Tractable Optimal Competitive Scheduling

Jeremy Frank and **James Crawford** and **Lina Khatib** * and **Ronen Brafman** †
Computational Sciences Division
NASA Ames Research Center, MS 269-3
frank@email.arc.nasa.gov
Moffett Field, CA 94035

Abstract

In this paper we describe the problem of Optimal Competitive Scheduling, which consists of activities that compete for a shared resource. The objective is to choose a subset of activities to schedule, sequence them, and decide how much time they are allowed, in such a way that temporal and resource constraints are satisfied and overall schedule quality is maximized. While most such problems are $\mathcal{NP}$-complete, very restricted versions of this problem are known to be tractable. In this work we describe tractable variations on this problem that correspond to realistic scheduling problems. The first class of tractable OCS problems arises due to limitations on the objective function that permit casting the problem as a Linear Program; with one additional assumption on activity feasibility windows, we identify a problem class where an optimal activity ordering can be found in polynomial time. The second class arises by reformulation of the problem as a Valued Constraint Satisfaction Problem and exploiting known results on tractability. We describe implementations of special-purpose algorithms designed to solve tractable OCS problems, and identify different solver performance characteristics based on properties of the problem instances.

Introduction

This paper is concerned with the problem of allocating resource to multiple users within a pre-defined time period. The users have preferences describing how they value different allocations of the resource. We seek an optimal allocation schedule for this resource. We are particularly motivated by the problem of scheduling resources for various NASA and industry applications, and cite a few motivating problems. NASA's Deep Space Satellite Network (DSN) is needed for transmitting data from various space missions to Earth. Each mission has different needs for DSN time, depending on satellite and planetary orbits. Typically, the DSN is over-subscribed, in that not all missions will be allocated as much time as they want. Scheduling satellite downlink (also known as Range Scheduling (BWH04)) is quite similar to DSN scheduling, with multiple satellites requiring time to downlink data of varying levels of importance. Telescope observation scheduling involves a telescope at a fixed location that must observe targets for varying periods of time, with science quality depending on the characteristics of the observations (Bre96). Finally, satellite observation scheduling involves choosing targets for a satellite with a steerable instrument, again with schedule quality depending on the targets observed and the characteristics of the observations (WS00).

These problems have several common themes:

- Activities have release times, due dates and processing times, and compete for a shared resource.
- Activities may have other constraints imposed on them (e.g. minimum task separation, synchronization constraints, and so on.)
- Activities have value based on the resource allocation, and those values can be combined (usually added) to calculate a global utility of a legal schedule.
- The problem spans a finite horizon, and the goal is to find a feasible schedule maximizing value.

We refer to such problems as *Optimal Competitive Scheduling* (OCS) problems. The problems feature time and resource constraints, but unlike problems of minimizing makespan, require packing as many requests as possible into the designated horizon in the best way possible [1]. OCS generalizes the problem of scheduling activities on unary resources with release times, due dates, processing times and weighted rejection penalties ($1|r_i p_i d_i|w_i U_i$ in Graham's notation (Brü98)) in that an activity A may have an associated *preference function* f_A that describes the value based on the resource allocation, rather than just a value. Thus, the general decision problem is $\mathcal{NP}$-complete. OCS also generalizes the recently introduced Temporal Knapsack Problem (BFH$^+$05) and Overconstrained Disjunctive Temporal Problems with Preferences (PMP05); in OCS, activities may have a wider set of inter-activity constraints and preference functions.

Numerous restricted versions of this problem have been shown to be tractable. One example is the case where the resource conflicts have been resolved, the preference function obeys some reasonable restrictions, and all that remains is to calculate optimal activity durations (MMK$^+$04). These

*QSS
†Ben Gurion University, and USRA-RIACS

[1] OCS can be thought of as over-subscription planning, limited to scheduling problems.

problems can be posed as Linear Programs. Another example is the case in which activities have fixed start times, duration and value, and all that remains is to choose the best feasible subset of activities (SS00). These problems are addressed by virtue of methods exploiting the underlying graph structure of the problem.

In this paper, we push the state of the art in terms of tractable algorithms for OCS problems. We begin by formally defining the OCS. Next, we extend tractability results for problems with flexible duration and preferences. We then extend tractability results for the case of choosing subsets of activities and ordering these activities. Here, we begin with the case of activities of fixed duration and fixed preference value, then handle finite numbers of duration choices with preferences over these durations. We describe the results of empirical studies to determine the best implementation in the case of choosing and ordering of activities. Finally, we describe some related work, and conclude with future work.

Optimal Competitive Scheduling

An OCS problem consists of a resource R, a time horizon $[0..h]$, and a set of activities $\mathcal{A}$. For activity $A \in \mathcal{A}$, let A_s be the start time, A_d be the duration, and A_e be the end time of activity. All activities A use one unit of resource R. For each activity A we have: $A_s + A_d = A_e$, $0 \leq a_{s_{lb}} \leq A_s \leq a_{s_{ub}} \leq h$, and $0 \leq a_{e_{lb}} \leq A_e \leq a_{e_{ub}} \leq h$. Similarly, activities can have minimum and maximum durations: $0 \leq a_{d_{lb}} \leq A_d \leq a_{d_{ub}} \leq h$. The problem may also contain Simple Temporal Constraints (DMP91) of the form $a \leq A_* - B_* \leq b$, where $A_* \in \{A_s, A_e\}$ and $B_* \in \{B_s, B_e\}$, which can enforce activity orderings. Let T be a fixed time assignment schedule; $A_s(T), A_e(T), A_d(T)$ refer to the assignments to variables of activity A in the schedule.

Generally, the quality of a schedule can depend both on start time and duration, leading to preferences of the form $f_A(A_s(T), A_d(T))$. For example, in telescope observation scheduling, the schedule quality depends on the telescope elevation achieved during an observation, since the line-of-sight through the atmosphere is reduced at higher elevations (FK05). These objective functions can have local optima that make optimization hard, even for the simple case of *bi-linear* objective functions (objective functions of the form $\sum_i \sum_{j \neq i} a_{ij} x_i x_j$) coupled with linear constraints (SN88).

