0	0	0	0	0
ABZ06 Alt.	13	30	6	42	50	0	50	0
ABZ06 Chain	10	36	3	44	50	0	50	0
LA21 Prec.	0	0	0	0	0	0	0	0
LA21 Alt.	11	39	2	48	50	0	50	0
LA21 Chain	3	47	2	48	50	0	50	0
LA36 Prec.	0	0	0	0	0	0	0	0
LA36 Alt.	3	47	1	49	50	0	50	0
LA36 Chain	0	50	0	50	50	0	50	0

ChainPrecedence(c,b), etc).

The behaviour of five representative propagators are tested: Exactly, Existence, Precedence, Alternate and Chain Precedence. The results obtained with two proposals, which use the same variables, are compared to study the efficiency:

1. With propagators: the user-defined constraints are used for the establishment of the high-level relations between the ConDec-R activities. In this case, the proposed filtering algorithms are responsible for removing inconsistent values from the domain of the variables.
2. Without propagators: the relations between the activities are established directly through COMET constraints, as shown in Section 3.1.

Some performance measures are reported (Table 1):

- B_P^M: Number of instances for which the makespan found by the proposal with propagators is shorter than the makespan found without propagators.
- B^M: Number of instances for which the makespan found by the proposal without propagators is shorter than the makespan found with propagators.
- B_P^T: Number of instances for which the solutions found by both proposals obtain the same makespan value, but the proposal with propagators is faster.
- B^T: Number of instances for which the solutions found by both proposals obtain the same makespan value, but the proposal without propagators is faster.

For both proposals, a complete search approach has been applied: first, the variables related to the number of times that each activity is executed are instantiated. The search procedure then determines the order of execution of the activities within each resource at each step, such that the next resource to be ranked is selected depending on its slack. Within each resource, the activities are ranked in a non-deterministic way.

5.2 Experimental Results

For the experiments, each algorithm is run until it finds the optimal solution or until a 5-minute CPU time limit has been reached. The machine for all experiments is an Intel Core2, 2.13 GHz, 1.97 GB memory, running Windows XP.

The scheduling benchmarks with the lowest complexity are those that only include precedence relations and have the lowest cardinality for the activities. In Table 1, some experimental results are shown. It is possible to see that:

- For problems with only precedence relations, the solutions for the two proposals are very similar. In this case, the problem is simply a Job Shop problem, and COMET includes efficient mechanisms for solving scheduling problems.
- For problems with alternate and chain precedence relations, the proposal without propagators obtains better solutions for problems with lower complexity (or cardinality). When the percentage of activities with greater cardinality increases, the proposal with propagators proves better than the other proposal in almost all instances.

In short, the proposal without propagators obtains better solutions for problems of lower complexity (only precedence relations, low cardinality), while the proposal with propagators is clearly much better for more complex problems, and hence can be considered better for general cases.

6 Related Work

In recent years, several filtering algorithms for specialized scheduling constraints have been developed. Specifically, (Beck and Fox 2000) and (Barták and Cepek 2010) model scheduling problems which include alternative and optional tasks respectively, together with their propagators. Furthermore, the work (Barták and Cepek 2008) proposes propagators for both precedence and dependency constraints in order to solve log-based reconciliation (P&S) problems in

databases. In those studies, the precedence constraints signify the same as in P&S problems, while the dependency constraints are given between optional activities which can potentially be included in the final schedule. The work (Laborie et al. 2009) introduces new types of variables (time-interval, sequence, cumulative, and state-function variables) for modelling and reasoning with optional scheduling activities. In our work, the proposed model and propagation for the optional activities are very similar to the proposal presented in (Laborie et al. 2009).

Regarding the works (Beck and Fox 2000), (Barták and Cepek 2008), (Barták and Cepek 2010) and (Laborie et al. 2009), the main contribution from the developed propagators is the complex reasoning about several combined innovative aspects, such as the alternating executions of repeated activities together with the variable number of times which these activities are executed (i.e. alternate and chain relations). Furthermore, the areas of application of those studies are unrelated to those of our proposal.

In BP environments, most of the related works integrate P&S with BPMS in the enactment phase (e.g. (Barba and Del Valle 2010)), while very few integrations are carried out during the modelling phase, as presented here. In (R-Moreno et al. 2007), planning tools are used in the generation of BP models, by considering the knowledge introduced through BP Reengineering languages. This knowledge is translated into predicate logic terms in order to be treated by a planner. In (González-Ferrer, Fernández-Olivares, and Castillo 2009), the BP information is provided through an execution/interchange language, XPDL. The XPDL file is analysed to obtain a workflow pattern decomposition, which is translated to the HTN-PDDL language. This is used as the input of the planner, and the resulting plans could be interpreted as workflow instances. Moreover, the work (Hoffmann, Weber, and Kraft 2010) proposes a planning formalism for modelling BP through an SAP specification (SAM), which is a variant of PDDL. To the best of our knowledge, the automatic generation of BP models from declarative specifications is not treated in any other work.

7 Conclusions and Future Work

This work proposes a constraint-based approach for the generation of optimal execution plans for activities which can be executed several times and are related by high-level templates. The proposed approach includes new propagators for the definition of the high-level relations between the repeated activities, and thereby facilitates the specification of the problem through global constraints at the same time as it enables the efficiency in the search for solutions to increase. The developed propagators deal with the combination of several aspects, and, in most cases, result in new complex propagators. Some experimental results are analysed, which confirm the advantages of using the developed propagators. The main application of this work is to provide business analysts with assistance in the generation of optimized BP models from declarative specifications. This interesting and innovative topic has yet to be addressed. The proposed propagators can be used to efficiently solve further P&S problems unrelated to BP environments, which include similar relations between repeated activities.

As for future work, various constraint-based hybrid search algorithms will be studied for the problem under consideration. It is also intended to extend this proposal by considering further objective functions.

8 Acknowledgments

This work has been partially funded by the Consejería de Innovación, Ciencia y Empresa of Junta de Andalucía (P08-TIC-04095) and by the Spanish Ministerio de Ciencia e Innovación (TIN2009-13714) and the European Regional Development Fund (ERDF/FEDER).

References

Barba, I., and Del Valle, C. 2010. Planning and Scheduling of Business Processes in Run-Time: A Repair Planning Example. In *Proceedings of the 19th International Conference on Information Systems Development (ISD 2010)*. Springer (in press).

Barták, R., and Cepek, O. 2008. Incremental filtering algorithms for precedence and dependency constraints. *Int. Journal on Artificial Intelligence Tools* 17(1):205–221.

Barták, R., and Cepek, O. 2010. Incremental propagation rules for a precedence graph with optional activities and time windows. *Transactions of the Institute of Measurement and Control* 32(1):73–96.

Beck, J., and Fox, M. 2000. Constraint-directed techniques for scheduling alternative activities. *Int. Journal on Artificial Intelligence* 121:211–250.

Dynadec. 2010. Comet Downloads. `http://dynadec.com/support/downloads/` . [accessed 16-March-2011].

González-Ferrer, A.; Fernández-Olivares, J.; and Castillo, L. 2009. JABBAH: A Java Application Framework for the Translation Between Business Process Models and HTN. In *ICKEPS*.

Hoffmann, J.; Weber, I.; and Kraft, F. M. 2010. SAP speaks PDDL. In *AAAI'10: 24th AAAI Conference on Artificial Intelligence*, 1096–1101.

Laborie, P.; Rogerie, J.; Shaw, P.; and Vilím, P. 2009. Reasoning with Conditional Time-Intervals. Part II: An Algebraical Model for Resources. In *FLAIRS Conference*.

Muehlen, M., and Ting-Yi Ho, D. 2005. Risk Management in the BPM Lifecycle. In *BPM Workshops*, 454–466.

Pesic, M., and van der Aalst, W. M. P. 2006. A declarative approach for flexible business processes management. In *BPM Workshops*, 169–180. Springer.

R-Moreno, M.; Borrajo, D.; Cesta, A.; and Oddi, A. 2007. Integrating planning and scheduling in workflow domains. *Expert Syst. Appl.* 33(2):389–406.

Rossi, F.; Van Beek, P.; and Walsh, T. 2006. *Handbook of Constraint Programming*. Elsevier.

Salido, M. 2010. Introduction to planning, scheduling and constraint satisfaction. *Journal of Intelligent Manufacturing* 21(1):1–4.

A CFLP Approach for Modeling and Solving a Real Life Employee Timetabling Problem *

Ignacio Castiñeiras and **Fernando Sáenz-Pérez**
Complutense University of Madrid, Spain
ncasti@fdi.ucm.es and fernan@sip.ucm.es

Abstract

In last years the number of applications of timetabling has grown spectacularly, and different paradigms have risen to tackle these problems. In this paper we present a Constraint Functional Logic Programming (CFLP) approach for modeling and solving a real life optimization employee timetabling problem. We describe the language supported by a particular implementation of the CFLP paradigm. Then, we present the concrete model followed to solve the problem, and we enumerate the advantage our framework provides w.r.t. other approaches. Running results are also reported.

1. Introduction

The Nurse Rostering Problem (NRP) has been extensively studied for more than forty years (Burke et al. 2004). Due to its relevance as a real life problem, it represents the most paradigmatic example of the family of employee timetabling problems. As it is a complex problem, a wide set of techniques has been applied to tackle it, both on formalization design and solving techniques terms. Here we mention Integer Programming (Burke, Li, and Qu 2010), Evolutionary Algorithms (Moz and Pato 2007) and Tabu Search (Burke, Kendall, and Soubeiga 2003) as some approaches. However, due to the constraint oriented nature of the problem, the Constraint Satisfaction Problem (CSP) paradigm (Tsang 1993) becomes a quite suitable framework to the formulation and solving of NRPs. Several programming paradigms have risen to tackle CSPs. First is Constraint Programming (CP) (Marriot and Stuckey 1998), which provides expressive languages for describing the constraints and powerful solver mechanism for reasoning with them. While a CP formulation becomes algebraic (a very abstract programming paradigm) it lacks the benefits of a general constraint programming language. As CP search space can become huge, different techniques as decomposition (Meisels and Kaplansky 2002) and relaxation (Métivier, Boizumault, and Loudni 2009) has been applied in the solving of NRPs. Another programming paradigm is Constraint Logic Programming (CLP) (Jaffar and Maher 1994), which combines Logic Programming (LP) and CP, providing general purpose languages also equipped with constraint solving. As constraints are basically true relations among domain variables, its integration in the logic field became in a quite natural way.

In this paper we focus on Constraint Functional Logic Programming (CFLP) as another approach to solve employee timetabling problems. CFLP adds constraint solving to the Functional Logic Programming (FLP) framework (Hanus 1994), and attempts to be an adequate framework for the integration of the main properties of Functional Programming (FP) and CLP. Adequation of CFLP($\mathcal{FD}$) to meet timetabling problems was proposed in (Brauner et al. 2005). The CFLP language presented in (Fernández et al. 2007) combines functional and relational notation, curried expressions, higher-order functions, patterns, partial applications, non-determinism, backtracking, lazy evaluation, logical variables, types, polymorphism, domain variables and constraint composition as some of its features. While its declarative semantics is based on a Conditional Term-Rewriting Logic (CRWL) (González-Moreno et al. 1999), its operational semantics is based on a constrained demanded narrowing calculus (López-Fraguas, Rodríguez-Artalejo, and Vado-Vírseda 2004), making possible to solve syntactic equality and disequality constraints over the Herbrand Universe ($\mathcal{H}$ domain constraints), as well as $\mathcal{FD}$ constraints, relying on the use of an external solver. An implementation of the system, named $\mathcal{TOY}(\mathcal{FD})$, was also presented. Another approach to this framework is $Curry$ (Hanus 1999), and its particular implementation $PAKCS$ (PAKCS), also supporting $\mathcal{FD}$ constraint solving by relying on an external solver. As it is quite similar to the $\mathcal{TOY}(\mathcal{FD})$ approach, a benchmark comparison between both systems was done in (Fernández et al. 2007).

This paper presents an application of $\mathcal{TOY}(\mathcal{FD})$ to the modeling and solving of a real life optimization employee timetabling problem, whose formulation can be seen as a particular instance of the NRP. The structure of the paper is the following: while sections 2 and 3 concern both with language and problem description, Section 4 presents our approach to the modeling of the problem. Section 5 points out the advantages a CFLP($\mathcal{FD}$) approach offers to face up to this problem, in contrast to using CP($\mathcal{FD}$) or CLP($\mathcal{FD}$). Section 6 presents running results for several instances of the problem. Finally, Section 7 reports some conclusions.

*This work has been partially supported by the Spanish projects TIN2008-06622-C03-01, UCM-BSCH-GR58/08-910502, and S2009TIC-1465

2. The $\mathcal{TOY}(\mathcal{FD})$ Language

We use constructor-based signatures $\Sigma = \langle DC, FS \rangle$, where $DC = \bigcup_{n \in \mathbb{N}} DC^n$ resp. $FS = \bigcup_{n \in \mathbb{N}} FS^n$ are respectively sets of *data constructors* and *defined function symbols* with associated arities. As notational conventions, we will assume $c, d \in DC$, $f, g \in FS$ and $h \in DC \cup FS$. We also assume that many countable variables (noted as X, Y, Z, etc.) are available. Given any set $\mathcal{X}$ of *variables*, we will consider the set $Exp_\Sigma(\mathcal{X})$ of all terms built from symbols in $\mathcal{X} \cup DC \cup FS$, and also the set $Term_\Sigma(\mathcal{X})$ of all terms built from symbols in $\mathcal{X} \cup DC$. Terms $l, r, e \in Exp_\Sigma(\mathcal{X})$ will be called *expressions*, while terms $s, t \in Term_\Sigma(\mathcal{X})$ will be called *constructor terms* or also *data terms*. Expressions without variables will be called *ground* or *closed*. Moreover, we will say that an expression e is in *head normal form* iff e is a variable X or has the form $c(\overline{e}_n)$ for some data constructor $c \in DC^n$ and some n-tuple of expressions $\overline{e}_n$.

A $\mathcal{TOY}$ program consists of *datatype*, *type alias* and *infix operator* definitions, and rules for defining *functions*. Its syntax is mostly borrowed from Haskell (Peyton-Jones 2002), with the remarkable exception that variables begin with upper-case letters whereas constructor symbols use lower-case, as function symbols do. In particular, functions are *curried* and the usual conventions about associativity of application hold.

Datatype definitions like `data nat = zero | suc nat` (which stands as a possible approach to define the natural numbers), define new (possibly polymorphic) *constructed types* and determine a set of *data constructors* for each type.

Types $\tau, \tau', \ldots$ can be constructed types, tuples $(\tau_1, \ldots, \tau_n)$, or functional types of the form $\tau \to \tau'$. As usual, $\to$ associates to the right. $\mathcal{TOY}$ provides predefined types such as `[A]` (the type of polymorphic lists, for which Prolog notation is used), `bool` (with constants `true` and `false`), `int` for integer numbers, or `char` (with constants `'a','b'`, ...).

