1 Alterations and Improvements

Coarsening boundary Several data used in our models are results of various approximations, and hence the boundary of the feasible area is not exact. [It it therefore ok] to slightly change this boundary during the projection. More specifically, during the redundancy removal phase we also remove inequalities which are *"almost redundant"*, meaning that their [maximal value] is within a given relative ϵ from its rhs. That is, $c : \mathbf{a} \cdot \mathbf{x} \leq b$ is almost redundant if

$$\max \mathbf{a} \cdot \mathbf{x} \text{ subject to } S \setminus \{c\} \text{ is less or equal to } b + \epsilon \cdot |b|. \tag{1}$$

See Figure... If $b = 0$, we instead require the maximum to be a small ϵ'.

[A method for coarsening the boundary of the feasible area that relies on removing almost redundant inequalities are used in [**?**] and [**?**] too, though the approach is different.]

Parallel redundancy checks We will use several threads to remove redundancy in the system S. This is done using a manager to keep track of $k > 0$ individual workers who do the actual redundancy checking. The manager assigns a different inequality to each of the workers, who in parallel each check if their own inequality is [real strict] in their own copy of S. When it is done with its task, each worker then reports back to the manager (in a thread safe manner) whether their inequality was [pure strict] redundant, almost redundant or non-redundant. The manager then assigns the worker a new inequality to check, and if the reported inequality was redundant, the other workers are notified such that they can remove this inequality from their own copy next time they start checking a new inequality. The manager collects/remembers a set of [purely strict] redundant inequalities and the ones that were almost redundant. When all has been checked, the redundant inequalities are removed.

For this step it is important that only [true strict] inequalities are removed and not almost redundant inequalities. Otherwise, in a situation as in Figure **??**, both i_1 and i_2 could simultaneously be found almost redundant by two different workers, causing them both to be eliminated. For similar reasons it is also required that S does not contain inequalities define the same halfspace; these are therefore removed prior to the redundancy removal [men hvis jeg har true strict.. then this would not be necessary].

After the parallel redundancy check, one worker goes through all the almost redundant inequalities one by one to and removes the ones that are still almost redundant. For the situation in **??**, both inequalities would be checked again, but only the one examined first would be removed.

Condition numbers [but isn't this more of an implementation issue?] The checkers also check and report on inequalities for which the optimization of its left hand side resulted in a bad condition number (also known as κ-value). This number is a measure for how much (small) differences in the input value effects the value of output; see e.g. [**?**]. In general, the larger this number is, the less we can trust the result, and we have chosen a threshold $\mathcal{K}$ that gives the limit for when we can trust the result. As the system gets smaller due to the removal of redundant inequalities, inequalities might be reexamined with a lower κ-value as a result, and therefore we also re-examine the inequalities that resulted in a bad κ-value when they were checked by the parallel redundancy checkers. Finally we also reexamine inequalities, for which the optimization software was not able to solve the optimization of its left-hand-side. [And some more stuff if many bad kappa values but I dont think we have that here, so not necessary]