Problems often have preferences over *sets* of activity allocations. For example, in telescope observation scheduling, astronomers express preferences over total observing time for a single target. This requires observing the target over the course of several nights. The preference is expressed over the total scheduled time on the target. Similarly, in the satellite range scheduling and DSN scheduling problems, it may be necessary to schedule several successive downlinks for one spacecraft; the preference in this case is expressed over the total scheduled downlink time. Let π be a partition of the activities in $\mathcal{A}$ and let π_i be the i^{th} element of the partition. The general preference function for π_i has the form $f_{\pi_i}(A_s(T)...A_d(T))$ for all $A \in \pi_i$. Throughout this paper, we will limit ourselves to preferences over sums of activity duration, that is, $f_{\pi_i}(\sum_{A \in \pi_i} A_d(T))$.

Tractable Choice of Activity Duration

In this section we consider tractable versions of the OCS that center on the choice of activity duration based on the preference function. Specifically, we extend the Linear Programming (LP) formulation described in (MMK$^+$04) to develop tractable classes of OCS problems.

The Simple Temporal Problem with Preferences

The Simple Temporal Problem with Preferences (STPP) consists of a set of events $\mathcal{E}$, including a special event E_0 whose value is constrained to be zero. Events are constrained by a set of Simple Temporal Constraints, of the form $a \leq |E_i - E_j| \leq b$, $E_i, E_j \in \mathcal{E}$. Finally, event distances have an associated preference function. We assume there is a dependent variable $D_{ij} = E_i - E_j$, and write the preference function $f_{ij}(D_{ij})$. In (MMK$^+$04) it is shown that, if $f_{ij}(D_{ij})$ is a *convex, piecewise linear function* and the Simple Temporal Constraints are consistent, then an assignment of time values to events maximizing $\sum_{i,j} f_{ij}(D_{ij})$ can be found by solving a Linear Programming (LP) problem. Let $f^k_{ij}(D_{ij})$ be the k^{th} linear component of $f_{ij}(D_{ij})$. The LP formulation contains a variable E_i for each event, a variable D_{ij} for each event distance, and a variable V_{ij} for each preference function $f_{ij}(D_{ij})$. The LP is then

$$\begin{array}{lll} \max & \sum V_{ij} & \\ subj & \forall E_i, E_j \in \mathcal{E} & D_{ij} = E_i - E_j \\ & \forall E_i, E_j \in \mathcal{E} & a \leq E_i - E_j \leq b \\ & \forall k & V_{ij} \leq f^k_{ij}(D_{ij}) \end{array}$$

The OCS problem is "activity" centric, but the set of activities $\mathcal{A}$ induces a set of events $\mathcal{E} = A_* | A \in \mathcal{A}$. It is easy to generalize STPPs to handle preferences over sums of activity durations according to a partition. Suppose the number of partitions is p; we now only need to introduce one variable D_p for each duration sum and one variable V_p for each preference function We rewrite the LP as follows:

$$\begin{array}{lll} \max & \sum V_p & \\ subj & \forall A \in \mathcal{A} & A_d = A_e - A_s \\ & \forall A, B \in \mathcal{A} & a \leq A_* - B_* \leq b \\ & \forall i & D_i = \sum_{A \in \pi_i} A_d \\ & \forall k & V_p \leq f^k_p(D_p) \end{array}$$

Optimal Activity Ordering

The STPP assumes that there are no resource violations consistent with the temporal constraints. To solve an OCS problem using STPPs, ordering activities may be required to guarantee no resource violations. However, an arbitrary set of activity orderings combined with the original constraints may be inconsistent. While this condition can be checked very efficiently using Bellman Ford or a variety of other techniques, *all* activity orderings may be inconsistent, i,e, the scheduling problem may have no feasible solution if all activities must be scheduled. In this case, some activities must be rejected. Finally, arbitrary orderings may be suboptimal. In general, these problems can only be resolved using combinatorial search.

We now describe our first tractable OCS problem. If only one activity can use the resource at any time, and the activities' feasibility windows obey a relatively simple condition, there is a way to optimally order activities, leading to an STPP that can be solved using the LP formulation described previously. We begin with some definitions:

Definition 1 *Two requested activities A, B are a* containment pair *if ($a_{s_{lb}} < b_{s_{lb}}$ and $b_{e_{lb}} < a_{e_{lb}}$).*

Definition 2 *Given an OCS problem on a unary resource R whose metric temporal constraints are feasible and limited to time windows and activity ordering such that there are no containment pairs. The* natural order $\prec_N$ *of activities is the ordering induced by the start times of the associated visibility windows, i.e. $A \prec_N B$ iff $a_{s_{lb}} < b_{s_{lb}}$.*

Certain scheduling problems turn out to have no containment pairs. For example, downlink windows for a constellations of satellites flying close together at the same altitude and the same orbit have this property, as do limited observation scheduling problems in astronomy.

Definition 3 *Given an OCS problem on a unary resource R whose metric temporal constraints are feasible and limited to time windows with no containment pairs. and a grounded schedule T. Let $A_s(T) < B_s(T)$. A, B are* adjacent *iff there is no request C such that $A_s(T) < C_s(T) < B_s(T)$. A, B are* mismatched *if $B \prec_N A$ and $A_s(T) < B_s(T)$.*

Theorem 1 *Given an OCS problem on a unary resource R whose metric temporal constraints are feasible and limited to time windows. If $\mathcal{A}$ is a set of activities with no containment pairs, $\mathcal{A}$ has no feasible solution with all activities, or an optimal grounded schedule T with no mismatched pairs*

To prove this theorem, it is enough to prove that given any feasible schedule S with mismatched pairs, we can derive a feasible schedule with no mismatched pairs of the same value. As a result, any optimal schedule can be transformed into an optimal schedule with no mismatched pairs. First we provide a lemma about swapping adjacent mismatched pairs, then an algorithm for deriving equivalent schedules with no mismatched pairs.

Lemma 1 *Given an OCS problem whose metric temporal constraints are feasible and limited to time windows and activity ordering, and assume there are no containment pairs. Let S be any feasible grounded schedule. Let A and C be any two activities. If A and C are adjacent and mismatched, then there is a feasible schedule T of the same quality as S in which A and C are not mismatched.*

Proof of Lemma (see illustration in Figure 1): Since A and C are mismatched we know $A \prec_N C$ and therefore $a_{s_{lb}} < c_{s_{lb}}$. Let I be the smallest interval containing the time allocated to both A and C, that is, $[C_s(T), A_e(T)]$. I is in the intersection of the visibility window of both activities satisfied by A and C, therefore the time span allocated to each of A and C may occur anywhere in I. Because A and C are adjacent, there is no other request with allocated time in I to interfere with repositioning A and C. Since the pair is mismatched and not a containment pair, we can swap A and B in the ground schedule without adjusting their size and without violating any constraints. Keeping their sizes the same leaves the schedule quality unchanged. □

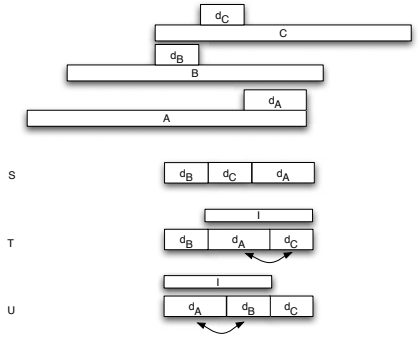

Figure 1: Swapping Adjacent Mismatched Pairs in Grounded Schedule.

