

LP Probing for Piecewise Linear Optimization in Scheduling

Farid Ajili Hani El Sakkout

IC-Parc, Imperial College. London SW7 2AZ, United Kingdom.

E-mail: {fa11,hhe}@icparc.ic.ac.uk

October 15, 2001

Abstract. A scheduling problem with piecewise linear (PL) optimization extends conventional scheduling by imposing a conjunction of combinatorial PL constraints involving the objective function variables. The problem is decomposed into two constraint sub-problems 1) resource feasibility constraints and 2) temporal and PL constraints. This decomposition is fully exploited to apply a *probe backtrack* algorithm which hybridizes constraint reasoning and Linear Programming (LP) based probing. The paper investigates and compares different probing techniques and shows how the PL constraints can be used to tighten dynamically the sub-problem delegated to the probing algorithm.

Keywords: Linear programming, Constraint programming, Scheduling, Hybrid algorithms.

1. Introduction

Many scheduling problems involve minimizing non-convex piecewise linear (PL) objective functions rather than tardiness or makespan for instance. In these problems, assigning resources and start times to each activity can result in a cost being incurred. For example, this happens when the cost of assigning a machine to produce a different type of product is doubled at weekdays, and tripled at weekends. In many cases the corresponding objective functions are in the PL class. This class captures various optimization problems (*e.g.* fixed charge and curve fitting problems) and is at the heart of goal programming (Ignizio 76, 1976; Schniederjans, 1995), portfolio construction (Bertsimas, Darnell and Soucy, 1999) and data fitting. In fact, arbitrary objective functions are often inferred from empirical data, and their piecewise linearizations provide, in general, acceptable approximations (*e.g.* non-linear networks (Ho, 1980)).

The general problem of scheduling in the presence of PL optimization inherits naturally the NP-hardness of conventional scheduling. Furthermore, extra complexity is incurred because of the loss of the objective function convexity, which guarantees that any local optimum is necessarily global. A *convex* PL optimization problem can be efficiently solved by Simplex (Fourer, 1992). Pure Simplex is not sufficient to solve the non-convex case: when it repeatedly navigates between vertexes we



© 2001 Kluwer Academic Publishers. Printed in the Netherlands.

must ensure that next vertex lies on one of the linear fragments of the function.

This work was motivated by a commercial scheduling application, where activities must be scheduled on a set of resources. The cost of the schedule is measured by a PL objective function which is the sum of the costs of the activities. The cost of each activity in the schedule depends on its start time and this dependency is captured by a PL function. The start times are all *discrete* and are constrained by a given set of temporal constraints. The objective is to generate a schedule that is resource feasible, satisfies the given temporal constraints, and minimizes cost by timing the activities appropriately.

The benefits of close hybridization of Constraint Programming (CP) and Linear Programming (LP) are increasingly recognized. To solve the scheduling problem with PL optimization, we build on two existing hybridization approaches: *probe backtrack search* (El Sakkout and Wallace, 2000) and *tight cooperation* (Refalo, 1999).

The probe backtrack search algorithm presented in (El Sakkout and Wallace, 2000) enhances traditional CP backtrack search with an LP-based “prober” that is used to suggest tentative values (called “probes”). In this paper, the core goal is to extend probe backtrack search for scheduling with PL optimization. The result is a repair-based backtrack search algorithm that interleaves CP search steps with a prober solving a dynamically changed LP sub-problem.

The tight cooperation scheme in (Refalo, 1999) suggests that the dynamic “information-passing” from the CP solver to the LP solver should not be restricted to only variable bounds. New variable bounds deduced by the CP solver can also be used to dynamically tighten the LP relaxation of individual combinatorial constraints. Refalo’s work demonstrated that this is useful in the context of PL optimization (Refalo, 1999) .

The paper is organized as follows. In Section 2, we briefly define the problem and formulate it as a constraint satisfaction & optimization problem containing resource, temporal and PL constraints. Section 3 highlights the chief features of the probe backtrack algorithm. Section 4 builds two probe generators which are based on mixed integer programming (MIP). In Section 5, we present a prober reasoning on a linear relaxation of the PL constraints. Section 6 is dedicated to a performance analysis. Finally, Section 7 concludes the paper.