A $\mathcal{TOY}$ program defines a set *FS* of functions. Each $f \in FS^n$ has an associated principal type of the form $\tau_1 \to \ldots \to \tau_m \to \tau$ (where τ does not contain $\to$). Number m is called the *type arity* of f and well-typedness implies that $m \geq n$. As usual in functional programming, types are inferred and, optionally, can be declared in the program.

We distinguish two important syntactic domains: *patterns* and *expressions*. Patterns can be understood as denoting data values, i.e., values not subject to further evaluation, in contrast to expressions, which can be possibly reduced by means of the rules of the program. Patterns $t, s, \ldots$ are defined by $t ::= X \mid (t_1, \ldots, t_n) \mid c\, t_1 \ldots t_n \mid f\, t_1 \ldots t_n$, where $c \in DC^m$, $n \leq m$, $f \in FS^m$, $n < m$, and t_i are also patterns. Notice that partial applications (i.e., application to less arguments than indicated by the arity) of c and f are allowed as patterns, which is then called a *higher order (HO) pattern*, because they have a functional type. Therefore function symbols, when partially applied, behave as data constructors. HO patterns can be manipulated as any other patterns; in particular, they can be used for matching or checked for equality. With this *intensional* point of view, functions become 'first-class citizens' in a stronger sense that in the case of 'classical' FP.

Expressions are of the form $e ::= X \mid c \mid f \mid (e_1, \ldots, e_n) \mid (e_1\ e_2)$, where $c \in DC$, $f \in FS$, and e_i are also expressions. As usual, application associates to the left and parentheses can be omitted accordingly. Therefore, $e\ e_1 \ldots e_n$ is the same as $(\ldots((e\ e_1)\ e_2)\ \ldots)\ e_n)$. Of course, expressions are assumed to be well-typed. *First order patterns* are a special kind of expressions which can be understood as denoting data values, i.e., values not subject to further evaluation, in contrast to expressions, which can be possibly reduced by means of the rules of the program.

Each function $f \in FS^n$ is defined by a set of conditional rules $f\ t_1 \ldots t_n = e \Longleftarrow l_1 == r_1, \ldots, l_k == r_k$ where $(t_1 \ldots t_n)$ form a tuple of linear (i.e., with no repeated variable) *patterns*, and e, l_i, r_i are *expressions*. No other conditions (except well-typedness) are imposed to function definitions. Rules have a conditional reading: $f\ t_1 \ldots t_n$ can be reduced to e if all the conditions $l_1 == r_1, \ldots, l_k == r_k$ are satisfied. The condition part is omitted if $k = 0$.

$\mathcal{TOY}$ includes a polymorphic version of the primitive equality constraint $seq :: A \to A \to bool$. The language provides the equality and disequality constraints == and /= to abbreviate $seq\ t\ s \to! true$ and $seq\ t\ s \to! false$ (resp.) Both constraints first request their arguments to be computed to head normal form, obtaining a variable or a total term. Thus, the symbol == stands for *strict equality*, which is the suitable notion (see e.g. (Hanus 1994)) for equality when non-strict functions are considered. With this notion, a condition `e == e'` can be read as: `e` and `e'` can be reduced to the same pattern. When used in the condition of a rule, == is better understood as a constraint (if it is not satisfyable, the computation fails).

A distinguished feature of $\mathcal{TOY}$ is that no confluence properties are required for the programs, and therefore functions can be *non-deterministic*, i.e., return several values for a given (even ground) arguments. For example, the rules `coin = heads` and `coin = tails` constitute a valid definition for the 0-ary non-deterministic function `coin`. Two reductions of `coin` are allowed, which lead to the values `heads` and `tails`. The system try in the first place the first rule, but, if backtracking is required by a later failure or by request of the user, the second rule is tried. Another way of introducing non-determinism in the definition of a function is by adding *extra* variables in the right side of the rules, as in `z_list = [0|L]`. Any list of integers starting by 0 is a possible value of `z_list`. Anyway, note that in this case only one reduction is possible.

The repertoire of $\mathcal{FD}$ constraints contains == and /=, that are truly polymorphic.Table 1 includes some of the predefined $\mathcal{FD}$ functions and operators supported, where relational constraints support reification (Marriot and Stuckey 1998). The propositional constraint `implication` posts to the solver a logical implication between the relational constraints `A` and `B`. `belongs` supports domain initialization to a set of values instead of a range as in `domain`. Finally, `sum List Op B` imposes a relational constraint between `B` and the sum of the elements of the list, and `count A List Op B` imposes a relational constraint between `B` and the number of elements in the list to be equal to `A`.

Table 1: Some of the $\mathcal{FD}$ Constraints and Operators

RELATIONAL
`(==), (/=), (#>), (#<), (#>=), (#<=) :: int → int → bool`
ARITMETICAL
`(#+), (#-), (#*), (#/) :: int → int → int`
`sum :: [int] → (int → int → bool) → int → bool`
COMBINATORIAL
`all_different :: [int] → bool`
`count :: int → [int] → (int → int → bool) → int → bool`
MEMBERSHIP
`domain :: [int] → int → int → bool`
`belongs :: [int] → [int] → bool`
PROPOSITIONAL
`implication :: int → (int → int → bool) → int → int → (int → int → bool)→ int → bool`

3. Problem Description

Once introduced the $\mathcal{TOY}(\mathcal{FD})$ system, the rest of the paper describes a real life optimization employee timetabling problem that can be seen as a particular instance of the NRP. Modeling and solving of this problem motivates the $\mathcal{TOY}(\mathcal{FD})$ usability. The problem comes from a technical department of the Spanish public television, where the employed workers must be scheduled to the requested shifts for *n* days. While the problem was formulated before in (R. González-del-Campo and F. Sáenz-Pérez 2007), here we present new requirements and problem formulation also embodying optimization.

Each day, several workers work at the company. On a working day three workers must work, covering shifts of 20, 22 and 24 hours, respectively. On a weekend two workers must work, covering two shifts of 24 hours. The company employs thirteen workers. Twelve of them are regular workers, and they are divided into three teams of four workers: $\{w_1, \ldots, w_4\}$ belong to t_1, $\{w_5, \ldots, w_8\}$ belong to t_2 and $\{w_9, \ldots, w_{12}\}$ belong to t_3. The extra worker e belongs to no team, and he is only selected by demand for coping with regular workers absences. Optimization arises in the problem because the company must pay regular workers for each extra hour they work, and extra worker hours are paid twice. Optimal schedule minimizes the extra hour payment.

The requirements any valid schedule must hold are the following: Each of the n days of the timetabling must be known as either working day or weekend. Each absence of worker w_i on day d_j must be provided. Each team must work each three days, working one day and resting two. Any shift of a day must be assigned to a unique worker. Each day, any worker must be assigned to either zero or one of the available shifts. Assigning no shift can be seen as assigning a shift of 0 hours. The extra worker can work any day d, but then he must rest on days $d+1$ and $d+2$. $Tight_{t,w}$ is a measure related to the shifts of kind s (0, 20, 22 or 24 hours) assigned to the regular workers of team t. Let us suppose that, by scheduling the timetabling, workers of t are assigned to k_1, k_2, k_3 and k_4 shifts of kind s. Then, $Tight_{t,w}$ represents the difference between the maximum and the minimum of these k_s. Shift distribution is constrained by $Tight$, representing the maximum value any $Tight_{t,w}$ can take. Scheduled timetabling contains $Total$ working hours, to be accomplished by regular workers. Assigning $Total/12$ hours per worker implies no extra hour payment, minimizing the optimization function.

4. Problem Implementation

In this section we present our modeling approach for solving the problem, which can be found at `http://gpd.sip.ucm.es/ncasti/TOY(FD).zip`. Due to its differences with (R. González-del-Campo and F. Sáenz-Pérez 2007) (problem decomposition, optimization, tightness of the shifts, etc.) modeling has been started from scratch. The data of each instance of the problem can be introduced by the user at command line, as well as to be included in a $\mathcal{TOY}$ file. First, we devote a subsection to describe the identification of decision variables. Then, next subsection presents a four stage process where the nature of the problem is exploited, decreasing the search space to be explored when looking for an optimal solution.

4.1. Decision Variables

Decision variables are represented by $\mathcal{FD}$ variables in a $13 \times n$ table where the columns represent the concrete days and the rows the concrete workers. We refer to this table as `timetable`. Each variable $timetable_{i,j}$ represents the shift assigned to worker w_i on day d_j, and it is initially assigned to the domain values $\{0, 20, 22, 24\}$.

The problem requirements requesting only one team to work each day, and each team to work each three days produces strong dependencies between the `timetable` variables. First, there is a dependency between the variables of day d, as if, for example, t_i works on d, the other two teams are precluded from working this day. Thus, their regular workers can be assigned to shifts of 0 hours. Second, there is a dependency between the variables of different days. In the previous example, t_i is also going to work on days $d+3$, $d+6$, etc., binding the variables for the workers of the other two teams to 0. And, also, t_i is not going to work on days $d+1$, $d+2$, $d+4$, $d+5$, etc., so their workers can be bound to 0 on those days. These connections reveal an independency between the different regular working teams. If a variable v_1 is susceptible to be bound to 0 by v_2 assignment to a shift, then we can model it with an `implication` constraint. However, this approach does not exploit the independency of the different teams, but treat the whole table as a single problem, making search space to remain at $4^{13 \times n}$. Thus, to model and solve an instance of the problem we are going to manage a smaller table, in order to reduce the search space to be explored by reducing both the number of variables and the size of their domains.

4.2. Solving Process

Our modeling approach follows a four stage process to schedule `timetable`. First stage is *Team Assignment*, and it concerns with assigning regular workers teams to days.

It is equivalent to trigger the `implication` constraints associated to the variable dependencies explained before. It allows us to detect the workers susceptible of being assigned to shifts for each day. To explore the whole search space, each feasible assignment of teams to days must be computed. For each feasible assignment, second, third and forth stages are performed. Second stage is *Timetabling Generation*, and it concerns with the creation of the simpler problem to be solved. On the one hand, after a team assignment, each day only five workers are susceptible of being scheduled to cover the shifts: the team regular workers and e. Thus, this stage creates `tt`, a $5 \times n$ table which can be mapped to the sub-table of `timetable` associated to the concrete team assignment performed on first stage. On the other hand, team independence is exploited in `tt`, obtaining the three sub-problems $\texttt{tt}_1$, $\texttt{tt}_2$ and $\texttt{tt}_3$, which can be solved independently. Third stage is *Timetabling Solving*, and it concerns with scheduling each $\texttt{tt}_i$. Finally, fourth stage is *Timetabling Mapping*, and it concerns with using the team assignment and the scheduled `tt` to construct the original `timetable`, which is outcome as a result to the user.

The main function `schedule` performs the four stage process to solve each particular instance of the problem. It contains six parameters. First four (`N`, `Abs`, `DClass` and `Tight`) represent necessary timetabling input data. Last two (`W`, `SS`) specify directives to the $\mathcal{FD}$ solver. `N` represents the number of days. `Abs` is a list of pairs (w_i, d_j) representing regular workers planned absences. `DClass` is a list of `N` `dayType` elements representing day classification. `Tight` was already explained in problem description. `W` is a `labelingOrder` representing the order of the variables to be labeled (`dayOrder` or `workerOrder`). `SS` is a `labelingStrategy` representing a particular variable selection strategy for labeling (`firstFail` or `firstUnbound`). Function `schedule` returns the tuple `(Timetabling, ExtraHours)` as a result, where `ExtraHours` represents the number of extra hours associated to the optimal schedule. As to explore the whole search space each team assignment must be computed, `collect` is applied to function `schedule` to get all solutions. We devote next four sections to describe each stage of the process.

Team Assignment In this stage no shift is assigned to a worker, but regular teams are assigned to days. This allows us to explore a subset of the `timetable` search space. In concrete, as there are three teams working each three days, there are up to six possible assignments of teams to days (each of them representing 1/6th of the `timetable` search space). Departing from a team assignment, `tt` is constructed, scheduled and mapped to `timetable`. In order to find the optimal schedule, all team assignments must be explored. This behavior is only possible because $\mathcal{TOY}(\mathcal{FD})$ allows reasoning with models, where backtracking and multiple search strategies (placed at any point of the problem description) are supported. Let us explain it in detail. Departing from a team assignment solution, the rest of the stages imply the use of new variables, constraints and search strategies. Performance of this stage possibly obtain a sub-optimal schedule (sub-optimal in the sense that it is optimal for that 1/6th `timetable` search space). Then, next feasible team assignment must be performed, implying backtracking to the *Team Assignment* stage. This also implies the removing of the variables, constraints and search strategies posted by second, third and fourth stages. Stage *Team Assignment* takes part on reducing the size of the search space of the timetabling schedule by acting in three different ways:

- By managing `tt` instead of `timetable`, up to eight variables per day are directly saved, which represent the 60% of the total `timetable` variables.
- By associating `tt` creation and solving to a team assignment, detection of non-feasible team assignments avoids exploring 1/6th of the remaining search space. Our main aim with this stage is to filter only those team assignments possibly leading to solutions. That is, those that, at least, provide for each day enough workers to accomplish the shifts. Thus, we have to take into account both regular worker absences and the two days resting constraint of e. On the one hand, let us suppose that `Abs = [(w_1,4), (w_2,4), (w_3,4)]` and that day 4 is classified as `workingDay`, where 20, 22 and 24 shifts must be assigned to workers. Then, it is for sure that t_1 can not be assigned to days 1, 4, etc, as on day 4 there would be only two available workers (w_4 and e) to cover three shifts. On the other hand, let us suppose that day 4 is classified as a `weekendDay`, where two shifts of 24 hours must be assigned. In this setting, t_1 can be assigned to days 1, 4, etc. assigning the two shifts of day 4 to w_4 and e. But, as t_1 requests e for day 4, the latter can not be requested on days 2, 3, 5, 6, due to its two days resting constraint. Two of these days are assigned to team t_2 and the other two to team t_3. So, for requesting e one day, t_1 disallows teams t_2 and t_3 to request two days from e. If, due to `Abs` planned absences, e needs to be requested on one of those days, then t_1 can not be assigned to days 1, 4, etc. Last but not least, as for each feasible team assignment (at least) two of each three days e must be 0, we can not save but bound (at least) another 5% of the `timetable` variables. Thus, `tt` will only contain as much as the 35% of the `timetable` variables.
- As e is susceptible of working any day, it acts as a link between the three working teams. But, as any feasible team assignment entails the resting constraint of e in the whole schedule, we can freely split it into three independent sub-problems, and solve independently each one. As each sub-problem only contains 33% of the variables, we have replaced the effort of solving a complex problem to the effort of solving three exponentially simpler problems.