Proof of Theorem 1: Let S be any optimal schedule for R. Let k be the number of activity allocations in S. Make $k-1$ traversals of the timeline, left-to-right: whenever two adjacent allocations appear that are mismatched, swap them as described in the lemma. After the $k-1$ traversals, there are (according the the theory of bubble sort!), no remaining mismatched pairs. Since S was optimal, so is the bubble-sorteded schedule.

Since there are no containment pairs, there is a unique total order of activities, guaranteeing termination of the sort. Since having a mismatched pair in a schedule implies having an adjacent mismatched pair, this completes the proof for Theorem 1. □

We note that this construction still works if there are temporal constraints between pairs of time-points. The only change is that we must check for containment after propagation of all temporal constraints; nontrivial timepoint separation and activity orderings opposite to the natural ordering $\prec_N$ may induce containments not apparent in the time windows. Finally, we consider the case of a resource with capacity c, and assume all activities use one unit of resource. Since we preserve the assumption that there are no containment pairs, if $d > c$ activities overlap, we can choose a set of c activities that may overlap, and order the remaining activities before and after this set. The number of possibilities is $O(d)$, since we are restricted to choosing activity sets that don't violate $\prec_N$.

It is tempting to think that we can find the optimal ordering for a containment pair. By way of underscoring the difficulties in this, we demonstrate that containment pairs actually break the optimal ordering for non-containment pairs of activities. The following simple example shows this:

activity A: $A_{s_{lb}} = 0, A_{e_{ub}} = 4, A_{d_{ub}} = 2, f_A(A_d) = 2d_A$
activity B: $B_{s_{lb}} = 1, B_{e_{ub}} = 5, A_{d_{ub}} = 2, f_B(A_d) = d_B$
activity C: $C_{s_{lb}} = 2, C_{e_{ub}} = 3, A_{d_{ub}} = 1, f_C(A_d) = 10d_C$

The highest value activity C takes on its maximum duration in the optimal solution. This leaves 2 gaps to be filled by lower value activities (A and B). The gaps are as follows: [0 2] and [3 5]. If $A < B$, A gets assigned [1 2] and B gets assigned [3 5]. The value is 2+2=4. If $B < A$, B gets assigned [1 2] and A gets assigned [3 5]. The value is 1+4=5. Thus, $B < A$ is the optimal ordering for A and B conditioned on C being accepted, contradicting the optimal ordering when no containment pairs are present.

Tractable Activity Subset Selection

In the previous section we described ways in which OCS can be made tractable, even when we must order activities. However, these cases do not allow searching for feasible or optimal subsets of activities to schedule. In this section, we will consider tractable cases of OCS even while searching over subsets of activities, as well as ordering and selection of activity duration.

Implicit Activity Rejection

To ensure tractability and still allow activity subset selection, it is enough to ensure every activity can be assigned zero duration during the calculation of the optimal activity duration. This means activities can be *implicitly rejected* after the sequencing phase is completed. This condition can be verified on initialization. Unfortunately, the combined assumption of convex preferences that extend to the case where activities may have zero duration is quite restrictive. For example, in satellite and telescope observation scheduling domains, very small amounts of data may be almost as bad as no data at all. Convex preferences and zero activity duration are not suitable to model such problems.

Tractable Explicit Activity Rejection

In this section we exploit a property of *Valued Constraint Satisfaction Problems* (VCSPs) to guarantee tractability. Consider an ordering O of the variables of a hypergraph H. Define H_i as follows: $H_n = H$, and H_i is constructed by taking all hyper-edges including variable V_i, merging them into one edge, and eliminating V_i. Recall that the *induced width* of ordering O (Dec03) is defined as the maximum size of any edge in any H_i, and the *elimination width* is the minimum induced width over all orderings O. Any class of VCSPs of bounded induced width k is solvable in exponential time in k using bucket elimination (LM03); since k is bounded as N grows, these problems are tractable. Bacchus (private communication) proves that the *tree-width* and the induced width of a graph are equal. Recall that *chordal* graphs have the property that any cycle of length ≥ 4 has an edge between two non-adjacent nodes on the cycle (called a chord). A theorem due to (Gol80) shows that if G is a chordal graph, the tree width = $\omega - 1$ (one less than the maximum clique in the graph). For chordal graphs, various polynomial time algorithms exist for calculating the variable ordering O inducing minimum tree width; for example, see (RL76). If we have a class of VCSPs whose induced constraint graphs are chordal, and the clique number ω grows logarithmically to ensure tractability.

We recall that a tractable, optimal algorithm exists for OCS where A_d has only one possible value a_d, $a_{s_{lb}} = a_{s_{ub}}$ and $a_{e_{lb}} = a_{e_{ub}}$ (SS00); these problems consists only of activity selection. This result is based on the observation that activities can be transformed into a weighted chordal graph; in this graph nodes are activities, edges correspond to overlapping activities (implying only one can be selected), and the problem is to select the maximum weight independent set. In the more general case where activities have a feasibility window, we observe that the feasibility windows still form a chordal graph. However, we now can order activities (within the constraints imposed by the feasibility windows) to resolve resource conflicts.

Based on these observations, we now construct a class of OCS problems that can be formulated as VCSPs with bounded induced width and with bounded variable domain size. Consider the class of OCS with a bounded number of overlapping activity feasibility windows; let that bound be k. For now, we also assume that there are only absolute temporal constraints, i.e. there are no time-point separation constraints. Finally, we assume that activity durations are fixed and activities provide fixed value $f(a_d)$. The problem is to select and order a set of activities $\mathcal{F} \subset \mathcal{A}$ satisfying all of the constraints that maximizes $\sum_{A \in \mathcal{F}} f(a_d)$.