2. Scheduling Problems with PL Optimization

A scheduling problem with PL optimization is given by a conventional scheduling problem and a system of PL constraints that restrict the objective to be minimized. We represent the scheduling problem by the *Kernel Resource Feasibility Problem* (KRFP), which provides a generic representation of varied scheduling problems including the job shop, ship loading (Aggoun and Beldiceanu, 1992) and bridge building (Bartusch, Möhring and Radermacher, 1988; El Sakkout and Wallace, 2000). More details about the KRFP and its instances are available in (El Sakkout and Wallace, 2000). The KRFP can be formulated as a constraint satisfaction problem. Solving the KRFP is finding an assignment of discrete values to the start and end times of the activities such that the temporal constraints are satisfied and that the resources are not over-allocated. The resource constraints state that at each time point of the schedule horizon, the activities do not exceed the available resources. The model of the resource constraints we use in this paper is the one presented in (El Sakkout and Wallace, 2000). We denote by TC the set of temporal constraints of KRFP. The set TC consists of constraints having the following forms only:

- $u R c$ (bounding constraints)
- $u R v \pm c$ (distance constraints)
 where $R \in \{ =, <, >, \leq, \geq \}$, $c \in \mathbb{N}$ where u, v represent start/end times of activities.

This tractable class of temporal constraints can capture many of the complex requirements that exist in scheduling problems. It corresponds to a *Simple Temporal Problem* (Dechter, Meiri and Pear, 1991).

A unary function ϕ is PL if its domain can be partitioned into intervals on which ϕ is linear. The boundaries of the intervals, where the slope changes, are its *breakpoints*. The linear *fragment* of each interval can be uniquely and simply characterized by its two bounding breakpoints. Different representations of a PL function do exist (Fourer and Gay, 1995), but we adopt the simple form, based on the list of its breakpoints. For sake of simplicity, the following Definition 1 considers *continuous* PL functions (*i.e.* it maps any element in its domain to *at most* one image). However, the results presented in this paper apply in the discontinuous case.

DEFINITION 1. *A PL function ϕ is completely defined by a non-empty list of breakpoints $[(a_1, b_1), \dots, (a_n, b_n)]$, where a_i, b_i are resp. integral and float numbers, and $[a_1, \dots, a_n]$ is sorted in strictly increasing*

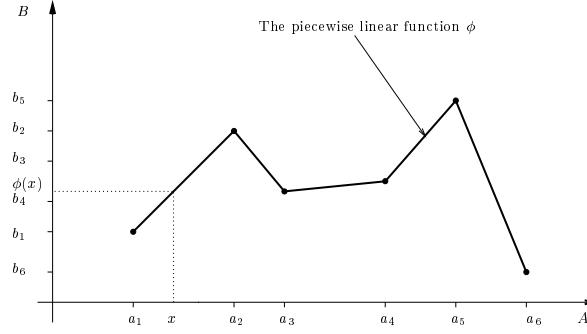


Figure 1. The PL function $\phi_{[(a_1, b_1), \dots, (a_6, b_6)]}$.

order. ϕ maps any a in the interval $[a_1, a_n]$ as follows:

$$\phi(x) = b_i + \frac{b_{i+1} - b_i}{a_{i+1} - a_i} * (x - a_i) \quad \text{if } x \in [a_i, a_{i+1}]. \quad (1)$$

We shall denote ϕ by $\phi_{[(a_1, b_1), \dots, (a_n, b_n)]}$.

ϕ is convex iff its corresponding slope sequence is sorted in increasing order. Figure 1 depicts a two-dimensional representation of non-convex PL function ϕ , where $n = 6$.

DEFINITION 2. Let X, Y and ϕ be respectively two arbitrary variables and a PL function. The triple (X, ϕ, Y) is called a PL constraint and it is simply denoted $Y = \phi(X)$. An assignment of values x, y to X and Y satisfies $Y = \phi(X)$, iff $y = \phi(x)$ holds.

We define a scheduling problem with PL optimization in the case of objective function minimization (without loss of generality).

DEFINITION 3. Scheduling Problem with PL optimization A scheduling problem PLKRFP with PL optimization is given by a triple $(\text{KRFP}, \text{PL}, \Phi)$, such that:

- KRFP is a Kernel Resource Feasibility Problem
- PL is a finite conjunction of PL constraints $Y_i = \phi(X_i)$ where X_i is the start time of the i^{th} activity and Y_i is its corresponding cost variable.
- $\Phi(Y_1, \dots, Y_N) = \sum_{i=1}^N Y_i$, where $Y_i = \phi(X_i) \in \text{PL}$ and N is the number of activities of KRFP, is the objective function to be minimized.

It can be established that a scheduling problem with PL constraints PLKRFP can be formulated as a constraint satisfaction problem consisting of the temporal constraints TC, PL constraints PL and resource constraints. The satisfaction problem for PLKRFP = (KRFP, PL, Φ) consists of fixing the start and end times of the activities and the cost variables Y_i such that the conjunction of the resource constraints of KRFP, TC and PL hold. The optimality problem of PLKRFP consists of finding a solution which minimizes the term $\Phi(Y_1, \dots, Y_N)$.