In summary, initial 4^k search space is reduced to $4^{0.35k}$. But, as we solve the three independent sub-problems independently, the search space becomes $3 \times 4^{0.12k}$. As we need to solve the six configurations to find the one minimizing the extra hour payment, the total search space of our approach is $6 \times (3 \times 4^{0.12k}) \equiv 18 \times 4^{0.12k}$.

Let us now present our approach for modeling this stage. First, our main aim is to assign a concrete team to each day

of the timetabling. Second, we need to ensure that a minimal number of regular workers are available to cover all shifts for each day. Third, we need to ensure e resting constraint. Thus, we need three lists of N new variables: `[D1, D2, D3, ..., Dn]`, where `Di` represents the concrete team assigned to day i, `[A1, A2, A3, ..., An]`, where `Ai` represents the number of absences of the regular workers of the selected team to day `i` and `[E1, E2, E3, ..., En]`, where `Ei` represents if e is requested to work by selected team to day `i`. At this point we can see that each `Ai` and `Ei` are related to the concrete value `Di` takes. In particular, for `Ai` we are only interested in the number of total absences of the regular workers of the selected team `Di` states. Also, `Ei` would be bound to 1 only if this team does not provide enough workers to accomplish shifts of this day. To build the list of $\mathcal{FD}$ variables, the function `genVList` is used:

```
genVList:: [A]
genVList = [X | genVList]

workEach3Days:: [A] -> bool
workEach3Days L = true <==
    length L < 4
workEach3Days [L1,L2,L3,L1|R] =
    workEach3Days [L2,L3,L1|R]
```

It generates a polymorphic infinite list, and contains a unique rule with extra variables on its right side. By applying `take N genVList`, lazy evaluation is performed to compute only `N` new variables. To ensure that each team (`D` variables) works each three days, function `workEach3Days` is applied. It recursively checks if a list contains more than three elements, to unify via pattern matching the first and the fourth ones. Then, `D` becomes `[D1, D2, D3, D1, D2, ...]`. Finally, a `domain` constraint attributes `(D1, D2, D3)` as $\mathcal{FD}$ variables with domain value set $\{1, 2, 3\}$. An `allDifferent` constraint `(D1, D2, D3)` ensures each team work each three days.

Now, information contained in `Abs` is reorganized to fit it into a new data structure, more suitable for posting constraints on `A` and `E`. Function `order_abs:: int` $\rightarrow$ `[(int,int)]` $\rightarrow$ `[[int]]` converts `Abs` in `OAbs`, a list of lists of integers `[[int]]` data structure. For each day it contains a list indicating which regular workers are absent. To create `OAbs`, the elements of tuples in `Abs` are firstly swapped and then sorted by days. Function `sortList` orders (with quicksort) its input list:

```
sortList :: [(int,int)] -> [(int,int)]
sortList [] = []
sortList [(X,Y)|Xs] =
  sortList (filter (ord (X,Y)) Xs)
  ++ [(X,Y)] ++
  sortList (filter (not . ord (X,Y)) Xs)
```

Its second rule uses first element `(X,Y)`, and relies on both `ord` and functional composition `not . ord` to compute lower and greatest elements (resp.):

```
ord:: (int,int) -> (int,int) -> bool
ord (X,Y) (Z,T) = true <==
  (X > Z)\/((X == Z)/\(Y > T)) == true
ord (X,Y) (Z,T) = false <==
  (X < Z)\/((X == Z)/\(Y <= T)) == true
```

In particular, this function is non-deterministic, as its two rules receive the same argument to compute different results (depending on the conditions). Once `Abs` is sorted, it is filtered by days to compute `OAbs`.

On the one hand, `DClass` is used to initialize the domain of `A` variables. While `workingDay` implies a domain `0,..,2`, `weekendDay` implies a domain `0,..,3`. With this, enough workers to accomplish required shifts is ensured. On the other hand, `E` variables are initialized to the domain `0,..,1`, and the constraint `sum [Ei,Ei+1,Ei+2]` $<= 1$ is posted on each three consecutive elements to ensure the resting constraint e. To relate `D` with `A` and `E` two new `[[int]]` data structures are built, `ATD` and `ETD`, computing absences per team and day and ensuring e per team and day (resp.) Then, `implication` constraints relate each possible value `Di` takes (1, 2 or 3) with the binding of `Ai` to `ATD[i,1]`, `ATD[i,2]` or `ATD[i,3]` (resp.) Same situation happens with `Ei` and `ETD`.

Finally, `labeling [] (take 3 D)` is used to start searching for feasible team assignments.

Timetabling Generation This stage performs three actions: create `tt`, bind to 0 as much as possible of its variables, and split it into $\texttt{tt}_1$, $\texttt{tt}_2$ and $\texttt{tt}_3$ (in order to reduce the effort of solving the problem to the effort of solving three exponentially simpler problems).

Table `tt` is typed as `[[int]]`. To create it, `take N (repeat (take 5 genVList))` cannot be used, as $\mathcal{TOY}$ call-time choice would lazily compute the same variables for each day. This is due to the fact that the argument of `repeat` is computed just once. Instead, `tt` is created by using the function `genTT [] N`, which computes a different list of variables for each day, by explicitly computing `N` times `take 5 genVList`.

```
genTT:: [[A]] -> int ->  [[A]]
genTT L 0 = L
genTT L N =
  genTT [(take 5 genVList)|L] (N-1) <== N > 0
```

Second, to bind as much variables as possible to 0 we create the `[[int]]` extra data structure `ZeroCal`. We distinguish between the regular worker absences and the days that e is not requested. To identify regular worker absences `OAbs` is filtered to deliver only that absences related to the regular workers of the selected team. To this end, list `D` is used. Also, for each day for which e is not requested (`Ei` = 0), `ZeroCal` is increased by 5. Finally, once `tt` and `ZeroCal` are built, both structures are mapped with function `putZeros`, which zeroes selected variables.

Third, once `tt` has been built with the minimum number of variables, it is splitted into the three different subproblems, one per team. As both `N` and `DClass` are requested for solving each sub-problem, they must also be split. Finally, the standard number of hours each regular worker should ideally accomplish is also requested by third stage to solve a single sub-problem. It is computed with the expression `(#/12) (foldl workHours 0 DClass) == Hours`. On the one hand, $\mathcal{TOY}(\mathcal{FD})$ capability of using both curried functions and constraints is used. In this case the constraint `(#/12)` is waiting to be applied to an `int`. On the other hand, it is applied to the

total number of hours of tt, which is computed by using the FP standard function foldl, and is presented as a higher order application to make the expression more compact. This foldl uses an initial 0 as accumulator. Function workHours adds 66 or 48 to the accumulator, depending on whether it deals with a workingDay or with a weekendDay (resp.) It is important to remark that a common number of standard hours for the twelve regular workers is maintained, instead of creating a specific number for the regular workers of each team.

Timetabling Solving As tt, N and DClass are split into three independent sub-problems, they can be solved separately. So, to improve performance, it could be possible to send the three sub-problems to different threads. Going even one step further, our solving process could be changed, making *Team Assignment* stage to collect first all the feasible team assignments, and second stage compute all its tt's and sub-problem tables. At that point, we would contain as much as 18 different sub-problems to be solved, and third stage could send them to different threads.

However, $\mathcal{TOY}$ does not support multi-threading up to now. Thus, we solve tt by solving sequentially tt_1, tt_2 and tt_3. In any case, as tt optimal schedule is the one which minimizes the extra hour payment, by monotonicity it could be computed as the sum of the optimal schedules for tt_1, tt_2 and tt_3.

Solving each tt_i implies several tasks. First, an initial domain on its variables must be imposed. That is, {0, 20, 22, 24} for workingDay and {0, 24} for weekendDay. Then, we impose an allDifferent constraint for workingDay, and a count constraint for setting two variables to 24 for weekendDay. But, if e is requested this can be simplified. e is only requested when team working does not provide enough regular workers to accomplish all shifts. In those cases we can ensure that both e and remaining regular workers are going to work (a value different from 0). Thus, for workingDay and weekendDay the case on which either e is requested (an unbound variable) or not (bound to value 0) is distinguished. If e is requested, on weekendDay regular workers rows are traversed and the unique variable not bound, say X, is collected. Then, both X and e can be bound to 24 without the need of posting $\mathcal{FD}$ constraints.

Second, we take into account Tight, to ensure generated schedules to maintain an homogeneous distribution of shifts. Let us suppose that we have two different schedules implying 36 extra hours. However, while in the first one the 36 extra hours are accomplished by a single regular worker, the second one divides it into 9 hours for the four regular workers of a working team. From the point of view of optimality, both solutions are equal. However, the second one is fairer with the distribution of the work. We let Tight to be an input parameter to be introduced by the user in order to make homogeneous the distribution of the work. As we said in problem description, we measure Tight by each working team and each shift. If we make Tight = 0 then all the regular workers of all the teams must be assigned to the same number of 0, 20, 22 and 24 shifts. If we make Tight = 1, then we allow each regular working team $\{w_i, w_{i+1}, w_{i+2}, w_{i+3}\}$ to assign different shifts to their workers. For example, let us suppose that the maximum number of 20 hours shifts assigned to a worker is k1, and the minimum is k2. Then Tight = 1 constraints that the difference between k1 and k2 is not greater than 1. It is important to remark that we have decided to include Tight as an input parameter, instead of wrapping it within the cost function. Our objective is to minimize the extra hour payment, not Tight.

Function ts is used to constrain with Tight a single working team and a single shift:

```
tS T [W1L,W2L,W3L,W4L,EL] S = true <==
  count S W1L (#=) A,
  count S W2L (#=) B,
  count S W3L (#=) C,
  count S W4L (#=) D,
  domain [A#-B, A#-C, A#-D, B#-C,
          B#-D, C#-D] (-T) T
```

In particular, expression tS T (transpose SubTT) WS is used to, first, transpose the list tt_i in order to access its variables by workers, instead of by days. tS creates four new $\mathcal{FD}$ variables SW1,...,SW4 representing the amount of shifts assigned to each of the regular workers of the working team. Second, to impose an homogeneous distribution we impose the difference of this variables to be in the domain (-T),...,T. Here we want to remark the $\mathcal{TOY}(\mathcal{FD})$ functional notation capabilities, that allow us to directly express the subtraction of each pair of variables in the domain constraint, instead of creating new variables. Third, we need to generate the cost function EHours == X1 # + X2 # + X3 # + X4 # +(2 # * X5). While X1,...,X4 represent the extra hours of the regular workers, X5 is the number of extra hours of e. Again we need to use transpose SubTT as we need to access its variables by workers. In this setting, X5 is computed with a sum constraint of e working hours. In the case of regular workers, we need four more sum constraints. But, in this case the generated $\mathcal{FD}$ variables must be compared with standard number of working hours. Thus, we use implication constraints. If a worker has done extra hours, then Xi represents that number of extra hours. But, if the worker has done less than standard hours, then Xi = 0.

Finally, a new labeling over tt_i is applied. The input parameters W and SS are taken into account to follow a particular search strategy. SS selects the particular variable selection strategy: firstUnbound or firstFail. W enumerates the tt_i variables by days or by workers. The labelingOrder is relevant for both labeling strategies. As tt_i variables contain few values in their domain, many ties will be produced when selecting variables by first fail, and the variable enumerated first will be the one selected. All the cases contain the option toMinimize ExtraHours, ensuring that optimal schedule (minimizing the extra hour payment) is selected as a solution.

Timetabling Mapping By scheduling the three tt_i (binding all their variables) all the variables of tt are indirectly bound. We recall here that the user is requested to use

`collect` to trigger exploration of all feasible team assignments. Users can then either check each sub-optimal solution or map a minimum function to the collected solutions to select the optimal one.

5. Paradigm Comparison

Let us discuss about the benefits of modeling this problem in CFLP($\mathcal{FD}$). First, we summarize the advantages the logic component offer us in CLP($\mathcal{FD}$) and CFLP($\mathcal{FD}$) in contrast to CP($\mathcal{FD}$). Then, we point out the advantages the functional component offer us in CFLP($\mathcal{FD}$) in contrast to CLP($\mathcal{FD}$).

CP($\mathcal{FD}$) is not thought to reason with models. First, the constraint set is imposed and then several labeling search strategies over disjoint variable sets can be declared. On the one hand, both tasks cannot be mixed. On the other hand, the set of imposed constraints is static and not meant to change. Both CLP($\mathcal{FD}$) and CFLP($\mathcal{FD}$) add to CP($\mathcal{FD}$) allow this task to be dynamic. In addition, the logic component eased the modeling and solving of the employee timetabling problem. We have pointed out the benefits of splitting this problem: Problem division, early detection of unfeasible solutions and reduction of $\mathcal{FD}$ variables. Our approach consists of posting a subset of the constraints of the problem and performs a labeling search strategy (obtaining a feasible team assignment). Then, departing from each obtained partial solution, we dynamically create the sub-problems to be solved. Finally, we post the rest of the constraint set over these subproblems, and we use parameterized labeling strategies to find the optimal schedule. Under this scenario, our model can not be directly translated into a classical CP approach with a requested isolation between constraint and search descriptions. To follow our CFLP($\mathcal{FD}$) approach described before, in CP($\mathcal{FD}$) we would need to create several CP models, and a general script to coordinate them. If we decide to keep it in a single model, then we need to create the table of $13 \times n$ $\mathcal{FD}$ variables to represent assigned shifts. But, only by doing that, we reach again the $4^{13 \times n}$ initial search space we depart from in our CFLP($\mathcal{FD}$) implementation (and that we reduce dramatically with our approach).

CFLP($\mathcal{FD}$) provides a sugaring syntax for LP predicates and thus any pure CLP($\mathcal{FD}$)-program can be straightforwardly translated into a CFLP($\mathcal{FD}$) program (Fernández et al. 2007). But, in addition, CFLP($\mathcal{FD}$) includes a functional component. This has helped us easing the task of modeling the calendar problem, as some of the extra capabilities CFLP($\mathcal{FD}$) enjoys have been used: (a) Types. Our system is strongly typed, and it has eased the modeling development and maintenance, by finding bugs at compile-time. (b) Polymorphic variables. (c) Lazy evaluation. Function arguments are evaluated to the required extent (the call-by-value used in LP vs. the call-by-need used in FP) (Peyton-Jones 1987). By this, we have managed infinite structures. (d) Call-time choice, allowing shared terms to be computed just once. (e) Higher order. Its use has simplified modeling. (f) Currying. Both curried functions and constraints have been used. (g) Functional notation, by which we have saved some $\mathcal{FD}$ variables. (h) Non-determinism rules have been used in some functions. (i) Function composition. It has allowed us to use more compact expressions.

