AURORA DEL CAMP – SPECIFICATION

JULIAN PFEIFLE

This document outlines some of the issues that arise in the implementation of Gilad's proposal [3].

Given the availability of powerful free and open source solvers for integer programs such as CBC [4], it seems natural to pursue an integer programming formulation. Of course, free solvers are not as good as the best commercial ones, but the most recent benchmarks [5] indicate that CBC is reasonably competitive; more precisely, it's the most competitive among all solvers that have an open source license (in the case of CBC, the "Eclipse Public License") that permits Gilad to use it commercially without paying any license fees.

1. Event-based modeling

1.1. **Activities to be considered.** In the parlance of [1], we want to schedule a set of *activities* subject to certain constraints. In our setup, each activity is associated to a certain specific part of the available fields. We refer to these parts or areas as *lots*, and collect them into a set L. We allow them to have different sizes.

The activities to be carried out in or on these lots come in two types: $A = A_c \cup A_s$, where the activities in A_c only affect the field and are thus *"common"* to all crops, and the activities A_s are *"specific"* to each crop. Moreover, we allow each activity to occur multiple times in different time windows, to take into account repeated sowing, harvesting, etc. We keep track of the repetitions by remembering, for each activity, a *valid start time window* $\omega_{\text{start}} = [w_1, w_2]$ of weeks in which it may begin. This has the added advantage that certain characteristics of the activity, such as the duration or the yield, may depend on the start time. For example, *"planting tomatoes in week 34 or 35"* and *"planting tomatoes in week 48 or 49"* will be separate activities, each with distinct durations and yield. The optimization process will determine whether to carry out none, one or both of these activities. As an abstract notation, we write W for the set of all useful start time windows $\omega = \omega_{\text{start}}$, i.e., all intervals contained in the planning horizon of the problem that are eligible for starting a task.

Gilad breaks the two types of activities down as follows:

Activities common to all crops: ti for tilling, rv for rotovating, gm for green manure planting, ft for fertilizing, bb for bed building, si for setting up irrigation, sr for setting rows, we for weeding. Since each of these activities can occur on each lot, and be repeated, we set

$$A_c = \{ti,\ rv,\ gm,\ ft,\ bb,\ si,\ sr,\ we\} \times L \times W.$$

A typical activity in A_c is therefore (ti, ℓ, ω) for some $\ell \in L$ and $\omega \in W$, which we write as $ti_{\ell,\omega}$ and take to mean "tilling the lot ℓ starting in the time window ω".

Activities specific to a crop: by for buying seeds, ss for soaking seeds, cs for cutting or separating cloned seeds, gc for false germination and cleaning, pl for planting[1], gr for growing, fu for

Date: Version of November 29, 2011.

[1]We consider transplanting and planting to be the same process.

fumigating, th for thinning, tr for trimming, co for covering, ha for harvesting. We assemble these into

$$A_s = \{by,\ ss,\ cs,\ gc,\ pl,\ fu,\ th,\ tr,\ co,\ ha\} \times C \times L \times W,$$

where C is the set of crops. A typical activity in A_s is therefore $(ha, c, \ell, \omega) = ha_{c,\ell,\omega}$, which means "harvesting the crop $c \in C$ in the lot $\ell \in L$, starting during the time window ω".

The estimates $|L| = 30$, $|R| = 5$, $|C| = 40$ yield an upper bound of

$$8 \times 30 \times 5 + 10 \times 40 \times 30 \times 5 \ = \ 61\,200$$

activities in the model, which is a very manageable figure for commercial solvers, and should also not present unsurmountable problems to free solvers such as CBC.

1.2. **Precedence constraints.** We record the precedence constraints between activities in a directed acyclic graph H. Thus, $t_1 \longrightarrow t_2$ (also written $t_1 < t_2$) is a directed edge in H if t_1 must be completed before t_2 can start.