3. The Probe Backtrack Algorithm

This work extends (El Sakkout and Wallace, 2000) to solve the scheduling problem with PL optimization. The algorithm in (El Sakkout and Wallace, 2000) was applied for scheduling in the context of *minimal perturbation*: the goal was to re-store the feasibility of the original schedule with a minimum number of schedule changes. The work establishes that the discreteness of the probe values (*i.e. unimodular probing*) is guaranteed along the search by pure Simplex, even if the set TC of temporal constraints is extended by the “minimal perturbation constraints” (*i.e.* those stating that the objective function is the sum of the absolute changes of the activity start times). To apply the probe backtrack algorithm (El Sakkout and Wallace, 2000), it is necessary to ensure that the way we break up the problem guarantees that the prober will return discrete assignments for the start times. Indeed, the discreteness property is crucial for the integration of the prober into CP search procedure.

A pseudo-code description of the algorithm can be found in (El Sakkout and Wallace, 2000). It is described informally here. The algorithm is a complete repair-based extension of conventional constraint backtrack search: the backtrack search engine is supported by a *lookahead* prober which is a parameter of the algorithm. The role of the prober is to “suggest” a *super-optimal* assignment (*i.e.* better with respect to the optimization function than the *feasible* optimal, but only partially consistent) for unassigned variables, satisfying the constraints of sub-problem it operates on. The prober “suggestions” are intended to be potentially good assignments. The key-role of the prober is to focus the backtrack search component on search regions where the probe violates resource constraints.

Now, we briefly present the operational behavior of the algorithm. The algorithm performs a depth-first traversal of a binary tree search. At each node of the search tree, the prober is invoked and it comes back with a value assignment which is super-optimal *w.r.t.* the objective

function. Based on this assignment, the resource constraints are monitored for violation. If there is no resource violation, then the assignment is globally consistent and a problem solution is found. Otherwise, the algorithm selects a constraint violated by the assignment and posts a search decision which potentially leads the prober into an other assignment closer to satisfying the violated constraint. A backtrackable choice corresponding to the disjunction of the search decision and its negation is created. Typically, the search decision consists of a *distance constraint* between two activities competing for a resource. Posting the decision triggers constraint propagation. From an optimization perspective, at each solution node the algorithm incorporates *cost bound constraints* in a Branch & Bound style.

The variant of the probe backtrack algorithm we apply for solving the scheduling problem with PL optimization embeds a cooperative hybridization of a domain/range reduction CP-based solver, on one side, and an LP-based linear solver, on the other. It conducts search in two interleaved and strongly inter-dependent phases: 1) resource feasibility and 2) PL & temporal probing. In the resource feasibility phase, the algorithm relieves resource contention by ordering the temporal variables. The probing consists of an optimization of Φ subject to the PL and temporal constraints. The returned probe represents a partially consistent optimal solution and satisfies the activity orderings decided in the resource feasibility phase.

The central and delicate issue is the way the PL constraints are treated by the algorithm. A first approach would be to globally delegate the handling of those hard combinatorial constraints to the resource feasibility phase: both the resource and PL constraints are monitored for violation. The main advantage offered by this approach is that the cheap prober operates on a tractable class of linear temporal constraints. However, serious limitations handicap this: since the prober will not consider the PL constraints, it will potentially come back with “over-optimistic” and “irrelevant” temporal values. The bad quality of the probe forces the search component to unnecessarily explore potentially inconsistent branches. This can lead to an excessive search tree.

The prober handles the set of PL constraints PL, in addition to the temporal constraints TC. At the same time, we believe that the potential pruning opportunities offered by the piecewise linearity needs to be emphasized and fully exploited especially in a scheduling environment. The reason is that the PL constraints link the cost with the temporal activity ordering.

Additionally, the algorithms proposed start from the hypothesis that mathematical programming techniques can provide a basis for probing in the presence of PL constraints.

The prober operates on a formulation, say \mathcal{M} , of $\text{TC} \wedge \text{PL}$ which is a more or less accurate representation of its solution set. This formulation may be more or less accurate, and two kinds of probers are investigated in the next two sections: an MIP-based prober and a linear relaxation-based prober. In the following, C denotes an arbitrary PL constraint $Y = \phi_{[(a_1, b_1), \dots, (a_n, b_n)]}(X)$ (or $Y = \phi_{a,b}(X)$) in the set PL . In addition, $\min(X)$ (resp. $\max(X)$) designates the lower (resp. upper) bound of the finite domain variable X .