6. Performance

In this section we present results for solving three instances of the problem. Each instance is assumed to start on Monday, where any week contains five working days followed by weekend. The three instances solve one, two and three weeks respectively, where the planned absences are shared among the instances: $\texttt{Abs} \equiv \{(w_1, 1),\ (w_2, 1),\ (w_5, 1),\ (w_5, 6), (w_6, 1), (w_6, 6), (w_7, 1), (w_7, 6), (w_{10}, 1), (w_{10}, 6),\ (w_{11}, 1),\ (w_{11}, 6),\ (w_{12}, 1),\ (w_{12}, 6)\}$. Thus, there will be only two feasible team assignments, as on day 1 only $t1$ can work. The two feasible team assignments correspond with permutation of $t2$ and $t3$ over days 2 and 3, respectively. Each assignment will request e on days 1 and 6. $\texttt{Tight} \equiv 1$ is used allow little differences between the regular workers of each team. The last two parameters specify the search strategy to be accomplished, as defined in Section 4.

Section 5 pointed out the benefits of using CFLP($\mathcal{FD}$) for modeling the problem. Now we are also interested in measuring the performance impact of its underlying lazy narrowing mechanism, in order to test if the framework maintains a good trade-off between expressivity and efficiency. Lazy narrowing elapsed time is computed via subtracting original instance elapsed time to the $\mathcal{FD}$ constraint solving elapsed time. To compute the latter one we isolate the concrete set of $\mathcal{FD}$ constraints being sent to the solver. Once obtained this set we re-formulate the instance via a new program, which only contains a function explicitly posting it. Next table summarizes the results. First column represents the number of weeks of the instance. As second and third columns represent the elapsed time of the original and constraint-only programs respectively (measured in milliseconds), the lazy narrowing overhead is presented on fourth column by simply computing their ratio. While for each measurement we have applied our four possible combinations of variable order and strategy selection explained in Section 4, in this table we only present the results of the fastest, to which we devote last column. Benchmarks have been run in a machine with an Intel Dual Core 2.4GHz processor and 4GB RAM memory and Windows XP SP3. $\mathcal{TOY}(\mathcal{FD})$ has been developed with SICStus Prolog, version 3.12.8, and ILOG CP 1.6, with ILOG Concert 12.2 and ILOG Solver 6.8 libraries. Microsoft Visual Studio 2008 tools were used for compiling and linking the $\mathcal{TOY}(\mathcal{FD})$ interface to ILOG CP.

Weeks	$\mathcal{TOY}(\mathcal{FD})$	$\mathcal{FD}$	Overhead	Strat.
1	1,027	859	20%	dO
2	3,404	3,100	10%	wO
3	199,359	195,609	2%	wO

Obtained results encourages our approach. On the one hand, we can solve timetables of up to three weeks in a reasonable amount of time. On the other hand, we can see that the lazy narrowing overload is also negligible. Its overload ranges from 20% for the smallest instance being measured (which is solved in less than one second) to a 2-3% for the

largest instance being measured (which is solved in more than three minutes). So, as problem scales, the impact of the lazy narrowing decreases, becoming very few or instead irrelevant for difficult CSPs involving long time during search.

7. Conclusions

CP is a suitable paradigm to tackle timetabling problems, as it provides expressive languages for describing the constraints and powerful solver mechanism for reasoning with them. However, it neither provides general purpose programming features nor a modeling reasoning framework, as CLP and CFLP frameworks do. In this paper we have presented an employee timetabling problem that puts some evidence of the benefits these two properties provide for the modeling and solving of this particular problem. On the one hand, users can benefit from the extra expressivity those frameworks provide, without loosing constraint solving capabilities. On the other hand, modeling framework allows us to dramatically reduce the search space to be explored for obtaining the optimal schedule, without supposing a big overload due to its underlying operational mechanism.

While our problem can not be directly translated to CP using a classical approach (due to the isolation between constraints and search strategies), there exist several ways of exploiting problem independence, relying on a distributed framework (Meisels and Kaplansky 2002). As future work we plan to make an in-depth comparison of CP($\mathcal{FD}$), CLP($\mathcal{FD}$) and CFLP($\mathcal{FD}$) for solving our problem. We plan to model the problem in CP, both with a distributed approach and with the classical approach. Also, we plan to model the problem in CLP. It would put much more evidence between the trade-off of expressivity and efficient our framework is. On the one hand, when modeling in CLP($\mathcal{FD}$) we can not take advantage of the expressivity of our system. On the other hand, as our system is built on top of SICStus, we can directly assess the impact of our lazy narrowing operational system.

References

[Brauner et al. 2005] Brauner, N.; Echahed, R.; Finke, G.; Gregor, H.; and Prost, F. 2005. Specializing narrowing for timetable generation: A case study. In *PADL*, 22–36.

[Burke et al. 2004] Burke, E. K.; Causmaecker, P. D.; Berghe, G. V.; and Landeghem, H. V. 2004. The state of the art of nurse rostering. *J. Scheduling* 7(6):441–499.

[Burke, Kendall, and Soubeiga 2003] Burke, E. K.; Kendall, G.; and Soubeiga, E. 2003. A tabu-search hyperheuristic for timetabling and rostering. *Journal of Heuristics* 9:451–470.

[Burke, Li, and Qu 2010] Burke, E. K.; Li, J.; and Qu, R. 2010. A hybrid model of integer programming and variable neighbourhood search for highly-constrained nurse rostering problems. *European Journal of Operational Research* 203(2):484–493.

[Fernández et al. 2007] Fernández, A. J.; Hortalá-González, T.; Sáenz-Pérez, F.; and del Vado-Vírseda, R. 2007. Constraint Functional Logic Programming over Finite Domains. *TPLP* 7(5): 537–582.

[González-Moreno et al. 1999] González-Moreno, J.; Hortalá-González, M.; López-Fraguas, F.; and Rodríguez-Artalejo, M. 1999. An approach to declarative programming based on a rewriting logic. *Journal of Logic Programming* 40:47–87.

[Hanus 1994] Hanus, M. 1994. The integration of functions into logic programming: a survey. *The Journal of Logic Programming* 19-20:583–628.

[Hanus 1999] Hanus, M. 1999. Curry: a truly integrated functional logic language. `http://www-ps.informatik.uni-kiel.de/currywiki/`.

[Jaffar and Maher 1994] Jaffar, J., and Maher, M. 1994. Constraint logic programming: a survey. *The Journal of Logic Programming* 19-20:503–581.

[López-Fraguas, Rodríguez-Artalejo, and Vado-Vírseda 2004] López-Fraguas, F.; Rodríguez-Artalejo, M.; and Vado-Vírseda, R. 2004. A lazy narrowing calculus for declarative constraint programming. In *PPDP'04, 43–54*. ACM.

[Marriot and Stuckey 1998] Marriot, K., and Stuckey, P. J. 1998. *Programming with constraints*. Cambridge, Massachusetts: The MIT Press.

[Meisels and Kaplansky 2002] Meisels, A., and Kaplansky, E. 2002. Scheduling agents - distributed timetabling problems(disttp). In *PATAT*, 166–180.

[Métivier, Boizumault, and Loudni 2009] Métivier, J.-P.; Boizumault, P.; and Loudni, S. 2009. Solving nurse rostering problems using soft global constraints. In *CP*, 73–87.

[Moz and Pato 2007] Moz, M., and Pato, M. V. 2007. A genetic algorithm approach to a nurse rerostering problem. *Computers & OR* 34(3):667–691.

[PAKCS] PAKCS. `http://www.informatik.uni-kiel.de/pakcs`.

[Peyton-Jones 1987] Peyton-Jones, S. 1987. *The implementation of functional programming languages*. Englewood Cliffs, N.J.: Prentice Hall.

[Peyton-Jones 2002] Peyton-Jones, S. 2002. Haskell 98 language and libraries: the revised report. Technical report. `http://www.haskell.org/onlinereport/`.

[R. González-del-Campo and F. Sáenz-Pérez 2007] R. González-del-Campo and F. Sáenz-Pérez. 2007. Programmed search in a timetabling problem over finite domains. *ENTCS* 177: 253–267.

[Tsang 1993] Tsang, E. 1993. *Foundations of constraint satisfaction*. London and San Diego: Academic Press.

The Distance-Optimal Inter-League Schedule for Japanese Pro Baseball

Richard Hoshino and Ken-ichi Kawarabayashi
National Institute of Informatics, 2-1-2 Hitotsubashi, Chiyoda-ku, Tokyo 101-8430, Japan

Abstract

Nippon Professional Baseball (NPB) is Japan's largest and most well-known professional sports league, with over 22 million fans each season. Each NPB team plays 24 *inter-league* games during a five-week period each spring. In this paper, we solve the problem of determining the best possible NPB inter-league schedule that minimizes the sum total of distances traveled by all teams. Despite the NP-completeness of the general problem with $2n$ teams, we show that for the 12-team NPB, the distance-optimal inter-league schedule can be determined from two heuristics that drastically cut down the computation time. Using these heuristics, we generate the best possible schedule when the home game slots are *uniform* (i.e., every team in one league plays their home games at the same time), and when they are *non-uniform*. Compared to the 2010 NPB inter-league schedule, our optimal schedules reduce the total travel distance by 15.3% (7849 km) in the uniform case, and by 16.0% (8184 km) in the non-uniform case.

Introduction

Nippon Professional Baseball (NPB) is Japan's most popular professional sports league, with annual revenues topping one billion U.S. dollars. In terms of actual attendance, the NPB ranked second in the world among all professional sports leagues, ahead of the National Football League (NFL), the National Basketball Association (NBA), and the National Hockey League (Wikipedia 2011).

The NPB is split into the six-team Pacific League and the six-team Central League. Each team plays 144 games during the regular season, with 120 *intra*-league games (against teams from their own league) and 24 *inter*-league games (against teams from the other league). To complete these $\frac{1}{2} \times 12 \times 144 = 864$ games, the teams travel long distances from city to city, primarily by airplane or bullet-train. During the 2010 regular season, Pacific League teams traveled a total of 153940 kilometres to play intra-league games, while the more closely-situated Central League teams traveled 79067 kilometres (Hoshino and Kawarabayashi 2011b).

By reformulating intra-league optimization as a shortest path problem, the authors determined a distance-optimal schedule that retained all of the NPB constraints that ensured competitive balance, while reducing total Pacific League team travel by 25.8% (nearly 40000 kilometres) and Central League team travel by 26.8% (over 21000 kilometres).

The motivation for this paper is to extend our NPB analysis to inter-league play, to determine whether the 2010 schedule requiring 51134 kilometres of total team travel can be improved, and perhaps even minimized to optimality. The rationale for our paper is timely, given current global economic and environmental concerns. Implementing a distance-optimal schedule would help this billion-dollar sports league be more efficient and effective, saving money, time, and greenhouse gas emissions.

The paper proceeds as follows. We formalize the NPB inter-league problem in Section 2, providing the locations of the twelve teams as well as the inter-league schedule from the 2010 regular season. We use this to explain the concepts of *uniform* and *non-uniform* tournament scheduling, to motivate the *Bipartite Traveling Tournament Problem (BTTP)* in Section 3. Despite the NP-completeness of the general problem with $2n$ teams, we describe two heuristics that allow us to solve *BTTP* for the 12-team NPB, which we do in Section 4. In Sections 5 and 6, we present uniform and non-uniform inter-league schedules requiring 43285 and 42950 kilometres of total travel, respectively. We prove these schedules are optimal, achieving a 15.3% and 16.0% reduction in travel distance as compared to the 2010 NPB schedule.

The 2010 NPB Inter-League Schedule

Each NPB team plays in a home city somewhere within Japan, whose location is marked in Figure 1.

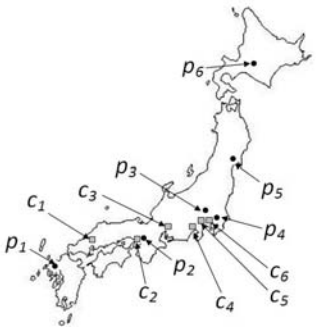

Figure 1: Location of the 12 teams in the NPB.

Let us explain the labelling. The Pacific League teams are p_1 (Fukuoka Hawks), p_2 (Orix Buffaloes), p_3 (Saitama Lions), p_4 (Chiba Marines), p_5 (Tohoku Eagles), and p_6 (Hokkaido Fighters). The Central League teams are c_1 (Hiroshima Carp), c_2 (Hanshin Tigers), c_3 (Chunichi Dragons), c_4 (Yokohama Baystars), c_5 (Yomiuri Giants), and c_6 (Yakult Swallows). For readability, we will refer to the teams by their labels rather than their full names.

Table 1 provides the 12×12 NPB *distance matrix* D, representing the distances between the home stadiums of each pair of teams in $\{p_1, \ldots, p_6, c_1, \ldots, c_6\}$. All distances are in kilometres. By symmetry, $D_{x,y} = D_{y,x}$ for any two teams x and y, and all diagonal entries are zero.

	c_1	c_2	c_3	c_4	c_5	c_6	p_1	p_2	p_3	p_4	p_5	p_6
c_1	0	323	488	808	827	829	258	341	870	857	895	1288
c_2		0	195	515	534	536	577	27	577	564	654	1099
c_3			0	334	353	355	742	213	396	383	511	984
c_4				0	37	35	916	533	63	58	364	886
c_5					0	7	926	552	51	37	331	896
c_6						0	923	554	48	39	333	893
p_1							0	595	958	934	1100	1466
p_2								0	595	582	670	1115
p_3									0	86	374	928
p_4										0	361	904
p_5											0	580
p_6												0

Table 1: The 12×12 NPB distance matrix.

In NPB inter-league play, each team in the Pacific League $P = \{p_i : 1 \leq i \leq 6\}$ plays four games against all six teams in the Central League $C = \{c_i : 1 \leq i \leq 6\}$, with one two-game set played at the home of the Pacific League team, and the other two-game set played at the home of the Central League team. All inter-league games take place during a five-week stretch between mid-May and mid-June, with no intra-league games occurring in that period.

Table 2 provides the 2010 NPB inter-league schedule, listing the twelve *sets*, with one set in each *time slot*, and two games in each set. In this table, as with all other schedules presented in this paper, home games are marked in bold.