1.3. **Chains.** We further group activities into *chains*, or activities that must go together. For example,

$$\chi \ = \ \big(pl_{c,\ell,[w_0,w_1]},\ we_{\ell,[w_2,w_3]},\ we_{\ell,[w_3,w_4]},\ gr_{\ell,[w_1,w_4]},\ ha_{c,\ell,[w_4,w_5]}\big)$$

with $w_0 < w_1 < w_2 < w_3 < w_4 < w_5$ is a chain consisting of *planting*, *weeding* (twice), and *harvesting* a crop c in a lot ℓ that needs $w_4 - w_1$ weeks to *grow*. The directed graph H records that in this particular chain, *growing* and *weeding* may be carried out in parallel, but both these activities come after *planting* and before *harvesting*. In general, the set of all chains is denoted by K, and if the first activity in a chain χ is executed, all the others must be executed too.

1.4. **Overview of the model.** We separate the *scheduling* part of the problem, in which the appropriate sequencing of events is determined, from the *allocation* part, in which activities and crops are assigned their proper place in the field. First, the optimal sequencing of events is determined in a way that respects the available field space and work force using an *event-based* formulation (see below); allocation is relegated to a second step. Separating the two phases makes it easier to formulate each one, and presumably makes them (and thus, the whole problem) easier to solve.

In general terms, we will plan over several years. In the final program, there will be an interface to put new activities into a *task queue* (a set of events that is not scheduled yet), and an interface to record the actual progress of activities. This act of recording the real start and end time of activities, environmental changes, changes in the available work force, etc., sets certain variables in the problem formulation to fixed, known values, and allows Gilad to frequently update the solution of the optimization problem and dynamically take into account the latest developments.

2. Scheduling

Before we start with the model, we remark that the setup in [1] allows for incorporating constraints on resources. The main *non-renewable* resource that comes to mind is "time", while some *renewable* ones are "space in the fields", "money" and "gasoline". Money is put back into the system by selling crops, and field space by tilling the remains of the crop. Gasoline is renewable, but costs money.

Of these resources, the only one we need to model explicitly at this point is field space. The resource "time" is handled by the processing times introduced below. I do not have sufficient understanding at the moment to know whether use of gasoline can be optimized or not. Money will be handled by the objective function.

2.1. **Variables.** We now proceed to model our problem. To the activities in A we associate a set E of *events*, which consist of the acts of starting and finishing each activity in A; thus, $|E| = 2|A| =: n$. In consequence, we may consider $E = \{1, 2, \dots, 2|A| = n\}$ to be totally ordered. Following [1], we introduce the following variables and data:

(1) A set of binary decision variables
$$Z \;=\; \big\{z_{a,e} : a \in A, e \in E\big\},$$
where each $z_{a,e} = 1$ if and only if activity a starts at event e or is still in execution at event e.

(2) A set of continuous variables that indicate the starting time of each event:
$$T \;=\; \big\{t_e : e \in E\big\}$$

(3) The set of continuous *processing times* for each activity:
$$\{p_a : a \in A\}$$
These must be estimated from experience, and are necessarily averages. Moreover, the processing times will often depend on when each activity is started.

Another issue is to take into account possible variations in the number of hands available on the farm. One way to do this could be that the processing times stored in the database are the times it would take *one* worker to complete each task. In the user interface, there will be a field to enter an estimate for the number of available helpers in each week of the year. Each time that a new instance of the optimization problem is generated, the processing times for the jobs in each chain in the database are divided by the number of available workers before being written to the optimization file. More sophisticated setups are of course possible, and we'll probably implement one of them before we're done.

(4) The total field space available, S, and the space taken up by each activity,
$$\{s_a : a \in A\}$$

(5) The yield $y_{c,w}$ of each crop $c \in C$, depending on the week $w \in W$ in which it is planted.