4. Mixed Integer Programming based Probing

Four methods to linearize the *convex* PL constraints by the introduction of new variables and linear constraints exist in the literature (Fourer, 1992; Ho, 1985). These methods provide four different linear formulations that are equivalent in the sense that they generate the exact solution set of PL. Their complexities and inter-relationships have been studied in (Ho, 1985), where it has been established that two of them are dual forms in some sense of the remaining ones. Consequently, we consider exclusively two non-dual ones which are called the λ and δ methods. Convex PL constraints can be solved by pure LP. For example, R. Fourer had applied specialized variant of Simplex (Fourer, 1992). The modeling of *non-convex* PL constraints needs additional non-linear “stipulation” constraints which require MIP (Williams, 1994). The convexity of the PL constraints guarantees that the optimal solution satisfies the stipulation constraints.

Special Ordered Set (SOS) constraints were introduced in (Beale and Tomlin, 1970). They are commonly used in mathematical programming (Nemhauser, Johnson and de Farias, ; Williams, 1994). A particular instance named “SOS of type 2” (SOS2) offers a natural formulation of PL constraints. An SOS2(L), where $L = [x_1, \dots, x_n]$ is a list, is satisfied iff at most two variables are assigned non-zero values, and if two variables are assigned non-zero values then they are necessarily adjacent.

4.1. THE δ -MODEL

Assume that $[s_1, \dots, s_{n-1}]$ is the sequence of slopes of their respective linear fragments of C . The δ -model (Williams, 1994) expresses X as a linear combination of distances between breakpoints with real coefficients δ_i for i in $[1, n - 1]$. Each variable δ_i is bounded in $[0, 1]$

and it represents the “used proportion” of the length of the interval $[a_i, a_{i+1}]$ to make up an assignment for X . The value of Y is the sum of the successive interpolations of δ_i via the i^{th} fragment with slope s_i . Intuitively, the model generates the curve of ϕ from a_1 to a_n and characterizes the function (Dantzig, 1963; Williams, 1994). To guarantee the (global) optimality even in the non-convex case, a stipulation constraint $R([\delta_1, \dots, \delta_{n-1}])$ on the proportions needs to be added to the model.

An assignment $\{\delta_1 \mapsto d_1, \dots, \delta_{n-1} \mapsto d_{n-1}\}$ solves $R([\delta_1, \dots, \delta_{n-1}])$ iff the following property holds: if d_i is non-zero then all preceding proportion equal 1 and all the succeeding proportion equal 0. The latter property ensures that the proportion values are “co-ordinated” to lead to summation of distances along the axes corresponding to X and Y . The constraint $R([\delta_1, \dots, \delta_{n-1}])$ can be captured by an SOS2 and other linear constraints, with the price of adding n extra variables γ_i . Each γ_i is meant to model the selection of a linear segment. The other constraints ensure that X and Y “truly” represent distances along their axes (Williams, 1994). Thus, C is modeled as follows:

$$\left(\begin{array}{l} X = a_1 + \sum_{i=1}^{n-1} (a_{i+1} - a_i) * \delta_i \\ Y = b_1 + \sum_{i=1}^{n-1} (b_{i+1} - b_i) * \delta_i \\ 0 \leq \delta_i \leq 1, \quad \forall i \in [1..n - 1] \\ 0 \leq \gamma_i \leq 1, \quad \forall i \in [0..n - 1] \\ \sum_{j=0}^{n-1} \gamma_j = 1 \\ \delta_i = \sum_{j=i}^{n-1} \gamma_j, \quad \forall i \in [1..n - 1] \\ SOS2([\gamma_0, \dots, \gamma_{n-1}]) \end{array} \right. \quad (2)$$

If the domain of X is refined by constraint propagation at a node, the δ -model needs to be updated dynamically. The aim is to infer as much information as possible to improve the quality of the relaxed solution at each MIP tree node and, thus, to prune the search space.

For a given constraint C , we attach a rule-based pruning handler. The handler is triggered as soon as one of the bounds of X changes and before MIP prover runs. The underlying idea is that some cuts on the proportion variables can be generated as search constrains the domain of the temporal variable. The handler maintains incrementally two indices of proportions, say l and r . The indice l (resp. r) is the indice of the left-most (resp. right-most) proportion variable indice which is still non-ground.