	1	2	3	4	5	6	7	8	9	10	11	12
c_1	$\boldsymbol{p_5}$	$\boldsymbol{p_6}$	p_2	p_1	$\boldsymbol{p_3}$	$\boldsymbol{p_4}$	p_5	p_6	$\boldsymbol{p_1}$	$\boldsymbol{p_2}$	p_4	p_3
c_2	$\boldsymbol{p_6}$	$\boldsymbol{p_5}$	p_1	p_2	$\boldsymbol{p_4}$	$\boldsymbol{p_3}$	p_6	p_5	$\boldsymbol{p_2}$	$\boldsymbol{p_1}$	p_3	p_4
c_3	$\boldsymbol{p_1}$	$\boldsymbol{p_2}$	p_4	p_3	$\boldsymbol{p_5}$	$\boldsymbol{p_6}$	p_1	p_2	$\boldsymbol{p_4}$	$\boldsymbol{p_3}$	p_5	p_6
c_4	$\boldsymbol{p_3}$	$\boldsymbol{p_4}$	p_5	p_6	$\boldsymbol{p_1}$	$\boldsymbol{p_2}$	p_3	p_4	$\boldsymbol{p_5}$	$\boldsymbol{p_6}$	p_1	p_2
c_5	$\boldsymbol{p_4}$	$\boldsymbol{p_3}$	p_6	p_5	$\boldsymbol{p_2}$	$\boldsymbol{p_1}$	p_4	p_3	$\boldsymbol{p_6}$	$\boldsymbol{p_5}$	p_2	p_1
c_6	$\boldsymbol{p_2}$	$\boldsymbol{p_1}$	p_3	p_4	$\boldsymbol{p_6}$	$\boldsymbol{p_5}$	p_2	p_1	$\boldsymbol{p_3}$	$\boldsymbol{p_4}$	p_6	p_5
p_1	c_3	c_6	$\boldsymbol{c_2}$	$\boldsymbol{c_1}$	c_4	c_5	$\boldsymbol{c_3}$	$\boldsymbol{c_6}$	c_1	c_2	$\boldsymbol{c_4}$	$\boldsymbol{c_5}$
p_2	c_6	c_3	$\boldsymbol{c_1}$	$\boldsymbol{c_2}$	c_5	c_4	$\boldsymbol{c_6}$	$\boldsymbol{c_3}$	c_2	c_1	$\boldsymbol{c_5}$	$\boldsymbol{c_4}$
p_3	c_4	c_5	$\boldsymbol{c_6}$	$\boldsymbol{c_3}$	c_1	c_2	$\boldsymbol{c_4}$	$\boldsymbol{c_5}$	c_6	c_3	$\boldsymbol{c_2}$	$\boldsymbol{c_1}$
p_4	c_5	c_4	$\boldsymbol{c_3}$	$\boldsymbol{c_6}$	c_2	c_1	$\boldsymbol{c_5}$	$\boldsymbol{c_4}$	c_3	c_6	$\boldsymbol{c_1}$	$\boldsymbol{c_2}$
p_5	c_1	c_2	$\boldsymbol{c_4}$	$\boldsymbol{c_5}$	c_3	c_6	$\boldsymbol{c_1}$	$\boldsymbol{c_2}$	c_4	c_5	$\boldsymbol{c_3}$	$\boldsymbol{c_6}$
p_6	c_2	c_1	$\boldsymbol{c_5}$	$\boldsymbol{c_4}$	c_6	c_3	$\boldsymbol{c_2}$	$\boldsymbol{c_1}$	c_5	c_4	$\boldsymbol{c_6}$	$\boldsymbol{c_3}$

Table 2: The 2010 NPB inter-league schedule.

Whenever a team is scheduled for a road trip consisting of multiple away sets, the team doesn't return to their home city but rather proceeds directly to their next away venue. Furthermore, we assume that every team begins the tournament at home, and returns home after its last away game. For example, in Table 2, team c_1 would travel a total distance of $D_{c_1,p_2} + D_{p_2,p_1} + D_{p_1,c_1} + D_{c_1,p_5} + D_{p_5,p_6} + D_{p_6,c_1} + D_{c_1,p_4} + D_{p_4,p_3} + D_{p_3,c_1} = (D_{p_1,p_2} + D_{p_3,p_4} + D_{p_5,p_6}) + \sum_{j=1}^{6} D_{c_1,p_j} = 1261 + 4509 = 5770$.

Let $M_p = D_{p_1,p_2} + D_{p_3,p_4} + D_{p_5,p_6}$ and $M_c = D_{c_1,c_2} + D_{c_3,c_6} + D_{c_4,c_5}$. From Table 2, the total travel distance is $M_p + \sum_{j=1}^{6} D_{c_i,p_j}$ for each c_i and $M_c + \sum_{j=1}^{6} D_{p_i,c_j}$ for each p_i. In this inter-league schedule, the six teams in each league are grouped into three pairs based on geographic proximity, so that the teams in each league have three identical road trips lasting two sets (four games).

For example, each team c_i has road trips consisting of the pairs $\{p_1, p_2\}$, $\{p_3, p_4\}$, and $\{p_5, p_6\}$ in some order. From Table 1, we can show that the Central League teams travel 27205 kilometres, and the Pacific League teams travel 23929 kilometres, for a total of 51134 kilometres.

The sums $\sum_{j=1}^{6} D_{p_i,c_j}$ and $\sum_{j=1}^{6} D_{c_i,p_j}$ are fixed for each i, regardless of how the six teams are paired up. Thus, if the teams in each league play the same set of three road trips, the total distance is minimized whenever the sum $M_p + M_c$ is minimized. Note that both M_p and M_c represent the total edge weight of a perfect matching in a complete graph on 6 vertices.

Therefore, the optimal value of M_c is the minimum value of $D_{c_{\pi(1)},c_{\pi(2)}} + D_{c_{\pi(3)},c_{\pi(4)}} + D_{c_{\pi(5)},c_{\pi(6)}}$ over all permutations π of $\{1, 2, 3, 4, 5, 6\}$. The distance-optimal schedule occurs by making each team play their three road trips based on the other league's minimum-weight perfect matching.

From Figure 1 (or Table 1), the minimum-weight perfect matching for the Central League occurs when $\pi = (1, 2, 3, 4, 5, 6)$. By replacing the Pacific League road trips with the optimal matching $\{c_1, c_2\}$, $\{c_3, c_4\}$, and $\{c_5, c_6\}$, we can reduce each team's travel by $(D_{c_3,c_6} + D_{c_4,c_5}) - (D_{c_3,c_4} + D_{c_5,c_6}) = 51$ kilometres.

This "optimal" grouping was used by the NPB in 2009, with the twelve teams traveling a total of $51134 - 51 \times 6 = 50828$ kilometres. From Figure 1, it is clear that the perfect matching used in the other league, namely $\{p_1, p_2\}$, $\{p_3, p_4\}$, and $\{p_5, p_6\}$, has minimum weight.

Hence, if we adopt the framework of Table 2, where the teams play inter-league games by alternating home and away sets two at a time, then the optimal schedule requires 50828 kilometres of total team travel. We note that for any 12×12 distance matrix, we can rapidly generate the distance-optimal schedule by finding the best possible pairing of three road trips for each league and arranging the tournament schedule in the format of Table 2. This argument easily generalizes to the scenario where there are $2n$ teams in each league, since there is an $O(n^3)$ algorithm to determine a minimum-weight perfect matching for any weighted complete graph on $2n$ vertices (Lawler 1976).

In Table 2, every time slot has the property that the teams in each league either all play at home, or all play on the road. We say that such a schedule is *uniform*. By relaxing this constraint and including *non-uniform* schedules for consideration, we expand the search space.

However, the next result shows that if we impose the restriction that no team can play more than two consecutive sets at home or on the road, the distance-optimal schedule must be uniform, and have the same structure as Table 2.

Proposition 1 *Let P and C each consist of $2n$ teams, for some $n \geq 1$. Consider an inter-league tournament between P and C, where each pair of teams p_i and c_j plays two sets, with one set at each team's home venue. If every team can play at most two consecutive sets at home or on the road, then any distance-optimal inter-league schedule must be uniform, alternating home and away sets two at a time.*

Proof Let $P = \{p_i : 1 \leq i \leq 2n\}$ be the Pacific League teams and $C = \{c_i : 1 \leq i \leq 2n\}$ be the Central League teams. Define ILB_t to be the *individual lower bound* for team $t \in P \cup C$. This value represents the minimum possible distance that can be traveled by team t in order to complete its $4n$ sets, independent of the other teams' schedules.

Then a trivial lower bound for the total travel distance is $TLB = \sum_{t \in P \cup C} ILB_t$. If team p_i does not play their $2n$ road sets in pairs, there would be at least two single-set road trips which can be combined to reduce that team's travel distance, by the Triangle Inequality. Letting M_c be the minimum weight of a perfect matching for the Central League, we have $ILB_{p_i} = M_c + \sum_{j=1}^{2n} D_{p_i,c_j}$. Analogously, $ILB_{c_i} = M_p + \sum_{j=1}^{2n} D_{c_i,p_j}$. Therefore,

$$TLB = 2n(M_p + M_c) + 2\sum_{i=1}^{2n}\sum_{j=1}^{2n} D_{p_i,c_j}.$$

By the same construction as Table 2, we can construct a uniform inter-league schedule with total distance TLB, which equals 50828 in the case of the NPB distance matrix.

To complete the proof, we must show that any schedule having total distance TLB must be uniform, and have the structure of alternating home and away sets two at a time.

If a schedule has total distance TLB, by definition, each team t must travel a total distance of ILB_t. Suppose on the contrary that there exists a non-uniform distance-optimal schedule. By the Triangle Inequality, each team t must play their $2n$ road sets in n pairs, as otherwise that team's travel distance would exceed ILB_t.

For this schedule, let $H(P, s)$ and $R(P, s)$ be respectively the number of teams in P playing at home and on the road in time slot s. Similarly, define $H(C, s)$ and $R(C, s)$. For all $1 \leq s \leq 4n$, we have $H(P, s) + R(P, s) = H(C, s) + R(C, s) = 2n$, $H(P, s) = R(C, s)$ and $R(P, s) = H(C, s)$.

For each $a, b \in [H, R]$, let p_{ab} be the number of teams in P playing their first set at a and their second set at b. For example, p_{HR} is the number of teams in P playing their first set at home and their second set on the road. By definition, $p_{HH}+p_{HR}+p_{RH}+p_{RR} = 2n$. Similarly define c_{ab} so that $c_{HH}+c_{HR}+c_{RH}+c_{RR} = 2n$. Note that $p_{RH} = c_{RH} = 0$ since every team must play their road sets in pairs.

Since $H(P, 1) = R(C, 1)$ and $H(P, 2) = R(C, 2)$, we have $p_{HH} + p_{HR} = c_{RR}$ and $p_{HH} = c_{HR} + c_{RR}$. This implies that $p_{HR} + c_{HR} = 0$, forcing $p_{HR} = c_{HR} = 0$. Thus, $p_{HH} = c_{RR}$ and $c_{HH} = p_{RR}$.

Each team counted in p_{HH} must play on the road in set 3, as otherwise some team would play three consecutive home sets. But since road sets take place in pairs, each of these teams must also play on the road in set 4. Thus, $R(P, 4) \geq p_{HH}$. Similarly, each team counted in c_{HH} must play on the road in sets 3 and 4, implying that $H(C, 4) \leq c_{RR}$. Since $R(P, 4) = H(C, 4)$ and $p_{HH} = c_{RR}$, we have $H(C, 4) = c_{RR}$. In other words, each team counted in c_{RR} plays at home in sets 3 and 4, forcing them to play on the road in sets 5 and 6. The same pattern holds for each team counted in p_{RR}. Thus, $R(P, 6) + R(C, 6) \geq p_{RR} + c_{RR}$.

Each team counted in p_{HH} and c_{HH} plays at home in set 5, as otherwise some team would play three consecutive road sets. Furthermore, each of these teams must also play at home in set 6, as otherwise $2n = R(P, 6) + R(C, 6) \geq p_{RR}+c_{RR}+1 = p_{RR}+p_{HH}+1 = 2n+1$, a contradiction.

Repeating this argument, each team counted in p_{HH} and c_{HH} must play two home sets followed by two road sets, and alternating that pattern until the end of the tournament. Conversely, each team counted in p_{RR} and c_{RR} must play the inverse alternating schedule of two road sets followed by two home sets.

Without loss, assume $p_{HH} > 0$. Suppose $c_{HH} > 0$. Then there exist two teams p_i and c_j that have the exact same home-road pattern HH-RR-HH-RR-$\cdots$-HH-RR. This is a contradiction as these two teams would then be unable to play each other. Thus, $c_{HH} = 0$, implying that $p_{RR} = 0$. We therefore have $p_{HH} = 2n$ and $c_{RR} = 2n$, proving that the tournament must be uniform, with the teams in each league alternating home and away sets two at a time. ∎

Therefore, we have solved the inter-league optimization problem for any $4n \times 4n$ distance matrix, given the constraint that every team can play at most *two* consecutive sets at home or on the road. By our analysis, there exists an $O(n^3)$ algorithm to construct an optimal tournament schedule, by pairing each team's $2n$ road sets according to the minimum-weight perfect matching. As shown in Proposition 1, any distance-optimal schedule *must* be uniform.

However, if we consider the constraint that every team can play at most *three* consecutive sets at home or on the road, then the problem of generating a distance-optimal inter-league schedule becomes extremely difficult. In fact, as we will explain in the next section, this problem becomes NP-complete, even when restricting our search space to the set of uniform schedules!

We analyze this extension to three-set home stands and road trips, since both scenarios frequently occur during NPB *intra*-league play, where teams play nine consecutive games (3 sets of 3 games) either at home or on the road. Since inter-league sets consist of two games, even if a team played three consecutive sets at home or on the road, that would only correspond to six games. This simple change from "at most 2" to "at most 3" motivates the *Bipartite Traveling Tournament Problem (BTTP)*, which we now present.

The Bipartite Traveling Tournament Problem

The challenge of creating a distance-optimal intra-league schedule motivated the *Traveling Tournament Problem* (TTP), in which every pair of teams plays two sets (i.e., a double round-robin tournament). The output is an optimal schedule that minimizes the sum total of distances traveled by the n teams as they move from city to city, subject to three constraints that ensure competitive balance:

(a) *at-most-three*: No team may have a home stand or road trip lasting more than three sets.

(b) *no-repeat*: A team cannot play against the same opponent in two consecutive sets.

(c) *each-venue*: Each pair of teams plays twice, with one set at each team's home venue.

The TTP involves both *integer programming* to prevent excessive travel, as well as *constraint programming* to create a schedule of home and road sets that meet all the feasibility requirements. While each problem is simple to solve on its own, its combination has proven to be extremely challenging (Easton, Nemhauser, and Trick 2001), even for small cases such as $n = 6$ and $n = 8$. The TTP has emerged as a popular area of study (Kendall et al. 2010) within the operations research community due to its surprising complexity.

We introduce the *Bipartite Traveling Tournament Problem (BTTP)*, the inter-league analogue of the TTP. Let $X = \{x_1, x_2, \ldots, x_n\}$ and $Y = \{y_1, y_2, \ldots, y_n\}$ be two leagues, where each pair of teams x_i and y_j (with $1 \leq i, j \leq n$) plays two sets, with one set at each venue. Given a $2n \times 2n$ distance matrix, the solution to *BTTP* is a distance-optimal double round-robin inter-league schedule satisfying the *at-most-three*, *no-repeat*, and *each-venue* constraints.