The variables of the VCSP are divided into two categories. The first category consists of the time-point variables A_*. The second category corresponds to *pairs* of activities A, B that overlap; we denote these variables V_{AB}. The values of the variable represent choices that ensure there are no resource or temporal constraint violations for this pair of activities. The possible choices for V_{AB} are:

- $<$ (A and B scheduled, $A < B$)
- $>$ (A and B scheduled, $B < A$)
- A (A scheduled, but B rejected)
- B (B scheduled, but A rejected)
- $\perp$ (Both A and B rejected)

The constraints of this VCSP are also divided into categories. The first category consists of the original temporal constraints in the problem. The second category consists of constraints that ensure that activity acceptance conditions are coordinated between the activity variables that "share" a activity. Consider two variables V_{AB} and V_{AC}. These *order and reject* constraints on V_{AB}, V_{AC} enforces the following:

- $(V_{AB} = \{<, >, A\}) \Leftrightarrow (V_{AC} = \{<, >, A\})$
- $(V_{AB} = \{B, \perp\}) \Leftrightarrow (V_{AC} = \{C, \perp\})$

The third category of constraints propagate the temporal constraints on activity time-points if overlapping activities are ordered. These *order and time* constraints on V_{AB}, A_s, A_e, B_a and B_e enforces the following:

- $(V_{AB} =<) \Leftrightarrow ((b_{s_{lb}} = a_{s_{lb}} + a_d) \wedge (a_{e_{ub}} = b_{e_{ub}} - b_d))$
- $(V_{AB} =>) \Leftrightarrow ((a_{s_{lb}} = a_{s_{lb}} + b_d) \wedge (b_{e_{ub}} = a_{e_{ub}} - a_d))$

The final ingredient is the valuation of each satisfying assignment in the constraints. Any assignment not enumerated above provides value $-\infty$. Satisfying assignments of

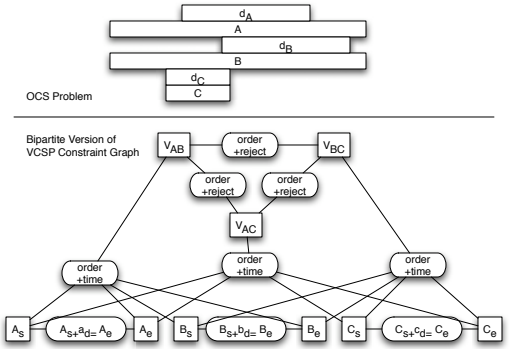

Figure 2: The Valued CSP Representation of the OCS Problem. We show the bipartite representation, using square nodes for variables, round nodes for constraints, with edges indicating a variable is in the scope of a constraint).

the temporal constraints and the order and time constraints provide no value. Let C_A be the number of order and reject constraints involving A. Then $V_{AB} = A$ has value $\frac{f(a_d)}{C_A}$, $V_{AB} = B$ has value $\frac{f(b_d)}{C_B}$, $V_{AB} = \{>, <\}$ has value $\frac{f(a_d)}{C_A} + \frac{f(b_d)}{C_B}$ and $V_{AB} = \perp$ has value zero. The constraint graph of this VCSP is shown in Figure 2.

Theorem 2 *The class of OCS problems with at most k overlapping activity feasibility windows, absolute temporal constraints, fixed duration activities and fixed value $f(a_d)$ is tractable.*

Proof of Theorem 2: Since there are N activities and at most k overlapping activity feasibility windows, there are at most $N\frac{k(k-1)}{2} + 2k$ variables in the VCSP. The domain size of the independent variables V_{AB} of the problem is 5 (the timepoint variables need not be assigned while solving the VCSP). Consider the clique bound ω on the constraint graph of this VCSP. Activity A overlaps with at most $k-1$ other activities, and B overlaps with at most $k-1$ distinct activities, leading to a contribution of at most $2k-2$ due to order and reject constraints. Order and time constraints only contribute 4 more (one per time-point of each activity.) So, any variable $V_{A,B}$ has a bound on degree $\delta(V_{AB})$ of $2k+2$. A time-point variable is involved in at most $k-1$ order and time constraints, and 1 metric temporal constraint. So any time-point variable has a degree bound $\delta(A_*) = k$. Thus, we have $\omega \leq \Delta \leq 2k+2$. □

We can generalize this result to handle the case of a finite number of duration choices for each activity. Assume that $A_d \in \{a_d^1 ... a_d^x\}$, that is, the number of activity durations is finite. The form of the preferences $\sum_i f_i(\sum_{A \in \pi_i} A_d(T))$ is not limited; however, there are only a finite number of discrete preferences that must be encoded. The *order and reject* constraint on V_{AB}, V_{AC} now captures the duration choice:

- $(V_{AB} = \{<, >, A\}) \wedge (A_d = a_d^i) \Leftrightarrow$
  $(V_{AC} = \{<, >, A\}) \wedge (A_d = a_d^i)$
- $(V_{AB} = \{B, \perp\}) \Leftrightarrow (V_{AC} = \{C, \perp\})$

The *order and time* constraint on V_{AB}, A_s, A_e, B_a and B_e also captures the duration choices:

- $(V_{AB} =<) \wedge (A_d = a_d^i) \wedge (B_d = b_d^j) \Leftrightarrow$
  $(b_{s_{lb}} = a_{s_{lb}} + a_d^i) \wedge (a_{e_{ub}} = b_{e_{ub}} - b_d^j)$
- $(V_{AB} =>) \wedge (A_d = a_d^i) \wedge (B_d = b_d^j) \Leftrightarrow$
  $(a_{s_{lb}} = a_{s_{lb}} + b_d^j) \wedge (b_{e_{ub}} = a_{e_{ub}} - a_d^i)$

Since the duration choices do not depend on the number of activities, the increase in size of the constraint representation does not depend on this either; the constraint graph is unchanged, so the above argument on the clique size bound is unchanged. The generalization to finite combinations of start time and duration choice proceeds similarly.

We can also generalize this result to handle the case of a resource capacity $c < k$ and unit resource utilization by activities. When we do this, we must change the VCSP representation slightly. First, we change the set of possible values of the variables:

- $\top_<$ (A and B scheduled and overlap, A starts first)
- $\top_>$ (A and B scheduled and overlap, B starts first)
- $<$ (A and B scheduled and $A < B$)
- $>$ (A and B scheduled and $B < A$)
- A (A scheduled, but B rejected)
- B (B scheduled, but A rejected)
- $\perp$ (Both A and B rejected)

We then add constraints that force at most c of k activities to overlap. This amounts to adding $\begin{pmatrix} k \\ c+1 \end{pmatrix}$ constraints of the form "not all $\top$" to every subset of $c+1$ overlapping activities. Finally, we must add a relation to the order and time constraint capturing the consequences of forcing overlap: $(V_{AB} = \top_<) \Leftrightarrow (A_s \leq B_s \vee A_e \leq B_e)$, and $(V_{AB} = \top_>) \Leftrightarrow (B_s \leq A_s \vee B_e \leq A_e)$. Even in this case, the degree of any variable in the VCSP is bounded above by $\begin{pmatrix} k \\ c+1 \end{pmatrix} + 2k + 2$.