If the lower bound of X excludes a linear segment defined on $[a_i, a_{i+1}]$, then any solution of C must be in the intersection of the half-spaces $a_{i+1} \leq X$ and $b_{i+1} \leq Y$. This means that the corresponding proportion of the linear segment must be “fully” used. We can force, therefore, the proportion δ_i to be 1. By a similar reasoning, we can set to zero

the proportion variable of any linear segment becoming excluded by the upper bound of X . The resulting pruning rules, which might be applicable more than once, are as follows:

- R1: $\mathcal{M} \wedge Y = \phi_{\bar{a},\bar{b}}(X) \longrightarrow \mathcal{M} \wedge Y = \phi_{\bar{a},\bar{b}}(X) \wedge \delta_i = 1$
if $a_{i+1} \leq \min(X)$
- R2: $\mathcal{M} \wedge Y = \phi_{\bar{a},\bar{b}}(X) \longrightarrow \mathcal{M} \wedge Y = \phi_{\bar{a},\bar{b}}(X) \wedge \delta_i = 0$
if $\max(X) \leq a_i$

Furthermore, if the lower bound of X belongs to the segment defined on $[a_i, a_{i+1}]$, a cut can be generated and added to \mathcal{M} . Indeed, each solution of \mathbb{C} is necessarily in the “super-feasible” space defined by intersecting the half-spaces defined by the inequations $\min(X) \leq X$ and $\phi(\min(X)) \leq Y$. Consequently, \mathbb{C} can be strengthened by enforcing a lower bound on the corresponding proportion variable. The same constraint reasoning applies on the right-hand side of the domain of X : an inequation imposing an upper bound on the δ of the segment bounding $\max(X)$. This leads to two further pruning rules:

- R3: $\mathcal{M} \wedge Y = \phi_{\bar{a},\bar{b}}(X) \longrightarrow \mathcal{M} \wedge Y = \phi_{\bar{a},\bar{b}}(X) \wedge \frac{\min(X) - a_i}{a_{i+1} - a_i} \leq \delta_i$
if $a_i < \min(X), \min(X) < a_{i+1}$
- R4: $\mathcal{M} \wedge Y = \phi_{\bar{a},\bar{b}}(X) \longrightarrow \mathcal{M} \wedge Y = \phi_{\bar{a},\bar{b}}(X) \wedge \delta_i \leq \frac{\max(X) - a_i}{a_{i+1} - a_i}$
if $a_i < \max(X), \max(X) < a_{i+1}$

When no rule is applicable, the handler updates the indices accordingly so that the next time it is triggered it would inspect only the proportion variables with indices between l and r . The handler rules are sound: they do not remove solutions. Indeed, they add *valid* constraints.

4.2. THE λ -MODEL

The underlying idea of the λ -model (Dantzig, 1963) is that any point of the convex hull of \mathbb{C} is a linear convex combination of the breakpoints. For each breakpoint vertex (a_i, b_i) , we associate a variable λ_i , interpreted as its “weight”. The additional stipulation condition imposes that the sequence of weights solve the SOS2 constraint to encode two non-zero neighboring weights: those two (possibly equal) breakpoints correspond to the segment on which solution point lies. Thus, each constraint \mathbb{C} is modelled by:

$$\begin{cases} X = \sum_{i=1}^n a_i * \lambda_i \\ Y = \sum_{i=1}^n b_i * \lambda_i \\ \sum_{i=1}^n \lambda_i = 1 \\ 0 \leq \lambda_i \leq 1, \quad \forall i \in [1..n] \\ SOS2([\lambda_1, \dots, \lambda_n]) \end{cases} \quad (3)$$

Similarly to the δ -model, we attach a rule-based pruning handler which is triggered if a bound of X changes and before MIP runs. For lack of space, we do not present the rules formally. The handler maintains incrementally the left and right-most indices, say l and r , of the weights which can contribute to a solution: the non-zero weights. Initially, l and r are set to 1 and n respectively. The first rule captures the fact that if a segment defined on $[a_i, a_{i+1}]$ is strictly on the left of $\min(X)$, then necessarily its weight λ_i can be set to zero. A similar rule sets to zero the weight λ_{i+1} if $\max(X)$ is excluded by the segment defined on $[a_i, a_{i+1}]$. These two rules are taken from (Refalo, 1999).

We can strengthen the pruning by generating further cuts on the weights. The idea is to infer a new upper bound on λ_l as soon as $\min(X)$ belongs to $[a_l, a_{l+1}]$. The new upper bound captures the case where the weight λ_l is maximal: it corresponds to the case $\min(X)$ participates to a solution. A similar right-hand side reasoning is used to infer an upper bound on the weight λ_{r-1} reflecting the case where the solution corresponds to λ_{r-1} which is minimal: $\max(X)$ is assigned to X .

The handler ends by updating the indices l and r accordingly and waits for further reduction of the domain of X . The rules are sound: they preserve the solution set of \mathcal{M} .