Let *BTTP** be the restriction of *BTTP* to the set of *uniform* tournament schedules. By definition, the solution to *BTTP** has total travel distance at least that of *BTTP*.

We proved that both *BTTP* and *BTTP** are NP-complete (Hoshino and Kawarabayashi 2011a) by obtaining a reduction from 3-SAT, the well-known NP-complete problem on boolean satisfiability.

To explain the difficulty of *BTTP*, we provide a simple illustration for the case $n = 3$ by providing two feasible tournaments in Table 3, with one uniform schedule and one non-uniform schedule.

	1	2	3	4	5	6
x_1	y_1	y_2	y_3	$\boldsymbol{y_1}$	$\boldsymbol{y_2}$	$\boldsymbol{y_3}$
x_2	y_2	y_3	y_1	$\boldsymbol{y_2}$	$\boldsymbol{y_3}$	$\boldsymbol{y_1}$
x_3	y_3	y_1	y_2	$\boldsymbol{y_3}$	$\boldsymbol{y_1}$	$\boldsymbol{y_2}$
y_1	$\boldsymbol{x_1}$	$\boldsymbol{x_3}$	$\boldsymbol{x_2}$	x_1	x_3	x_2
y_2	$\boldsymbol{x_2}$	$\boldsymbol{x_1}$	$\boldsymbol{x_3}$	x_2	x_1	x_3
y_3	$\boldsymbol{x_3}$	$\boldsymbol{x_2}$	$\boldsymbol{x_1}$	x_3	x_2	x_1

	1	2	3	4	5	6
x_1	y_3	y_2	$\boldsymbol{y_1}$	$\boldsymbol{y_3}$	y_1	$\boldsymbol{y_2}$
x_2	$\boldsymbol{y_1}$	y_3	$\boldsymbol{y_2}$	y_1	y_2	$\boldsymbol{y_3}$
x_3	$\boldsymbol{y_2}$	y_1	$\boldsymbol{y_3}$	y_2	y_3	$\boldsymbol{y_1}$
y_1	x_2	$\boldsymbol{x_3}$	x_1	$\boldsymbol{x_2}$	$\boldsymbol{x_1}$	x_3
y_2	x_3	$\boldsymbol{x_1}$	x_2	$\boldsymbol{x_3}$	$\boldsymbol{x_2}$	x_1
y_3	$\boldsymbol{x_1}$	$\boldsymbol{x_2}$	x_3	x_1	$\boldsymbol{x_3}$	x_2

Table 3: Two feasible inter-league tournaments for $n = 3$.

For each team, define a *trip* to be a pair of consecutive sets where that team doesn't play at the same location in time slots s and $s + 1$. In Table 3, the top schedule has 24 total trips, while the bottom schedule has 32 total trips.

Now let the teams x_1, x_3, y_1, and y_2 be located at $(0, 0)$ and let x_2 and y_3 be located at $(1, 0)$. Then the top schedule has total distance 16 and the bottom schedule has total distance 12. So minimizing trips does not correlate to minimizing total travel distance; while the former is a trivial problem, the latter is extremely difficult, even for the case $n = 3$. Of course, a brute-force enumeration of all possible tournaments is one approach, but this only works for small cases, and not for the NPB, which has $n = 6$.

The case $n = 6$ requires the optimal scheduling of $6 \times 12 = 72$ matches. The 12-team *BTTP* is comparable in difficulty to the 8-team TTP (with 56 total matches) and the 10-team TTP (with 90 total matches), both of which were solved recently on a benchmark data set using the aid of powerful computers running calculations over multiple processors (Trick 2011). Nonetheless, with the aid of two clever heuristics that we present in the next section, we can quickly solve the NP-complete problems *BTTP** and *BTTP* for the 12×12 NPB distance matrix.

Two Heuristics for *BTTP** and *BTTP*

We now present two theorems that tackle the Bipartite Traveling Tournament Problem. The first theorem is applicable for any n, and the second theorem is specific for the case $n = 6$. These results form the theoretical basis for our heuristics that solve *BTTP** and *BTTP* for the NPB. Before presenting our results, we provide several key definitions.

For each $t \in X \cup Y$, let S_t be the set of possible schedules that can be played by team t satisfying the *at-most-three* and *each-venue* constraints. Let $\pi_t \in S_t$ be a possible schedule for team t. For each π_t, we just list the opponents of the six *road* sets, and ignore the home sets, since we can determine the total distance traveled by team t just from the road sets. To give an example, below is a feasible schedule $\pi_{x_1} \in S_{x_1}$ for the case $n = 6$:

	1	2	3	4	5	6	7	8	9	10	11	12
x_1	y_1	y_6	$\boldsymbol{y}$	$\boldsymbol{y}$	y_3	y_5	y_4	$\boldsymbol{y}$	$\boldsymbol{y}$	$\boldsymbol{y}$	y_2	$\boldsymbol{y}$

In the above team schedule π_{x_1}, each $\boldsymbol{y}$ represents a home set played by x_1 against a unique opponent in Y. Note that π_{x_1} satisfies the *at-most-three* and *each-venue* constraints.

Let $\Phi = (\pi_{x_1}, \pi_{x_2}, \ldots, \pi_{x_n}, \pi_{y_1}, \pi_{y_2}, \ldots, \pi_{y_n})$, where $\pi_t \in S_t$ for each $t \in X \cup Y$. Since road sets of X correspond to home sets of Y and vice-versa, it suffices to list just the time slots and opponents of the n road sets in each π_t, since we can then uniquely determine the full schedule of $2n$ sets for every team $t \in X \cup Y$, thus producing an inter-league tournament schedule Φ. We note that Φ is a feasible solution to *BTTP* iff each team plays a unique opponent in every time slot, and no team schedule π_t violates the *no-repeat* constraint.

In the following sections, we will frequently refer to *team* schedules π_t and *tournament* schedules Φ. From the context it will be clear whether the schedule is for an individual team $t \in X \cup Y$, or for all $2n$ teams in $X \cup Y$.

As before, define ILB_t to be the individual lower bound of team t, the minimum possible distance that can be traveled by team t in order to complete its $2n$ sets.

For each $\pi_t \in S_t$, let $d(\pi_t)$ be the integer for which $d(\pi_t) + ILB_t$ equals the total distance traveled by team t when playing the schedule π_t. By definition, $d(\pi_t) \geq 0$.

For each $\Phi = (\pi_{x_1}, \ldots, \pi_{x_n}, \pi_{y_1}, \ldots, \pi_{y_n})$, define

$$d(\Phi) = \sum_{t \in X \cup Y} d(\pi_t).$$

Since $\sum ILB_t$ is fixed, the optimal solution to ***BTTP*** is the schedule Φ for which $d(\Phi)$ is minimized. This is the motivation for the function $d(\Phi)$.

For each subset $S_t^* \subseteq S_t$, define the *lower bound* function

$$B(S_t^*) = \min_{\pi_t \in S_t^*} d(\pi_t).$$

If $S_t^* = S_t$, then $B(S_t^*) = 0$ by the definition of ILB_t. For each subset S_t^*, we define $|S_t^*|$ to be its cardinality.

If n is a multiple of 3, we define for each team the set R_3^t as the subset of schedules in S_t for which the n road sets occur in $\frac{n}{3}$ blocks of three (i.e., team t takes $\frac{n}{3}$ three-set road trips). For example, in the top schedule of Table 3 (which has $n = 3$), every team t plays a schedule $\pi_t \in R_3^t$.

Finally, we define Γ to be a *global constraint* that fixes some subset of matches, and S_t^Γ to be the subset of schedules in S_t which are consistent with that global constraint. For example, if Γ is the simple constraint that forces y_2 to play against x_1 at home in time slot 3, then $S_{x_1}^\Gamma$ would only consist of the team schedules where slot 3 is a road set against y_2. If Γ is a much more complex global constraint (e.g. where the number of fixed matches is large), then each $|S_t^\Gamma|$ will be significantly less than $|S_t|$.

We present the first theorem, a powerful reduction heuristic that drastically cuts down the computation time.

Theorem 1 *Let M be a fixed positive integer. For any global constraint Γ, define for each $t \in X \cup Y$,*

$$Z_t = \left\{ \pi_t \in S_t^\Gamma : d(\pi_t) \leq M + B(S_t^\Gamma) - \sum_{u \in X \cup Y} B(S_u^\Gamma) \right\}.$$

If $\Phi = (\pi_{x_1}, \ldots, \pi_{x_n}, \pi_{y_1}, \ldots, \pi_{y_n})$ is a feasible tournament schedule consistent with Γ so that $d(\Phi) \leq M$, then for each $t \in X \cup Y$, team t's schedule π_t appears in Z_t.

Proof Consider all tournament schedules consistent with Γ. If there is no Φ with $d(\Phi) \leq M$, then there is nothing to prove. So assume some schedule Φ satisfies $d(\Phi) \leq M$. Letting $Q = \sum_{u \in X \cup Y} B(S_u^\Gamma)$, we have $M \geq d(\Phi) = \sum_{u \in X \cup Y} d(\pi_u) \geq \sum_{u \in X \cup Y} B(S_u^\Gamma)$, so that $M \geq Q$.

If $\pi_t \in Z_t$, then $Z_t \subseteq S_t^\Gamma$ implying that $d(\pi_t) \geq B(S_t^\Gamma)$.

Now suppose there exists some $v \in X \cup Y$ with $\pi_v \notin Z_v$. Since π_v is consistent with Γ, $\pi_v \in S_v^\Gamma$ and $d(\pi_v) > M + B(S_v^\Gamma) - Q \geq B(S_v^\Gamma)$. This is a contradiction, as

$$\begin{aligned} d(\Phi) &= d(\pi_v) + \sum_{u \in X \cup Y, u \neq v} d(\pi_u) \\ &> (M + B(S_v^\Gamma) - Q) + \sum_{u \in X \cup Y, u \neq v} B(S_u^\Gamma) \\ &= (M + B(S_v^\Gamma) - Q) + (Q - B(S_v^\Gamma)) \\ &= M. \end{aligned}$$

Hence, if $\Phi = (\pi_{x_1}, \ldots, \pi_{x_n}, \pi_{y_1}, \ldots, \pi_{y_n})$ is a feasible tournament schedule consistent with Γ so that $d(\Phi) \leq M$, then $\pi_t \in Z_t$ for all $t \in X \cup Y$. ∎

Theorem 1 shows how to perform some reduction *prior* to propagation, and may be applicable to other problems. To apply this theorem, we will reduce ***BTTP*** to k scenarios where in each scenario the six home sets for four of the Pacific League teams are pre-determined. Expressing these scenarios as the global constraints $\Gamma_1, \Gamma_2, \ldots, \Gamma_k$, each Γ_i fixes 24 of the 72 total matches.

For every Γ_i, we determine Z_{c_j} for the Central League teams and by setting a low threshold M, we show that each $|Z_{c_j}|$ is considerably smaller than $|S_{c_j}^\Gamma|$, thus reducing the search space to an amount that can be quickly analyzed.

From there, we run a simple six-loop that generates all 6-tuples $(\pi_{c_1}, \pi_{c_2}, \pi_{c_3}, \pi_{c_4}, \pi_{c_5}, \pi_{c_6})$ that can appear in a feasible schedule Φ with $d(\Phi) \leq M$. By Theorem 1, each $\pi_{c_j} \in Z_{c_j}$ for $1 \leq j \leq 6$. From this list of possible 6-tuples, we can quickly find the optimal schedule Φ which corresponds to the solution to ***BTTP***.

Next, we present a special result that works only for the case $n = 6$, when two teams from one league are located quite far from the other 10 teams, forcing the distance-optimal schedule Φ to have a particular structure.

Theorem 2 *Let M be a fixed positive integer, and define $S_t^* = \{\pi_t \in S_t : d(\pi_t) \leq M\}$. Suppose there exist two teams $x_i, x_j \in X = \{x_1, x_2, \ldots, x_6\}$ for which $S_{x_i}^* \subseteq R_3^{x_i}$, $S_{x_j}^* \subseteq R_3^{x_j}$, and for each team $y_k \in Y$, every schedule in $S_{y_k}^*$ has the property that y_k plays their road sets against x_i and x_j in two consecutive time slots.*

If $\Phi = (\pi_{x_1}, \ldots, \pi_{x_6}, \pi_{y_1}, \ldots, \pi_{y_6})$ is a feasible tournament schedule with $d(\Phi) \leq M$ where each $\pi_t \in S_t^*$, then the team schedules π_{x_i} and π_{x_j} both have the home-road pattern HH-RRR-HH-RRR-HH; moreover, each team's six home slots must have the following structure for some permutation (a, b, c, d, e, f) of $\{1, 2, 3, 4, 5, 6\}$:

	1	2	3	4	5	6	7	8	9	10	11	12
x_i	$\boldsymbol{y_a}$	$\boldsymbol{y_b}$	y	y	y	$\boldsymbol{y_c}$	$\boldsymbol{y_d}$	y	y	y	$\boldsymbol{y_e}$	$\boldsymbol{y_f}$
x_j	$\boldsymbol{y_b}$	$\boldsymbol{y_a}$	y	y	y	$\boldsymbol{y_d}$	$\boldsymbol{y_c}$	y	y	y	$\boldsymbol{y_f}$	$\boldsymbol{y_e}$

Proof We first note that if π_{x_i} and π_{x_j} have the above structure, they satisfy all the given conditions since $\pi_{x_i} \in R_3^{x_i}$, $\pi_{x_j} \in R_3^{x_j}$, and every team $y_k \in Y$ plays road sets against x_i and x_j in two consecutive time slots. For example, y_d plays road sets against x_j in slot 6 and against x_i in slot 7. We now prove that π_{x_i} and π_{x_j} must have this structure.

For each team $x_t \in X$ and time slot $s \in [1, 12]$, define $O(x_t, s)$ to be the *opponent* of team x_t in set s. We define $O(x_t, s)$ only when x_t is playing at *home*; for the sets when x_t plays on the road, $O(x_t, s)$ is undefined.

Since $\pi_{x_i} \in S_{x_i}^*$ and $S_{x_i}^* \subseteq R_3^{x_i}$, there are four possible cases to consider:

(1) x_i plays set 1 at home, and sets 2 to 4 on the road.
(2) x_i plays sets 1 and 2 at home, and sets 3 to 5 on the road.
(3) x_i plays sets 1 to 3 at home, and sets 4 to 6 on the road.
(4) x_i plays sets 1 to 3 on the road, and set 4 at home.

We examine the cases one by one. In each, suppose there exists a feasible schedule Φ satisfying all the given conditions. We finish with case (2).