Finally, we can generalize this result to handle metric temporal constraints between timepoints. Essentially, we can permit any time-point variable's degree $\delta(A_*)$ to grow logarithmically in n, and still maintain tractability.

Empirical Results

In this section we describe a series of experiments to investigate prototype algorithms utilizing our results. We generated random problems to test these algorithms and investigate their behavior.

Tractable Choice of Activity Duration

In this section we investigate the case of tractable choice of activity duration. The input to the random generator are the horizon size $Horz$, bounds on feasibility windows size $Wmin$ and $Wmax$, bounds on activity durations $Dmin$ and $Dmax$, bounds on activity value $Vmin$ and $Vmax$. First,

the activity start time, feasibility window size, and maximum duration are selected uniformly from the given range. These characteristics are then modified to ensure that no pair of activities is a containment pair as follows. Whenever a new feasibility window A is generated we check to see if there is already a window B that contains it. If so, we expand A to just after the latest finishing time $B_{e_{ub}}$ of the containing window. If we find A to be contained in an already generated window C, we shrink the new window to line up with the latest finish time $C_{e_{ub}}$ of the contained window. To generate a nondecreasing preference for the activity, we generate the maximum quality and the number of pieces. We then randomly generate a slope for each piece until the maximum is reached.

If we recall the LP formulation, we see that the number of linear constraints is proportional to the number of activities, *and* the number of pieces of the preference function. As such, we generated problems that scale in both of these parameters to investigate how solution times vary. We fixed all other parameter values to $Horz = 4000$, $Wmin = 0$, $Wmax = 20$, $Dmin = 0$, $Dmax = 15$, $Vmin = 0$ and $Vmax = 40$. (Every activity's minimum duration is zero, we only generate the upper bound of the duration.) We ran each parameter setting 100 times and show the mean and variance in CPU time in Figure 3. The implementation makes use of lpsolve 5.1 running on an Apple Macintosh G4. Since the time to solve the LP dominates, we only record this time. We see that solution time increases linearly with the number of activities for each fixed value of the maximum number of preference function pieces. We also see that as the number of activities increases, the rate of solution time increase also goes up, indicating solution time becomes more and more dependent on the linear constraints comprising the preference function.

Tractable Activity Subset Selection

In this section we consider implementation choices that impact performance when solving the tractable case with fixed duration activities over feasibility windows. We limited our empirical study to the case of unary resources and one duration option for activities. Thus, the problem is to select a subset of activities and order them in such a way that the maximum schedule quality is achieved. We also assume each activity has a non-zero value.

Rather than directly use generic VCSP algorithms, we created specialized algorithms that only operate on ordering variables and use STN propagation. The algorithms incrementally build the complete schedule. At each step they construct schedule prefixes, called *sub-schedules*, then prune sub-schedules that are "dominated" by other sub-schedules. They differ in how the sub-schedules are extended, how pruning is impacted by these decisions, and ultimately, how many sub-schedules are stored. Unsurprisingly, the algorithms perform well for different problem regimes, but to our surprise, a simple algorithm performed better than expected. All algorithms essentially traverse the activities from those that can occur "early" to those that can occur "late" in the schedule. These strategies ensure bounded induced width on the VCSP.

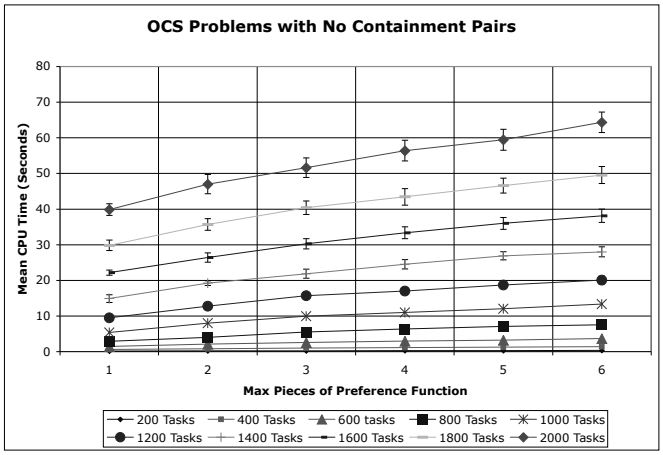

Figure 3: Variation in solution time for the case of no containments.

Domination Rules and Pruning Domination rules for sub-schedules are based on the combination of three parameters: the accumulated value of scheduled activities, the end time of its last activity assuming the sub-schedule is an earliest-time assignment, and the potential to increase the value with activities not yet scheduled. Domination is defined as follows:

Definition 4 *A* sub-schedule S *is defined as an earliest-time assignment of activities that satisfies all resource and temporal constraints. Let* $\mathcal{A}_S$ *be the activities in a schedule* S*. The* extent *of a sub-schedule* $e(S)$ *is* $\max_{A \in \mathcal{A}_S} A_e$ *i.e. the end time of the last activity in* S*. The* value *of* S*,* $v(S)$*, is* $\sum A \in \mathcal{A}_S f(a_d)$*. The* Future Set *of* S*,* $\mathcal{F}(S)$ *is* $A \in \mathcal{A} - \mathcal{A}_S$ *such that imposing the constraint* $e(S) < A$ *does not violate any temporal constraints.*

Consider two sub-schedules S *and* T*.* S dominates T *iff* $e(S) \leq e(T)$*,* $v(S) \geq v(T)$*, and* $\mathcal{F}(T) \subseteq \mathcal{F}(S)$*.*

A common core of the algorithms is the domination process, by which newly generated sub-schedules are pruned by domination. Figure 4 describes the process by which a new sub-schedule S_n is added to a set $\mathcal{S}$ of pairwise-non-dominated sub-schedules. If any element of $\mathcal{S}$ dominates S_n, it is discarded; otherwise, every schedule in $\mathcal{S}$ is checked and any sub-schedule dominated by S_n is discarded, and S_n is added to $\mathcal{S}$.