4.3. MIP PROBERS AND TIME DISCRETENESS

It is necessary now to check whether the λ and δ probers guarantee the *integrality* of the temporal variables. It happens that they both might come up with an optimal solution assigning non-integral values to temporal variables.

After much investigation, we have discovered that cost bounds cause the violation of the discreteness property. To overcome this problem, we exclude the cost bounds from \mathcal{M} . After each MIP call, the algorithm tests whether the returned optimum is worse than the best cost found so far. If the test succeeds, then the search component prunes the node and forces a backtrack. Indeed, all the sub-tree rooted at the node can not yield a cost better than the current cost.

5. Linear Relaxation Technique for Probing

In the previous section, the MIP probers solve formulations which have the same solution set as C. The other alternative is to let the CP search control the Simplex solver instead of the MIP system. As in Refalo's work (Refalo, 1999), we encode C by a *linear* relaxation $R(C)$ which

does not require new variables. Since the disjunctive aspect of C is dropped from $R(C)$, further search branching is required to satisfy C . The branching exploits the information provided by the relaxed solution. If the relaxed solution returned by the Simplex-based prober (the probe) assigns values x and y to X and Y which do not solve C (*i.e.* $\phi(x) \neq y$) then a “relaxation conflict” *w.r.t.* C is pending for repair: *i.e.* the probe may now violate PL constraints as well as resource constraints. In the case where a PL constraint is violated, and the probe backtrack procedure selects that constraint for repair, the relaxation conflict may be resolved as in (Refalo, 1999), by splitting the domain of X and creating a backtrackable choice:

$$X \leq x \vee X \geq x + 1. \quad (4)$$

A set of conflicts is associated with each node. The conflicts belong to two types. As before, the first type consists of resource violations, whereas the second represents the relaxation violations: the discrepancy between the relaxed cost y and the “real” cost $\phi(x)$. The adopted search strategy resolves resource conflicts *before* relaxation conflicts: it favors feasibility against optimality. This accelerates the convergence towards a problem solution and allows the resulting cost bound constraint to prune the search. In fact, the cost bound constraint could *a priori* prune some choices when fixing relaxation conflicts. Resolving the relaxation conflicts before the resource conflicts could drive the algorithm into search regions which are potentially inconsistent *w.r.t.* resource requirements. We hypothesize that our strategy will be better on tightly constrained scheduling problems, while for loosely constrained problems the converse may be true. Both the resource conflict and the relaxation conflict selection heuristics use the *maximum infeasibility* principle.

Unlike the variable fixing and cutting plane strategies of (Refalo, 1999), the linear relaxation $R(C)$ of a PL constraint C is the formulation of its *exact convex hull*: the smallest *convex hull* generated by the breakpoints $(a_1, b_1), \dots, (a_n, b_n)$ and the domains of X and Y . The convex hull procedure is incremental since it maintains and updates the convex hull of each C as the search progresses. As soon as a bound of X and/or Y changes, the procedure reacts by locally strengthening, if necessary, the convex hull formulation by accumulating the appropriate cuts according to the new bounds of X and/or Y .

6. Evaluation and Analysis

This section reports results of a systematic evaluation of the hybrid solver under different probers. The hybrid algorithm run with the prober P, called H(P), was implemented in the CP platform ECLⁱPS^e 5.1.3, where CPLEX 6.5 played the role of the external linear solver. The evaluation was carried out on PCs with Pentium II 450 MHz processors. The timeout is 30 CPU minutes.

6.1. SETTING UP THE BENCHMARK INSTANCES

A configurable data generator outputs a randomly-generated and *feasible* PLKRFP containing the following entries:

- A set of N activities, with their *fixed* start/end time points, for a single resource pool over the restricted time horizon T
- A set TC of *feasible* binary temporal constraints. This generation is guided by a temporal density parameter θ , which is proportional to the probability of the existence of a temporal constraint between any pair of start/end points of activities.
- A set of N objective functions defined over the time horizon. The complexity of a PL function is a measurement of “how hard” it is to find its global minimum, regardless of the other constraints.

From any feasible PLKRFP, we can create many benchmarks by varying the following generator scenario parameters:

1. For each temporal variable, the maximum/minimum allowed time shift around its initial value in the *feasible* PLKRFP. This imposes lower and upper bounds on the variable (*i.e.* “time window”).
2. The amount of resource reduction R: the desired percentage of resources to be pulled out (*i.e.* saved) from the set of resources consumed by PLKRFP.

H solves a test by re-scheduling the activities within their time windows subject to the updated set of resources. To get a large-scale search space, we do not set any restriction on the amount of time shifts of activities: each activity can start at any time point within the horizon. The tightness of a test is defined by the tightness degrees of its resource and optimization parts.