In (1), let $O(x_i, 1) = y_a$. Then $O(x_j, 2) = y_a$, since y_a must play road sets against x_i and x_j in consecutive time slots. Since $\pi_{x_j} \in R_3^{x_j}$ and x_j plays at home in set 2, x_j must also play at home in set 1. Thus, $O(x_j, 1) = y_b$ for some y_b, which forces $O(x_i, 2) = y_b$. This is a contradiction as x_i plays set 2 on the road.

In (3), let $O(x_i, 1) = y_a$, $O(x_i, 2) = y_b$, and $O(x_i, 3) = y_c$. Then $O(x_j, 2) = y_a$ and $O(x_j, 4) = y_c$. Either $O(x_j, 1) = y_b$ or $O(x_j, 3) = y_b$. In either case, we violate the *at-most-three* constraint or the condition that $\pi_{x_j} \in R_3^{x_j}$.

In (4), team x_i starts with a three-set road trip. In order to satisfy the *at-most-three* constraint, π_{x_i} must have the pattern RRR-HHH-RRR-HHH. Then this reduces to case (3), as we can read the schedule Φ backwards, letting $O(x_i, 12) = y_a$, $O(x_i, 11) = y_b$, $O(x_i, 10) = y_c$, and applying the argument in the previous paragraph.

In (2), let $O(x_i, 1) = y_a$ and $O(x_i, 2) = y_b$. Then $O(x_j, 2) = y_a$ and $O(x_j, 1) = y_b$. If $O(x_j, 3) = y_c$ for some y_c, then $O(x_i, 4) = y_c$, forcing x_i to play a single road set in slot 3. Thus, x_j must play on the road in set 3, and therefore also in sets 4 and 5. Hence, both x_i and x_j start with two home sets followed by three road sets. Since this is the only case remaining, by symmetry x_i and x_j must end with two home sets preceded by three road sets. Thus, these two teams must have the pattern HH-RRR-HH-RRR-HH.

In order for each y_k to play their road sets against x_i and x_j in two consecutive time slots, we must have $O(x_i, 6) = O(x_j, 7)$, $O(x_i, 7) = O(x_j, 6)$, $O(x_i, 11) = O(x_j, 12)$, and $O(x_i, 12) = O(x_j, 11)$. This completes the proof. ∎

We will use Theorem 2 to solve *BTTP*, since teams p_5 and p_6 are located quite far from the other ten teams (see Figure 1). This heuristic of isolating two teams and finding its common structure significantly reduces the search space and enables us to solve *BTTP* for the 12-team NPB in hours rather than weeks.

By applying these results, we do not require weeks of computation time on multiple processors. With these two heuristics, *BTTP** can be solved in *two minutes* and *BTTP* can be solved in less than *ten hours*, each on a single laptop. All of the code was written in Maple and compiled using Maplesoft 13 using a single Toshiba laptop under Windows with a single 2.10 GHz processor and 2.75 GB RAM. In the next two sections, we present the optimal uniform and non-uniform schedules for the NPB and justify their optimality.

Solution to *BTTP** for the NPB

We first compute ILB_t for each team $t \in P \cup C$, based on the distance matrix given in Table 1. We find that for each team t, a schedule $\pi_t \in S_t$ satisfies $d(\pi_t) = ILB_t$ only if $\pi_t \in R_3^t$, i.e., π_t plays its six road sets in two blocks of three.

We determine that $\sum ILB_t = \sum ILB_{p_i} + \sum ILB_{c_i} = 16686 + 26077 = 42763$. For any feasible inter-league tournament schedule Φ, the total distance traveled by the teams is $d(\Phi) + \sum ILB_t = d(\Phi) + 42763$.

Table 4 presents a uniform inter-league schedule Φ^* that is a feasible solution of *BTTP** with $d(\Phi^*) = (0 + 0 + 1 + 11 + 0 + 0) + (0 + 1 + 153 + 339 + 11 + 6) = 522$.

	1	2	3	4	5	6	7	8	9	10	11	12
p_1	c_2	c_3	c_1	$\boldsymbol{c_2}$	$\boldsymbol{c_3}$	$\boldsymbol{c_6}$	c_4	c_5	c_6	$\boldsymbol{c_1}$	$\boldsymbol{c_4}$	$\boldsymbol{c_5}$
p_2	c_4	c_6	c_5	$\boldsymbol{c_3}$	$\boldsymbol{c_6}$	$\boldsymbol{c_1}$	c_2	c_3	c_1	$\boldsymbol{c_2}$	$\boldsymbol{c_5}$	$\boldsymbol{c_4}$
p_3	c_3	c_1	c_2	$\boldsymbol{c_6}$	$\boldsymbol{c_4}$	$\boldsymbol{c_2}$	c_5	c_6	c_4	$\boldsymbol{c_5}$	$\boldsymbol{c_3}$	$\boldsymbol{c_1}$
p_4	c_5	c_4	c_6	$\boldsymbol{c_5}$	$\boldsymbol{c_2}$	$\boldsymbol{c_4}$	c_3	c_1	c_2	$\boldsymbol{c_6}$	$\boldsymbol{c_1}$	$\boldsymbol{c_3}$
p_5	c_1	c_2	c_3	$\boldsymbol{c_4}$	$\boldsymbol{c_1}$	$\boldsymbol{c_5}$	c_6	c_4	c_5	$\boldsymbol{c_3}$	$\boldsymbol{c_2}$	$\boldsymbol{c_6}$
p_6	c_6	c_5	c_4	$\boldsymbol{c_1}$	$\boldsymbol{c_5}$	$\boldsymbol{c_3}$	c_1	c_2	c_3	$\boldsymbol{c_4}$	$\boldsymbol{c_6}$	$\boldsymbol{c_2}$
c_1	$\boldsymbol{p_5}$	$\boldsymbol{p_3}$	$\boldsymbol{p_1}$	p_6	p_5	p_2	$\boldsymbol{p_6}$	$\boldsymbol{p_4}$	$\boldsymbol{p_2}$	p_1	p_4	p_3
c_2	$\boldsymbol{p_1}$	$\boldsymbol{p_5}$	$\boldsymbol{p_3}$	p_1	p_4	p_3	$\boldsymbol{p_2}$	$\boldsymbol{p_6}$	$\boldsymbol{p_4}$	p_2	p_5	p_6
c_3	$\boldsymbol{p_3}$	$\boldsymbol{p_1}$	$\boldsymbol{p_5}$	p_2	p_1	p_6	$\boldsymbol{p_4}$	$\boldsymbol{p_2}$	$\boldsymbol{p_6}$	p_5	p_3	p_4
c_4	$\boldsymbol{p_2}$	$\boldsymbol{p_4}$	$\boldsymbol{p_6}$	p_5	p_3	p_4	$\boldsymbol{p_1}$	$\boldsymbol{p_5}$	$\boldsymbol{p_3}$	p_6	p_1	p_2
c_5	$\boldsymbol{p_4}$	$\boldsymbol{p_6}$	$\boldsymbol{p_2}$	p_4	p_6	p_5	$\boldsymbol{p_3}$	$\boldsymbol{p_1}$	$\boldsymbol{p_5}$	p_3	p_2	p_1
c_6	$\boldsymbol{p_6}$	$\boldsymbol{p_2}$	$\boldsymbol{p_4}$	p_3	p_2	p_1	$\boldsymbol{p_5}$	$\boldsymbol{p_3}$	$\boldsymbol{p_1}$	p_4	p_6	p_5

Table 4: Solution to *BTTP** with total distance 43285 km.

We claim that Φ^* is an optimal solution, with total distance $d(\Phi^*) + \sum ILB_t = 522 + 42763 = 43285$. To establish this claim, we show that $d(\Phi) \geq 522$ for any uniform schedule $\Phi = (\pi_{p_1}, \pi_{p_2}, \ldots, \pi_{p_6}, \pi_{c_1}, \pi_{c_2}, \ldots, \pi_{c_6})$.

We exploit the uniformity of Φ and split *BTTP** into two separate optimization problems: first, we show that in any tournament schedule Φ, the six-tuple $(\pi_{p_1}, \pi_{p_2}, \ldots, \pi_{p_6})$ must satisfy $\sum d(\pi_{p_i}) \geq 12$. Then we show that for any Φ, the six-tuple $(\pi_{c_1}, \pi_{c_2}, \ldots, \pi_{c_6})$ must satisfy $\sum d(\pi_{c_i}) \geq 510$. This will prove that $d(\Phi) \geq 12 + 510 = 522$.

First we explain why an optimal uniform schedule Φ must have the home-road pattern HHH-RRR-HHH-RRR for one league, and RRR-HHH-RRR-HHH for the other league. Note that no league can play its six road sets in a way that a single road set is sandwiched between two home sets (e.g. HH-RR-HH-R-HH-RRR), since the time slots can be reordered to reduce the total travel distance by the Triangle Inequality. Thus, each league's road sets must take place in two blocks of three (as in Table 4) or three blocks of two (as in Table 2).

In Section 2, we showed that if the six road sets occur in three blocks of two, the minimum total travel distance is $23929 > \sum ILB_{p_i} + 522$ for the Pacific League and $26899 > \sum ILB_{c_i} + 522$ for the Central League, for a total of 50828 kilometres. Thus, if Φ is distance-optimal, then neither league can have its six road sets in three blocks of two, as otherwise $d(\Phi) > 522$. Hence, $\pi_t \in R_3^t$ for each team $t \in P \cup C$. Without loss, assume the Central League teams play at home in set 1.

Let Γ be the global constraint that every Pacific League team plays its road sets in time slots 1, 2, 3, 7, 8, and 9. Then for each t, S_t^Γ is the set of team schedules π_t consistent with Γ. Recall that each π_t just lists the time slots and opponents for the six road sets. Since π_t must have a fixed home-road pattern (either HHH-RRR-HHH-RRR or RRR-HHH-RRR-HHH), there are only 6! possible options for π_t. Hence, $|S_t^\Gamma| = 720$ for each $t \in P \cup C$.

Since each team's ILB_t is attained by some schedule $\pi_t \in R_3^t$, we have $B(S_t^\Gamma) = 0$. By Theorem 1, if Φ is a feasible schedule with $\sum d(\pi_{p_i}) \leq 12$, then $\pi_{p_i} \in Z_{p_i}$ where $Z_{p_i} = \{\pi_{p_i} \in S_{p_i}^\Gamma : d(\pi_{p_i}) \leq 12\}$.

By this reduction heuristic, we determine that $(|Z_{p_1}|, |Z_{p_2}|, \ldots, |Z_{p_6}|) = (32, 32, 32, 48, 24, 16)$; for example, only 32 of the 720 schedules in $S_{p_1}^\Gamma$ satisfy $d(\pi_{p_1}) \leq 12$. This reduces the search space considerably.

From there, we apply a simple six-loop procedure, where on the k^{th} step, we examine all possible schedules $\pi_{p_k} \in Z_{p_k}$ for team p_k and compare them to the set of feasible schedules $\{\pi_{p_1}, \pi_{p_2}, \ldots, \pi_{p_{k-1}}\}$, adding π_{p_k} to each feasible set if no two teams play against the same opponent c_j in the same time slot. Since the cardinality of each set Z_{p_i} is so small, Maplesoft can rapidly perform the six-loop computation (in just 7.8 seconds), generating 128 possible 6-tuples $(\pi_{p_1}, \pi_{p_2}, \ldots, \pi_{p_6})$ that can appear in Φ with $\sum d(\pi_{p_i}) = 12$. Furthermore, this computation shows that there is no 6-tuple with $\sum d(\pi_{p_i}) < 12$.

Thus, $\Phi = (\pi_{p_1}, \pi_{p_2}, \ldots, \pi_{p_n}, \pi_{c_1}, \pi_{c_2}, \ldots, \pi_{c_n})$ must satisfy $\sum d(\pi_{p_i}) \geq 12$. We now show that $\sum d(\pi_{c_i}) \geq 510$.

To establish this much-harder bound for the Central League teams, we can repeat the same process as above by setting the bound $M = 510$ to determine that $(|Z_{c_1}|, |Z_{c_2}|, \ldots, |Z_{c_6}|) = (160, 224, 272, 144, 152, 152)$. But a six-loop computation here would take far too long; instead, we provide a fast algorithm inspired by Theorem 2.

For each Central League team c_j, let $T_{c_j} \subseteq S^{\Gamma}_{c_j}$ be the subset of schedules for which c_j does *not* play road sets against p_5 and p_6 in two consecutive time slots. We find that $B(T_{c_1}) = 365$, i.e., any schedule $\pi_{c_1} \in T_{c_1}$ satisfies $d(\pi_{c_1}) \geq 365$. Similarly we have $B(T_{c_2}) = 314$, $B(T_{c_3}) = 153$, $B(T_{c_4}) = 313$, $B(T_{c_5}) = 324$, and $B(T_{c_6}) = 319$.

If each c_j has the pattern HHH-RRR-HHH-RRR, it is easy to see that (at least) two Central League teams cannot play road sets against p_5 and p_6 in two consecutive time slots. For example, if c_1 and c_2 are two such teams, then $\sum d(\pi_{c_i}) \geq d(c_1) + d(c_2) \geq B(T_{c_1}) + B(T_{c_2}) = 365 + 314 > 510$. Thus, if there exists Φ for which $\sum d(\pi_{c_i}) \leq 510$, then there can only be two such teams, and one of them must be c_3.

We split the analysis into five cases. For each $j \in \{1, 2, 4, 5, 6\}$, Φ is generated from selecting π_{c_3} and π_{c_j} from T_{c_3} and T_{c_j} respectively. For the other four teams in the Central League, we let $Z_{c_i} = \{\pi_{c_i} \in S^{\Gamma}_{c_i} : d(\pi_{c_i}) \leq 510 - B(T_{c_3}) - B(T_{c_j}) \leq 44\}$. We then run our six-loop, computing all possible 6-tuples $(\pi_{c_1}, \pi_{c_2}, \ldots, \pi_{c_6})$ satisfying the given conditions.

If $j = 4$, then $(|Z_{c_1}|, |Z_{c_2}|, |T_{c_3}|, |T_{c_4}|, |Z_{c_5}|, |Z_{c_6}|) = (16, 48, 16, 32, 64, 64)$, We find that in this case, there are 512 different 6-tuples with $\sum d(\pi_{c_i}) = 510$ and none with $\sum d(\pi_{c_i}) < 510$. As for the other values of j, there is no feasible 6-tuple with $\sum d(\pi_{c_i}) \leq 510$. Maplesoft is exceptionally fast; all five cases run in a total of 119 seconds.