Simple Solver The simplest solver (called Simple-Solver()) is shown in Figure 5. This solver scans every time instant in the time horizon. For each instant i, we consider all possible extensions of all schedules by an activity that could start at i. We also carry sub-schedules in which no activity starts at i, since we might be able to start a valuable activity after i. Domination rules are used to prune sub-schedules as defined previously.

Dominate($S_n, \mathcal{S}$)

for $T \in \mathcal{S}$
  if T dominates S_n reject S_n and break;
end for
if S_n was not dominated
  for $T \in \mathcal{S}$
    if S_n dominates T
      Remove T from $\mathcal{S}$
    end if
  end for
  add S_n to $\mathcal{S}$
end if

Figure 4: The Dominate() routine.

SimpleSolver()

$\mathcal{S} = \{\emptyset\}$
for each time instant i
  for each sub-schedule $S \in \mathcal{S} | e(S) \leq i$
    for $A \in \mathcal{F}(S) | (a_{s_{lb}} \leq i \leq a_{s_{ub}})$
      Append A to the end of S; call the new schedule S_n
      Dominate($S_n, \mathcal{S}$)
    end for
    Update $e(S) = i + 1$; call the new schedule S_m
    Dominate($S_m, \mathcal{S}$)
    Remove S from $\mathcal{S}$
  end for
end for

Figure 5: The SimpleSolver() algorithm.

FixedPrefix Solver An alternative to building schedules by iterating over time is to iterate over the number of activities in the schedule. The FixedPrefixSolver, shown in Figure 6, builds schedules this way. We control the order in which activities are added to ensure we traverse the activities in an order ensuring bounded induced width. To do so, for each sub-schedule S we first identify the activity A whose addition to the schedule minimally increases the new schedule $e(S_n)$. We then identify any activity $B \in \mathcal{F}(S)$ that could also be added to S, but whose earliest time assignment in S conflicts with A's assignment in S_n. We generate all resulting sub-schedules. Up to the point where additional activities cannot be assigned due to conflict with the end horizon, all newly sub-schedules have the same number of activities. Again, domination is checked as before.

We found that it is possible to dominate some sub-schedules *without* appending activities to the end of them, by observing that there may be a gap before the next feasible activity could start. If so, the extent of this schedule is effectively *larger* than the end time of it's last activity, and this sub-schedule might now be dominated by some other sub-schedule. This can be accomplished by a simple modification of the FixedPrefixSolver() algorithm, as shown in Figure 7.

FixedPrefixSolver()

$\mathcal{S} = \{\emptyset\}$
for each sub-schedule $S \in \mathcal{S}$
  Let $A \in \mathcal{F}(S)$ minimize $e = \max(e(S), a_{s_{lb}}) + a_d)$
  # Append every activity that could conflict with A
  for $B \in \mathcal{F}(S) | ((e(S) \leq b_{s_{ub}}) \wedge (b_{s_{lb}} \leq e))$
    Append B to the end of S; call the new schedule S_n
    Dominate($S_n, \mathcal{S}$)
  end for
  Remove S from $\mathcal{S}$
end for

Figure 6: The FixedPrefixSolver() algorithm.

FixedPrefixSolverEarlyDomination()

$\mathcal{S} = \{\emptyset\}$
for each sub-schedule $S \in \mathcal{S}$
  Let $A \in \mathcal{F}(S)$ minimize $e = \max(e(S), a_{s_{lb}}) + a_d)$
  # Append every activity that could conflict with A
  for $B \in \mathcal{F}(S) | ((e(S) \leq b_{s_{ub}}) \wedge (b_{s_{lb}} \leq e))$
    if $\min_{A \in \mathcal{F}(S)} a_{s_{lb}} > i$
      Update $e(S) = i$; call the new schedule S_m
      Dominate($S_m, \mathcal{S}$)
    else
      Append B to the end of S; call the new schedule S_n
      Dominate($S_n, \mathcal{S}$)
    end else
  end for
  Remove S from $\mathcal{S}$
end for

Figure 7: The FixedPrefixSolverEarlyDomination() algorithm.

Problem Generation

Again we used randomly generated problems to investigate the properties of these algorithms. In this case we do not need to control for the existence of containment pairs, but must control for the number of overlapping activity feasibility windows. We also do not need to generate a preference function, but instead must choose a single value for each activity. In order to explore the range of behaviors we varied the number of overlapping activities $k = \{2, 4, 6\}$ for two horizons, one short and one long. For $Horz = 3000, Wmin = 3, Wmax = 10, Dmin = 1, Dmin = 8, Vmin = 5, Vmax = 72$ we set the number of activities $n = 100, 200, 300, 400, 500, 600$. For $Horz = 20000, Wmin = 5, Wmax = 10, Dmin = 5, Dmin = 10, Vmin = 25, Vmax = 700$ we varied the number of activities $n = 100, 500, 1000, 1500, 2000$. To ensure there was

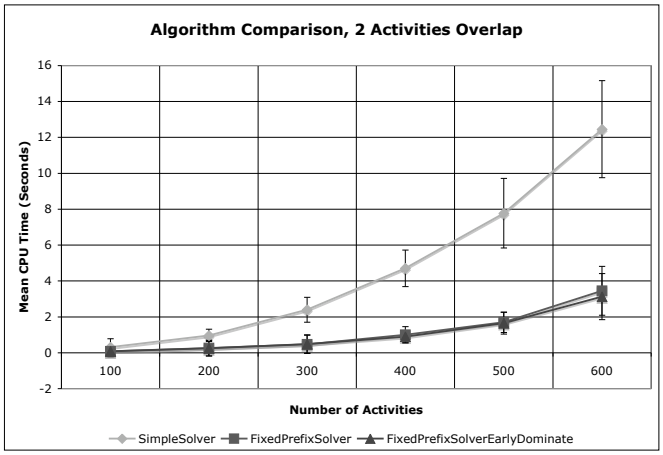

Figure 8: Variation in solution time for the case $Horz = 3000, k = 2$.