The tightness of the resource part is captured by the amount of resource reduction: requiring larger resource saving could tighten aggressively the problem. Consequently, it can cause terrific perturbations (changes) to the initial PLKRFP.

Table I. Definition of the four classes of experiments.

Class	Nb of tests	PL curves	R(%)	T	θ
\mathcal{C}_1	24	“nearly convex”	0	197	0.5
\mathcal{C}_2	24	“nearly convex”	30	197	0.5
\mathcal{C}_3	24	“nasty”	0	197	0.5
\mathcal{C}_4	24	“nasty”	30	197	0.5

The tightness of the optimization part reflects the complexity of the PL curves. In this paper, a curve is easy if it is “nearly convex”: its distance to convexity is small. Intuitively, this distance should be somehow proportional to the variations of the signs of its slopes. The more often the slope sign changes, the harder is the curve. We choose to consider a curve “nearly convex” if it contains *at most* three local minima. Indeed, this restricts, to a certain extent, the sign variations within the slopes. This obviously covers the convex case. By contrast, a curve becomes “nasty”, when more restriction-free “disorder” occurs within its list of slope signs (*i.e.* higher number of local minima). The generator distributes, by monitoring the slopes, the number of minima of a nasty curve such it is arbitrary, but greater than 4.

Depending on the complexities of its resource and optimization components of a test, we run the data generator, such that 96 experiments are created and partitioned into four classes \mathcal{C}_i , where $i \in [1..4]$. A summary of the run scenario is depicted in Table I. In addition, we vary the number of activities such that each class \mathcal{C}_i contains 8 instances with N activities, where N is in $\{30, 60, 90\}$.

6.2. COMPUTATIONAL RESULTS

The experimentation showed that the MIP probers are surprisingly ineffective without the pruning rules. They significantly speed up the MIP search by providing a tighter linear relaxation which is as consistent as possible with the CP store. This shows the high sensitivity of the performance to constraint passing between the prober and the CP solver. $H(\lambda\text{-prober})$ outperforms $H(\delta\text{-prober})$, especially on large-scale problems. This might be explained by the fact that δ -model is more complicated than the λ -model: it contains more rows and columns.

In Figure 2, we sketch the computational results of running H with the λ -prober ($H(\lambda)$) and relaxation-based prober ($H(\text{relax})$). Regarding the number of tests solved to optimality (*i.e.* the search is

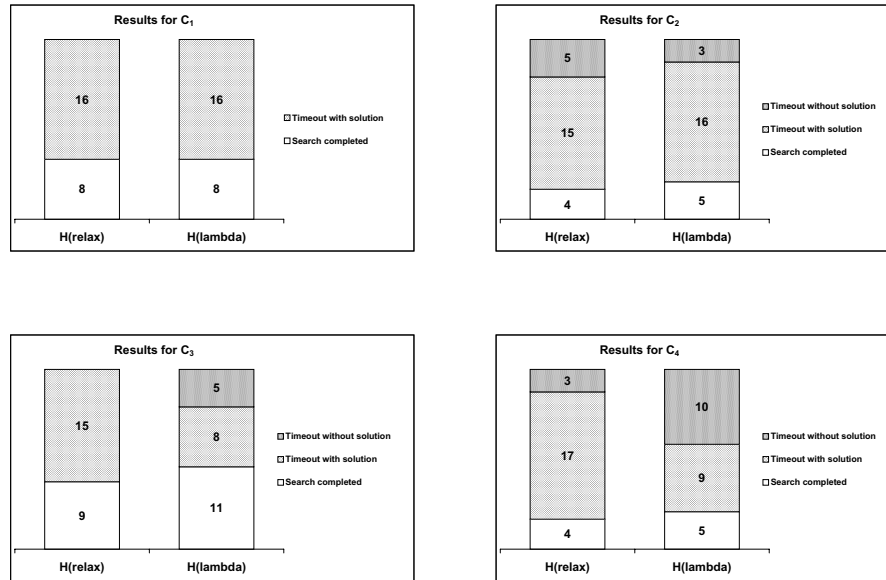


Figure 2. Results of running H(relax) and H(lambda) on the four test classes.

completed), H(lambda) is, at least, as good as H(relax): MIP is better at optimality.

Run on C_1 , H(lambda) and H(relax) behave equally well and they solved all the 24 tests. Since the resource constrained problem has a sparse solution space, few CP choice points are required in both cases. Furthermore, the response times and the qualities of the probes are very close, because the PL curves are “easy”. We can draw the same conclusion from running the algorithm on C_2 . The only difference is that H(lambda) behaves slightly better. Indeed, H(relax) solved less tests: failed to solve 2 tests that H(lambda) successfully solved. This could be seen as two counter examples to the “repair resources first” strategy. To avoid Simplex initialization overheads, it could be beneficial to solve all the easy relaxation conflicts by a single MIP, rather than delaying and fixing them later stages by separate Simplex invocations.