2.2. **Constraints internal to the model.** We now adapt the individual constraints from [1]:

Not all activities have to execute: We do *not* incorporate a constraint $\sum_{e \in E} z_{a,e} \;=\; 1$ for all $a \in A$, because we do not wish to require all activities to execute. This leaves Gilad margin to queue activities such as "sowing and harvesting beans for the fifth time" that may or may not take place, but where the decision on them having take place or not is an outcome of the optimization process and not an a-priori input to the problem formulation.

Initially, it therefore seems to be a good idea to queue more repetitions of activities than could reasonably be undertaken, so that the optimal number of repetitions may be learned from the optimization process. We'll see how this works out.

Activities in a chain must go together: The fact that either all or none of the activites in a given chain $\chi = (a_1, a_2, \dots, a_r)$ must be executed is expressed via the equality of the corresponding 0/1-variables:
$$a_1 \;=\; a_2 \;=\; \dots \;=\; a_r. \tag{2.1}$$

Setting the starting time: Instead of using $t_0 = 0$, we set
$$t_0 \;=\; w_0, \tag{2.2}$$
where w_0 indexes the week of the year where optimization starts. In general, using weeks as units for time seems to be a good idea.

Ordering the execution starts:

$$t_{e+1} \;\geq\; t_e \qquad \text{for all } e \in E \smallsetminus \{n\} \tag{2.3}$$

Execution start constraints: Relations that implement start time windows:

$$w_{1,e} \;\leq\; t_e \;\leq\; w_{2,e} \qquad \text{for all } e \in E \tag{2.4}$$

Duration constraints:

$$t_f \;\geq\; t_e + \big((z_{a,e} - z_{a,e-1}) - (z_{a,f} - z_{a,f-1}) - 1\big)p_a \qquad \text{for all } f > e \in E,\ a \in A \tag{2.5}$$

As discussed in [1], these constraints ensure that, if activity a starts at event e and ends at f, then the time difference between f and e is at least the processing time of a: $t_f \geq t_e + p_a$.

Contiguity constraints: As proved in [2, Proposition 1], the constraints

$$\sum_{i=1}^{e-1} z_{a,i} \;\leq\; e\,\big(1 - (z_{a,e} - z_{a,e-1})\big) \qquad \text{for all } e \in E \smallsetminus \{1\},\ a \in A \tag{2.6}$$

$$\sum_{i=e}^{n} z_{a,i} \;\leq\; (n-e)\,\big(1 + (z_{a,e} - z_{a,e-1})\big) \qquad \text{for all } e \in E \smallsetminus \{1\},\ a \in A \tag{2.7}$$

ensure *non-preemption*, i.e., the events after which the activity a is being processed are adjacent.

Precedence constraints: The implication $(z_{a,e} = 1) \implies \big(\sum_{i=1}^{e} z_{b,i} = 0\big)$ that describes the directed edge $a \longrightarrow b \in H$ for each event e is modeled by the linear inequality

$$z_{a,e} + \sum_{i=1}^{e} z_{b,i} \;\leq\; 1 + (1 - z_{a,e})e \qquad \text{for all } e \in E,\ a \longrightarrow b \in H \tag{2.8}$$

Space constraints: At any given time, all current activities must take up no more than the entire available space:

$$\sum_{e=1}^{n} s_a\, z_{a,e} \;\leq\; S \qquad \text{for all } a \in A \tag{2.9}$$

2.3. **External constraints.** We may also incorporate constraints that come from the way crops behave. For example, Gilad gives the example that *"A head of lettuce planted in summer must be harvested the week after it is planted, but if it is planted in winter, it can stay in the ground for up to two months."* This can be modeled via a sequence of chains