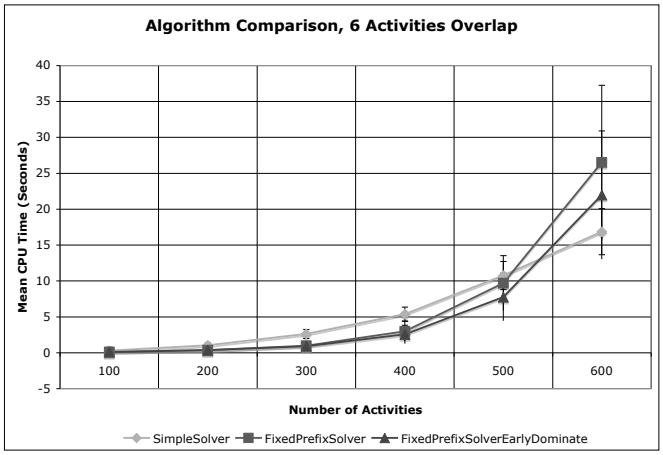

Figure 9: Variation in solution time for the case $Horz = 3000, k = 6$.

a positive correlation between activity duration and activity value, the value of an activity was set to the product of its duration and another randomly chosen value. We generated 100 instances per parameter setting, and show the mean and variance in CPU time.

The Implications of Implementation The SimpleSolver() was implemented as a sanity check for the correctness of the other, more sophisticated, algorithm. Given its code simplicity, it is easy to verify that it is correctly implemented. The SimpleSolver() appears impractical for several reasons. Scanning each timepoint in the horizon is clearly inefficient for long horizons, and for real-valued time it is impossible. Furthermore, we depend on assigning new activity start times explicitly, rather than posting ordering constraints. We expected the FixPrefixSolver() versions to ultimately dominate the SimpleSolver(), and we also expected the early domination check to be a strict improvement. Figure 8 shows a typical example of this. For the case of $k = 2$, iteration over the horizon is increasingly dominated by the other approaches as the number of activities increases

To our surprise, however, the FixPrefixSolver() did not outperform SimpleSolver() in all cases. This is shown in Figure 9, where $k = 6$. SimpleSolver() begins to outperform the other two algorithms as more and more activities are added. Comparing the performances of SimpleSolver() and FixedPrefixSolver(), SimpleSolver() seems to have better performance on problems that are dense with many competing activities. This is because simply traversing the horizon pays off when some new activity could start at almost each timepoint on the horizon. For completeness, we show the same data for $Horz = 20000$ in Figures 10 and 11 and see a similar trend.

One possible explanation for the difference in performance between the algorithms is the potential for variation in the extent of the sub-schedules. The SimpleSolver() implicitly controls the extent $e(S)$ of the sub-schedules $S \in \mathcal{S}$ by only adding activities that can start at i (the current timepoint). If the maximum duration is d_u then the extents of sub-schedules can differ by at most d_u (because of how the extents of schedules are adjusted when no activity is appended). By contrast, there is no such control of the extent in the FixedPrefixSolver(). This divergence in extent may lead the FixedPrefixSolver() to generate many undominated schedules.

FixedPrefixSolverEarlyDominate() is guaranteed to outperform FixPrefixSolver() in terms of *maximum number of sub-scheduels carried* (earlier detection of dominations) in all cases. Generally, one would expect the runtime performance to follow this trend; we found this to be the case in our experiments. The trend is most clearly seen in Figure 11; we see that adding the early domination detection prevents the generation of large numbers of schedules as the number of activities increases, leading to significant time savings.

Related Work

Wang and Smith (WS05) address a scheduling problem similar to the OCS. Their work differs from ours in several ways. The flexibility of their activity durations are defined by a minimum value but no maximum value (ours has both). The activity feasibility window extends from a release time to the end of the scheduling horizon (ours extend to a due date that is any where between the release and end of the horizon). They allow no rejection of activities (we handle rejections). Their quality profile associated with each activity is linear (ours is piece-wise linear) and depends only on activity duration. The simple case of this problem, where

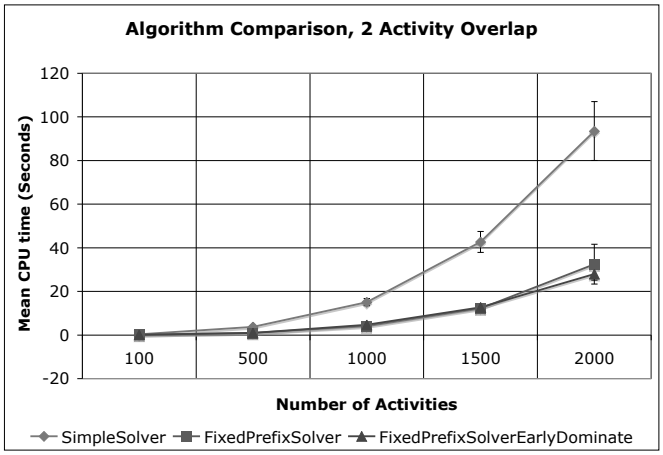

Figure 10: Variation in solution time for the case $Horz = 20000, k = 2$.

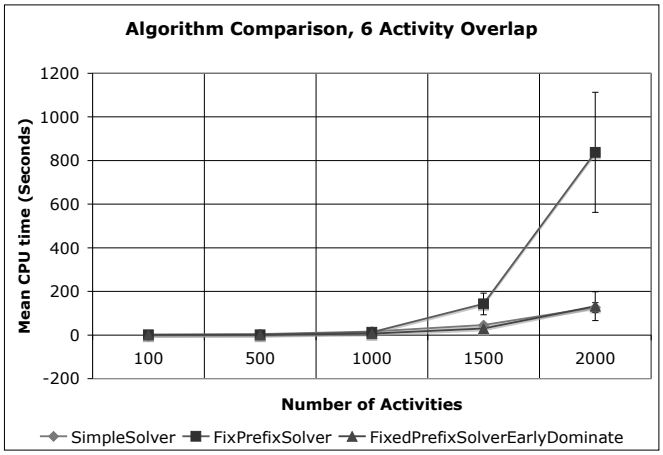

Figure 11: Variation in solution time for the case $Horz = 20000, k = 6$.

resource capacity is 1, reduces to Linear Programming. We also observe that, given that all windows extend all the way to the end of the horizon, inequalities ensure there is no containment pairs, which implies there is an easy to determine optimal ordering for the activities.

Bartlett et al. (BFH$^+$05) discuss algorithms for the $\mathcal{NP}$-complete Temporal Knapsack Problem (TKP). Peinter et al. (PMP05) describe algorithms for the even more general Disjunctive Temporal Constraints with Preferences problem. These problems are quite general, allowing for multi-capacity resources, and complex preferences for satisfying activities. We observe that our tractable OCS classes map naturally into tractable classes of these problems. We point out that the TKP formulation stipulates integer time, which leads to Integer Linear Programming (ILP) formulations that have one constraint per time-point. We make no such assumptions in general about OCS, although some of our tractability results depend on discrete activity duration choices.