Meanwhile, H(relax) solved all tests of C_3 . Since the resource constraints have a dense solution space and the curves are nasty, one would expect H(lambda) to be more effective. This is not the case, because H(lambda) failed on 5 tests. In the search tree explored by H(lambda), the traversal of the branches that lead to failure is time-consuming because they require invoking an expensive MIP at each node. This

is why on the 5 tests, H(λ) times out before it explores the potentially successful branches which were pending for exploration.

The performance gap between the solvers gets clearly wider on \mathcal{C}_4 , where the resources are more limited. The MIP search invests too much computational effort (the curves are nasty) in obtaining optimal probes that are not easily repaired to be feasible due to the limited resources. Thus, the expensive MIP probes are likely to be revised by the CP search. By contrast, the adopted search strategy which repairs resource before relaxation conflicts makes H(relax) more oriented towards resource feasibility: it delivers a first solution faster and, thus, enabling cost-pruning at early stages. In fact, the search strategy of H(λ) should be interpreted from the opposite angle: the MIP solver is invoked to repair all the relaxation conflicts at once *before* focusing on relieving the resource contentions.

7. Conclusion

In *scheduling*, most common optimization criteria may be described or approximated by piecewise linear (PL) objective functions. This paper presents an effective approach for scheduling problems with PL constraints, based on probe backtrack CP-LP hybridization. A number of alternative probing strategies were systematically evaluated and compared. The results demonstrated that the linear relaxation method with a “repair resources first” strategy is particularly effective at solving scheduling-like problems with disjunctive constraints and PL objective functions.

References

- Aggoun, A. and Beldiceanu, N. Extending CHIP in order to solve complex scheduling and placement problems. *Premières Journées Francophones sur la Programmation en Logique*, 1992.
- Bartusch, M., Möhring, R. H. and Radermacher, F. J. Scheduling project networks with resource constraints and time windows. *Annals of Operations Research*, 16:201–240, 1988.
- Beale, E. M. L. and Tomlin, J. A. Special facilities in a general mathematical system for non-convex problems using ordered sets of variables. In *Proceedings of the Fifth International Conference on Operations Research*, pages 447–445, London, 1970. Tavistock Publications.
- Bertsimas, D., Darnell, Ch. and Soucy, R. Portfolio construction through mixed-integer programming at grantham, mayo, van otterloo and company. *Interfaces*, 29(1):49–66, January 1999.
- Dantzig, G. B. *Linear Programming and Extensions*. Princeton university Press, 1963.

- Dechter, R., Meiri, I. and Pearl, J. L. Temporal constraint networks. *Artificial Intelligence*, 49:61–95, 1991.
- El Sakkout, H. and Wallace, M. Probe backtrack search for minimal perturbation in dynamic scheduling. *Constraints, Special Issue on Industrial Constraint-Directed Scheduling*, 5(4), 2000.
- Fourer, R. A Simplex algorithm for piecewise-linear programming iii: Computational analysis and applications. *Mathematical Programming*, 53:213–235, 1992.
- Fourer, R. and Gay, D. M. Expressing special structures in an algebraic modeling language for mathematical programming. *ORSA Journal on Computing*, 7:166–190, 1995.
- Ho, J. K. A successive linear optimization approach to the dynamic traffic assignment problem. *Transportation Science*, 14:295–305, 1980.
- Ho, K. James Relationships among linear formulations of separable convex piecewise-linear programs. *Mathematical Programming Study*, 24:126–140, 1985.
- Ignizio, J. P. *Goal Programming and Extensions*. Lexington Books, Lexington, Mass., 1976.
- Nemhauser, G. L., Johnson, E. L. and de Farias, I. R. A generalized assignment problem with special ordered sets : A polyhedral approach. To Appear in the *Journal of Mathematical Programming*.
- Refalo, P. Tight cooperation and its application in piecewise linear optimization. In *Proceedings of 5th International Conference of Principles and Practice of Constraint Programming (CP'99)*, volume 1713 of *Lecture Notes in Computer Science*, pages 375–389, Alexandria, Virginia, USA, October 1999. Springer.
- Schniederjans, M. J. *Goal programming : methodology and applications*. Kluwer Academic Publishers, Boston, 1995.
- Williams, H. P. *Model Building in Mathematical Programming*. John Wiley & Sons, third revised edition, 1994.