Therefore, we have shown that in any uniform schedule $\Phi = (\pi_{p_1}, \pi_{p_2}, \ldots, \pi_{p_6}, \pi_{c_1}, \pi_{c_2}, \ldots, \pi_{c_6})$, we must have $\sum d(\pi_{p_i}) \geq 12$ and $\sum d(\pi_{c_i}) \geq 510$. Since Table 4 is a uniform inter-league schedule with $d(\Phi) = 12 + 510 = 522$, our proof is complete; our optimal solution to *BTTP** reduces the total travel distance by 7849 kilometres, or 15.3%, compared to the 2010 NPB schedule.

Solution to *BTTP* for the NPB

Table 5 presents an inter-league tournament schedule Φ that is a solution to *BTTP* with $d(\Phi) = (0+4+0+0+1+1) + (51+9+31+58+19+13) = 187$.

	1	2	3	4	5	6	7	8	9	10	11	12
p_1	$\boldsymbol{c_3}$	$\boldsymbol{c_5}$	$\boldsymbol{c_1}$	c_3	c_2	c_1	$\boldsymbol{c_6}$	$\boldsymbol{c_2}$	c_4	c_5	c_6	$\boldsymbol{c_4}$
p_2	$\boldsymbol{c_5}$	$\boldsymbol{c_3}$	c_2	c_1	c_3	$\boldsymbol{c_6}$	$\boldsymbol{c_1}$	c_4	c_5	c_6	$\boldsymbol{c_4}$	$\boldsymbol{c_2}$
p_3	$\boldsymbol{c_4}$	$\boldsymbol{c_2}$	c_6	c_5	c_4	$\boldsymbol{c_3}$	$\boldsymbol{c_5}$	$\boldsymbol{c_1}$	c_3	c_2	c_1	$\boldsymbol{c_6}$
p_4	$\boldsymbol{c_2}$	$\boldsymbol{c_4}$	$\boldsymbol{c_5}$	c_4	c_6	c_5	$\boldsymbol{c_3}$	$\boldsymbol{c_6}$	$\boldsymbol{c_1}$	c_3	c_2	c_1
p_5	$\boldsymbol{c_1}$	$\boldsymbol{c_6}$	c_4	c_6	c_5	$\boldsymbol{c_2}$	$\boldsymbol{c_4}$	c_3	c_2	c_1	$\boldsymbol{c_5}$	$\boldsymbol{c_3}$
p_6	$\boldsymbol{c_6}$	$\boldsymbol{c_1}$	c_3	c_2	c_1	$\boldsymbol{c_4}$	$\boldsymbol{c_2}$	c_5	c_6	c_4	$\boldsymbol{c_3}$	$\boldsymbol{c_5}$
c_1	p_5	p_6	p_1	$\boldsymbol{p_2}$	$\boldsymbol{p_6}$	$\boldsymbol{p_1}$	p_2	p_3	p_4	$\boldsymbol{p_5}$	$\boldsymbol{p_3}$	$\boldsymbol{p_4}$
c_2	p_4	p_3	$\boldsymbol{p_2}$	$\boldsymbol{p_6}$	$\boldsymbol{p_1}$	p_5	p_6	p_1	$\boldsymbol{p_5}$	$\boldsymbol{p_3}$	$\boldsymbol{p_4}$	p_2
c_3	p_1	p_2	$\boldsymbol{p_6}$	$\boldsymbol{p_1}$	$\boldsymbol{p_2}$	p_3	p_4	$\boldsymbol{p_5}$	$\boldsymbol{p_3}$	$\boldsymbol{p_4}$	p_6	p_5
c_4	p_3	p_4	$\boldsymbol{p_5}$	$\boldsymbol{p_4}$	$\boldsymbol{p_3}$	p_6	p_5	$\boldsymbol{p_2}$	$\boldsymbol{p_1}$	$\boldsymbol{p_6}$	p_2	p_1
c_5	p_2	p_1	p_4	$\boldsymbol{p_3}$	$\boldsymbol{p_5}$	$\boldsymbol{p_4}$	p_3	$\boldsymbol{p_6}$	$\boldsymbol{p_2}$	$\boldsymbol{p_1}$	p_5	p_6
c_6	p_6	p_5	$\boldsymbol{p_3}$	$\boldsymbol{p_5}$	$\boldsymbol{p_4}$	p_2	p_1	p_4	$\boldsymbol{p_6}$	$\boldsymbol{p_2}$	$\boldsymbol{p_1}$	p_3

Table 5: Solution to *BTTP* with total distance 42950 km.

In Table 5, we see that only seven of the twelve teams satisfy $\pi_t \in R^t_3$, namely c_1 and all six of the Pacific League teams. However, unlike Table 4, every Central League team in this schedule plays road sets against p_5 and p_6 in consecutive time slots. This explains why each $d(c_j)$ in Φ is small.

We claim that Φ is an optimal solution, with total distance $d(\Phi) + \sum ILB_t = 187 + 42763 = 42950$. To prove this, we set $M = 187$. Define $S^*_t = \{\pi_t \in S_t : d(\pi_t) \leq M\}$, from which we determine that $S^*_{p_5} \subseteq R^{p_5}_3$ and $S^*_{p_6} \subseteq R^{p_6}_3$.

In the previous section, we defined $T_{c_i} \subseteq S^{\Gamma}_{c_i}$ to be the subset of schedules for which c_i does *not* play their road sets against p_5 and p_6 in two consecutive time slots. From this, we showed that $B(T_{c_3}) = 153$, and that $B(T_{c_j}) > M = 187$ for $j \in \{1, 2, 4, 5, 6\}$. We claim that if Φ satisfies $d(\Phi) \leq 187$, then $\pi_{c_j} \notin T_{c_j}$ for all $1 \leq j \leq 6$.

It suffices to prove the claim for $j = 3$. There are 144 schedules in T_{c_3}, all of which belong to the set $R^{c_3}_3$. For example, one such schedule π_{c_3} is

	1	2	3	4	5	6	7	8	9	10	11	12
c_3	$\boldsymbol{p}$	p_2	p_1	p_6	$\boldsymbol{p}$	$\boldsymbol{p}$	$\boldsymbol{p}$	p_3	p_4	p_5	$\boldsymbol{p}$	$\boldsymbol{p}$

Suppose there exists a tournament schedule Φ with $d(\Phi) \leq 187$ and $\pi_{c_3} \in T_{c_3}$. There are nine possible home-road patterns for $\pi_{p_5} \in R^{p_5}_3$ (e.g. HHH-RRR-H-RRR-HH and H-RRR-HHH-RRR-HH), each of which gives rise to $6! = 720$ possible orderings for the six home sets. Thus, there are $9 \times 720 = 6480$ ways we can select the time slots and opponents for the six home sets in π_{p_5}. Similarly, there are 6480 ways to do this for π_{p_6}. A simple Maplesoft procedure shows that only 140 of the 6480^2 possible pairs (π_{p_5}, π_{p_6}) are consistent with at least one $\pi_{c_3} \in T_{c_3}$.

For each of these 140 cases, define the global constraints $\Gamma_1, \Gamma_2, \ldots, \Gamma_{140}$, obtained from fixing the twelve home sets in $\{\pi_{p_5}, \pi_{p_6}\}$. For each Γ_k, define for each $j \in \{1, 2, 4, 5, 6\}$ the set $Z_{c_j} = \{\pi_{c_j} \in S^{\Gamma_k}_{c_j} : d(\pi_{c_j}) \leq M - B(T_{c_3}) = 34\}$. Then we run our six-loop to compute all possible 6-tuples $(\pi_{c_1}, \pi_{c_2}, \ldots, \pi_{c_6})$ satisfying the given conditions with $\pi_{c_3} \in T_{c_3}$ and $\pi_{c_j} \in Z_{c_j}$ for $j \neq 3$. Within twenty minutes, Maplesoft solves all 140 cases and returns no feasible 6-tuples that can appear in a schedule Φ with $d(\Phi) \leq 187$.

Therefore, in Φ, each c_j must play road sets against p_5 and p_6 in consecutive time slots. Thus, teams p_5 and p_6 satisfy the conditions of Theorem 2. Hence, the home-road pattern of π_{p_5} and π_{p_6} in Φ must be HH-RRR-HH-RRR-HH.

Without loss, assume that p_5 plays a home set against c_1 within the first *six* time slots; otherwise we can read the schedule backwards by symmetry. Thus, there are $\frac{6!}{2} = 360$ ways to assign opponents to the six home sets in π_{p_5}. By Theorem 2, each of these 360 arrangements uniquely determines the six home sets in π_{p_6}.

A short calculation shows that in order for $d(\Phi) \leq M = 187$, teams p_1 and p_3 must also play their six road sets in two blocks of three. In other words, $\pi_{p_1} \in R_3^{p_1}$ and $\pi_{p_3} \in R_3^{p_3}$. As mentioned earlier, there are $9 \times 6!$ possible ways to select the six home sets for each of π_{p_1} and π_{p_3}.

Thus, there are $360 \times (9 \cdot 6!) \times (9 \cdot 6!)$ ways we can select the 24 home sets played by the teams in $\{p_1, p_3, p_5, p_6\}$. We eliminate all scenarios in which some p_i and p_j play against some c_k in the same time slot. For the possibilities that remain, we create a global constraint to apply Theorem 1.

Let $\{\Gamma_1, \Gamma_2, \ldots, \Gamma_k\}$ be the complete set of global constraints derived from the above process, where each Γ_i fixes 24 of the 72 matches, corresponding to the home sets of $\{p_1, p_3, p_5, p_6\}$. The reduction heuristic of Theorem 1 allows us to quickly verify the existence of feasible tournament schedules Φ consistent with Γ_i for which $d(\Phi) \leq M$.

To explain this procedure, let us illustrate with the inter-league schedule in Table 5. Let Γ be the constraint that fixes the 24 home sets of teams p_1, p_3, p_5, and p_6 in Table 5. Then $S_{c_5}^{\Gamma}$, defined as the subset of schedules in S_{c_5} consistent with Γ, consists only of team schedules π_{c_5} for which c_5 plays road sets against p_1 in slot 2, p_3 in slot 7, p_5 in slot 11, and p_6 in slot 12.

We find that there are only 11 schedules $\pi_{c_5} \in S_{c_5}^{\Gamma}$ with $d(\pi_{c_5}) \leq M$ that are consistent with Γ. Furthermore, each $d(\pi_{c_5}) \in \{19, 41, 46, 48\}$, implying that $B(S_{c_5}^{\Gamma}) = 19$. Similarly, we can calculate the other values of $B(S_{c_j}^{\Gamma})$.

We find that $\sum_{j=1}^{6} B(S_{p_j}^{\Gamma}) = 0$ and $\sum_{j=1}^{6} B(S_{c_j}^{\Gamma}) = 51 + 9 + 31 + 58 + 19 + 13 = 181$, implying that $Z_{c_5} = \{\pi_{c_5} \in S_{c_5}^{\Gamma} : d(\pi_{c_5}) \leq 187 + 19 - 181 = 25\}$. Hence, Z_{c_5} reduces to just the *two* schedules with $d(\pi_{c_5}) = 19$, including the team schedule π_{c_5} in Table 5.

By Theorem 1, any schedule Φ consistent with Γ satisfying $d(\Phi) \leq M$ must have the property that $\pi_t \in Z_t$ for each team t. Since each $|Z_{c_j}|$ is small, the calculation is extremely fast. Of course, if any $|Z_{c_j}| = 0$, then no schedule Φ can exist.

This algorithm, based on Theorems 1 and 2, runs in 34716 seconds in Maplesoft (just under 10 hours). Maplesoft generates zero inter-league schedules with $d(\Phi) < 187$ and 14 inter-league schedules with $d(\Phi) = 187$, including the schedule given in Table 5. Since we made the assumption that p_5 plays a home set against c_1 within the first six time slots, there are actually twice as many distance-optimal schedules by reading each schedule Φ backwards.

In each of the 28 distance-optimal schedules Φ, we find that $(d(\pi_{p_1}), d(\pi_{p_2}), \ldots, d(\pi_{p_6})) = (0, 4, 0, 0, 1, 1)$ and $(d(\pi_{c_1}), d(\pi_{c_2}), \ldots, d(\pi_{c_6})) = (51, 9, 31, 58, 19, 13)$.

Therefore, we have proven that Table 5 is an optimal inter-league schedule for the NPB, reducing the total travel distance by 8184 kilometres, or 16.0%, compared to the 2010 NPB schedule.

Conclusion

In this paper, we introduced the Bipartite Traveling Tournament Problem and applied it to the Nippon Professional Baseball (NPB) league, illustrating the richness and complexity of bipartite tournament scheduling. There may be other sports leagues for which *BTTP* is applicable. We can also expand our analysis to model *tripartite* and *multipartite* tournament scheduling, where a league is divided into three or more conferences. A specific example of this is the newly-created Super 15 Rugby League, consisting of five teams from South Africa, Australia, and New Zealand. In addition to intra-country games, each team plays four games (two home and two away) against teams from each of the other two countries. It would be interesting to see whether we can determine the distance-optimal tripartite tournament schedule for this rugby league using our two heuristics.

While the solution to the uniform *BTTP** was relatively simple, our solution to the non-uniform *BTTP* required 10 hours of computations. Furthermore, we were only able to solve *BTTP* by applying Theorem 2, whose requirements would not hold for a randomly-selected 12×12 distance matrix. As a result, we require a more sophisticated technique that improves upon our two heuristics, perhaps using methods in constraint programming and integer programming, such as a hybrid CP/IP. We wonder if there exists a general algorithm that would solve *BTTP* given any distance matrix, for "small" values of n such as $n = 6$, $n = 7$, and $n = 8$. We leave this as a challenge for the interested reader.

Acknowledgements

This research has been partially supported by the Japan Society for the Promotion of Science (Grant-in-Aid for Scientific Research), the C & C Foundation, the Kayamori Foundation, and the Inoue Research Award for Young Scientists.

References

Easton, K.; Nemhauser, G.; and Trick, M. 2001. The traveling tournament problem: description and benchmarks. *Proceedings of the 7th International Conference on Principles and Practice of Constraint Programming* 580–584.

Hoshino, R., and Kawarabayashi, K. 2011a. The inter-league extension of the traveling tournament problem and its application to sports scheduling. *Proceedings of the 25th AAAI Conference on Artificial Intelligence* (to appear).

Hoshino, R., and Kawarabayashi, K. 2011b. The multi-round balanced traveling tournament problem. *Proceedings of the 21st International Conference on Automated Planning and Scheduling* (to appear).

Kendall, G.; Knust, S.; Ribeiro, C.; and Urrutia, S. 2010. Scheduling in sports: An annotated bibliography. *Computers and Operations Research* 37:1–19.

Lawler, E. 1976. *Combinatorial Optimization: Networks and Matroids*. New York: Holt, Rinehart, and Winston.

Trick, M. 2011. Challenge traveling tournament problems. [Online; accessed 22-April-2011].

Wikipedia. 2011. List of sports attendance figures. [Online; accessed 22-April-2011].