$$\begin{aligned}
\chi_{\text{lettuce, \#42, 25}} &= \big(pl_{\text{lettuce, \#42, 25}},\ we_{\text{lettuce, \#42, [25,26]}},\ gr_{\text{lettuce, \#42, [25,26]}},\ ha_{\text{lettuce, \#42, [25,26]}}\big),\\
\chi_{\text{lettuce, \#42, 26}} &= \big(pl_{\text{lettuce, \#42, 26}},\ we_{\text{lettuce, \#42, [26,27]}},\ gr_{\text{lettuce, \#42, [26,27]}},\ ha_{\text{lettuce, \#42, [26,27]}}\big),\\
&\dots\\
\chi_{\text{lettuce, \#42, 35}} &= \big(pl_{\text{lettuce, \#42, 35}},\ we_{\text{lettuce, \#42, [35,36]}},\ gr_{\text{lettuce, \#42, [35,36]}},\ ha_{\text{lettuce, \#42, [35,36]}}\big)
\end{aligned}$$

that say that lettuces *planted* from the middle of June (week 25) to the last week of August (week 35) in a certain lot (#42 in this example) must be *weeded* exactly once, need one week to *grow*, and must be *harvested* one week after planting; and a sequence of chains

$$
\begin{aligned}
\chi_{\text{lettuce, \#42, 47}} &= \big(pl_{\text{lettuce, \#42, 47}},\ we_{\text{lettuce, \#42, [47,49]}},\ we_{\text{lettuce, \#42, [50,52]}},\ we_{\text{lettuce, \#42, [53,55]}},\\
&\qquad gr_{\text{lettuce, \#42, [47,55]}},\ ha_{\text{lettuce, \#42, [47,55]}}\ \big),\\
\chi_{\text{lettuce, \#42, 48}} &= \big(pl_{\text{lettuce, \#42, 48}},\ we_{\text{lettuce, \#42, [48,50]}},\ we_{\text{lettuce, \#42, [51,53]}},\ we_{\text{lettuce, \#42, [54,56]}},\\
&\qquad gr_{\text{lettuce, \#42, [48,56]}},\ ha_{\text{lettuce, \#42, [48,56]}}\ \big),\\
&\dots\\
\chi_{\text{lettuce, \#42, 56}} &= \big(pl_{\text{lettuce, \#42, 56}},\ we_{\text{lettuce, \#42, [56,58]}},\ we_{\text{lettuce, \#42, [59,61]}},\ we_{\text{lettuce, \#42, [62,64]}},\\
&\qquad gr_{\text{lettuce, \#42, [56,64]}},\ ha_{\text{lettuce, \#42, [56,64]}}\ \big)
\end{aligned}
$$

that express that if a head of lettuce is planted between the third week of November (week 47) and the last week of January (week 56), up to eight weeks can pass before it must be harvested ($56 + 8 = 64$); in exchange for that, we must weed three times.

2.4. Objective function.

The objective function we want to maximize is thus

$$f = \sum_{c\in C, y\in Y} y_{c,w} \sum_{a\in A} x_{c,w,a}$$

3. Allocation

4. Implementing and optimizing the problem formulation

4.1. Implementation.

We will probably use either PHP or Python to generate the input file to the optimizer from a database of constraints and other data.

4.2. Optimizations.

As remarked in [1], there is no need to create events for the ending of the last activities.

5. Server-side technology

Gilad's intention is to make the program available on a server. That's fine, except that we need to be able to install c++ and cbc on such a server.

References

[1] C. Artigues, O. Koné, P. Lopez, and M. Mongeau, *Comparison of mixed integer linear programming models for the resource-constrained project scheduling problem with consumption and production of resources*. `http://www.math.univ-toulouse.fr/~mongeau/MILP-RCPSP-CPR-submitted.pdf`, February 2011.

[2] ———, *Event-based MILP models for resource-constrained project scheduling problems*, Computers & Operations Research, 38 (2011), pp. 3–13.

[3] G. Buzi, *Aurora Del Camp Crop Planner — a small farm plans big*, November 2011.

[4] *Cbc (COIN-OR branch and cut), an open-source mixed integer programming solver written in C++*. `https://projects.coin-or.org/Cbc`.

[5] H. D. Mittelmann, *Performance of optimization software — an update*. `http://plato.asu.edu/talks/mittelmann_bench.pdf`, November 2011.