As previously discussed, Morris et al. (MMK$^+$04) introduce a tractable problem that combines Simple Temporal Networks with convex preferences on inter-event distances. Kumar (Kum04) shows that STPPs where preferences are piecewise constant are tractable. This is distinguished from the results due to (MMK$^+$04) by the fact that the preference function need not be convex. The solution methodology used is weighted anti-chains, which may be more efficient than the more general Linear Programming techniques used in our work and by (MMK$^+$04).

Numerous authors, including (SS00), have considered tractable combinatorial auction problems. We observe that our tractable classes provide numerous ways to extend such results, even in the case where the auction combines combinatorial choices (which bids to award) with arbitrarily divisible goods (how much time to award).

Conclusions and Future Work

We have described the Optimal Competitive Scheduling (OCS) problem as one of selecting, ordering and choosing activity duration, constrained by temporal and resource constraints, in order to optimize a global objective. We have described tractable OCS problems for a variety of quite reasonable, but restricted, assumptions. Our tractable classes both extend the state of the art in temporal problems with preferences, and make use of existing work on solving tractable VCSPs. Figure 12 places these results in context with existing work in this area. We have also described empirical results showing that problem characteristics impact algorithm performance. When selecting activity duration only, the LP formulation is affected by the number of activities and the characteristics of the preference function. When also selecting subsets of activities and ordering, the "density" of the schedule dictates which of two algorithms performs best. We have also shown that, even these tractable problems, care must be taken to ensure that all available information is exploited to achieve good performance.

While these results close the gap between tractable and intractable optimization problems, there are still open questions to be resolved. The tractable case of activity duration requires that no pair of feasibility windows are containment pairs. Relaxing this assumption may be possible.

The most general case of tractable activity subset selection requires discrete choices for durations (as well as bounded induced width of the underlying constraint graph). Extending this case to handle continuous durations with a reasonable assumption on the form of the preference function (e.g. piecewise linear) would also lead to a significant

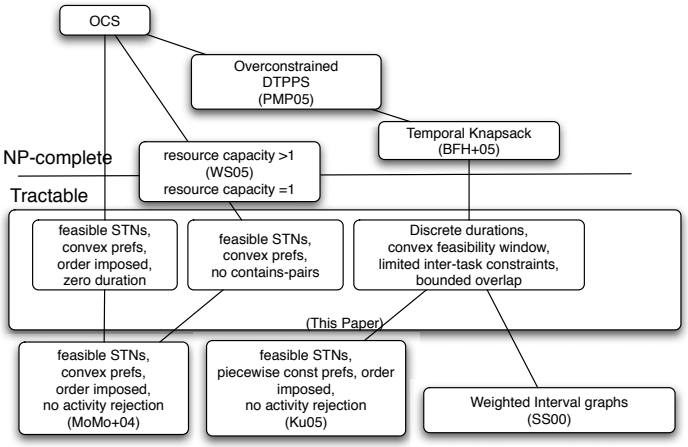

Figure 12: Theoretical Results in this paper, placed in context of previous work.

advance in the theory of tractable VCSPs.

None of the work in this paper or related work addresses the case of preference problems with setup actions. These actions consume time that might otherwise be used for valued activities. The most general case of sequence dependent setup actions might influence otherwise optimal schedules.

A final avenue of interesting work is concerned with *approximate* solutions. In particular, this may be a promising way to handle non-convex preference functions; these functions can be bounded above or below by convex preferences. The open issue is how to reason about the quality of solutions in such cases.

References

M. Bartlett, A. Frisch, Y. Hamadi, I Miguel, S. Tarim, and C. Unsworth. The temporal knapsack problem and its solution. In *Proceedings of the* 2^d *International Conference on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, pages 34 – 48, 2005.

J. Bresina. Heuristic-biased stochastic sampling. In *Proceedings of the 13th National Conference on Artificial Intelligence*, 1996.

P. Brücker. *Scheduling Algorithms*. Springer, 1998.

L. Barbulescu, D. Whitley, and A. Howe. Leap before you look: An effective strategy in an oversubscribed scheduling problem. In *Proceedings of the* 19^{th} *National Conference on Artificial Intelligence*, 2004.

R. Dechter. *Constraint Processing*. Morgan Kauffmann, 2003.

R. Dechter, I. Meiri, and J. Pearl. Temporal constraint networks. *Artificial Intelligence*, 49:61–94, 1991.

J. Frank and E. Kürklü. High and dry: Trading water vapor, fuel and observing time for airborne infrared astronomy. In *Proceedings of the* 25^{th} *International Geoscience and Remote Sensing Symposium*, 2005.

M. Golumbic. *Algorithmic Graph Theory and Perfect Graphs*. Academic Press, 1980.

T. K. Satish Kumar. A polynomial time algorithm for simple temporal prefrerence problems with piecewise constant domain preference functions. In *Proceedings of the* 19^{th} *National Conference on Artificial Intelligence*, pages 67–72, 2004.

J. Larrosa and P. Meseguer. Boosting search with variable elimination in constraint optimization and constraint satisfaction problems. *Journal of Constraints*, 8(3):303–326, 2003.

P. Morris, R. Morris, L. Khatib, S. Ramakrishnan, and A. Bachmann. Strategies for global optimization of temporal preferences. In *Proceedings of the* 10^{th} *International Conference on the Principles and Practices of Constraint Programming*, pages 408 –422, 2004.

B. Peinter, M. D. Moffitt, and M. E. Pollack. Solving overconstrained disjunctive temporal problems with preferences. In *Proceedings of the* 15^{th} *International Conference on Automated Planning and Scheduling*, 2005.

D. J. Rose and R. E. Tarjan G. S. Lueker. Algorithmic aspects of vertex elimination on graphs. *SIAM J. Computing*, 5:266–283, 1976.

H. D. Sherali and F. Nordai. NP-hard, capacitated, balanced p-median problems on a chain graph. *Mathematics of Operations Research*, 13(1):32–49, 1988.

T. Sandholm and S. Suri. Improved algorithms for optimal winner determination in combinatorial auctions and generalizations. In *Proceedings of the* 17^{th} *National Conference on Artificial Intelligence*, pages 90–96, 2000.

W. Wolfe and S. Sorensen. Three scheduling algorithms applied to the earth observing domain. *Management Science*, 46(1), 2000.

X. Wang and S. Smith. Retaining flexibility to maximize quality: When the scheduler has the right to decide activity duration. In *Proceedings of the International Conference on Automated Planning and Scheduling*, 2005.