Lagrangian Decomposition via sub-problem Search

Geoffrey Chu, Graeme Gange, and Peter J. Stuckey

National ICT Australia, Victoria Laboratory,
Department of Computing and Information Systems,
University of Melbourne, Australia
{geoffrey.chu,gkgange,pstuckey}@unimelb.edu.au

Abstract. One of the critical issues that affect the efficiency of branch and bound algorithms in Constraint Programming is how strong a bound on the objective function can be inferred at each search node. The stronger the bound that can be inferred, the earlier failed subtrees can be detected, leading to an exponentially smaller search tree. Normal CP solvers are only capable of inferring a bound on the objective function via propagating the problem constraints. Unfortunately, for many problem classes, this does not yield a very strong bound. Recently, Lagrangian decomposition methods have been adapted and applied to Constraint Programming in order to yield stronger bounds on the objective function. While these methods yield some success, they are somewhat limited in the types of problems they can be effectively applied to. In particular, the set of constraints has to be divided into subsets such that each subset can be solved efficiently via a specialized propagator, e.g., consists of a knapsack problem, or a cost-MDD problem. For many more practical problem classes, such a division of constraints is simply not possible and thus those methods cannot be applied. In this paper, we propose a Lagrangian decomposition method where the sub-problems are solved via search rather than through a specialized propagator. This has the benefit that the method can be applied to a much wider range of problems. We present experiments to show the effectiveness of our method.

1 Introduction

Constraint Programming (CP) approaches are state-of-the-art for solving many combinatorial optimization problems using a branch and bound approach. But a critical issue that effects the the efficiency of branch and bound algorithms in CP is how strong a bound on the objective function can be inferred at each search node. The stronger the bound that can be inferred, the earlier failed subtrees can be detected, leading to an exponentially smaller search tree. Normal CP solvers are only capable of inferring a bound on the objective function via propagating the problem constraints. Unfortunately, for many problem classes, this does not yield a very strong bound. Indeed for this reason Mixed Integer Programming (MIP) approaches are preferable to CP for solving many forms of

combinatorial optimization problem – they have strong bounds derived from the linear programming relaxation of the problem.

Recently, Lagrangian decomposition methods have been adapted and applied to Constraint Programming in order to yield stronger bounds on the objective function [1, 2]. Lagrangian decomposition allows us to break an optimization problem down into parts that act independently, analogous to the way that CP solvers treat different constraints. For Lagrangian decomposition each of these parts is a constrained optimization problem, and together they generate bounds on the objective, which can be much stronger than simply propagating the objective constraint. The use of Lagrangian decomposition in CP is an exciting development, exactly because it gives us scope for the same powerful heterogeneous approach to constraint satisfaction used in CP, through separate communicating propagators, to be used for constraint optimization, through separate communicating optimizers.

While the introduction of Lagrangian decomposition to CP is an important development, current methods are quite limited in the types of problems they can be applied to. In particular, the set of constraints have to be divided into subsets such that each subset can be solved efficiently via a specialized propagator/optimizer. Examples considered so far restrict the sub-problems to be either knapsack problems [2], or problems specified by a cost-MDD constraint [1] (although this can theoretically express any COP).

For many problem classes, such a division of constraints is simply not possible and thus those methods cannot be applied. In this paper, we propose a Lagrangian relaxation method where the sub-problems are solved via search rather than through a specialized propagator. This has the benefit that the method can be applied to a much wider range of problems. We present experiments to show the effectiveness of our method.

The contributions of this paper are:

- a generic approach to Lagrangian decomposition, applicable to any problem with a linear objective.
- a meta-search based approach to solving Lagrangian decomposed sub-problems, in order to improve bounds on the objective.
- experiments showing that the search based approach to Lagrangian decomposition can be highly effective.

2 Background and Definitions

2.1 Constraint Programming with Lazy Clause Generation

Let $\mathcal{V}$ be a set of (integer) variables (we will treat Boolean variables as 0-1 integers).

A *valuation*, θ, is a mapping of variables to values, denoted $\{x_1 \mapsto d_1, \ldots, x_n \mapsto d_n\}$. Define $vars(\theta) = \{x_1, \ldots, x_n\}$. We can apply a valuation to a variable $\theta(x_i)$ to return the value d_i, and extend application of valuations θ to arbitrary expressions involving $vars(\theta)$ in the obvious way.

A *primitive constraint*, c, is a set of valuations over a set of variables $vars(c)$. A valuation θ is a *solution* of c if $\{x \mapsto \theta(x) \mid x \in vars(c)\} \in c$. A *constraint* C is a conjunction of primitive constraints, which we often treat as a set. A valuation θ is a solution of constraint C if it is a solution for each $c \in C$. We write $C_1 \models C_2$ if every solution of C_1 is a solution of c_2. We extend this notation to valuations, writing $\theta \models C$ if $\bigwedge_{i=1}^{n} x_i = d_i \models C$ where $\theta = \{x_1 \mapsto d_1, \ldots, x_n \mapsto d_n\}$.

A *literal* is a unary constraint (we can restrict to the forms $x = d, x \neq d, x \geq d, x \leq d$), or *false*. A *domain* D is a conjunction of literals over $vars(D)$. D is a *false domain* if it has no solutions. We use notation $D(x) = \{\theta(x) \mid \theta \text{ is a solution of } D\}$. We use *range notation* $[\, l \,..\, u \,] = \{d \mid l \leq d \leq u\}$. We can map a valuation θ to a domain $D_0 = \wedge_{x \in vars(D)} x = \theta(x)$.

A *propagator* $p(c)$ for constraint c is an inference algorithm, it maps a domain D to a conjunction of literals $p(c)(D)$, where $D \wedge c \models p(c)(D)$. We assume each propagator is *checking*, that is if $\forall x \in vars(c).|D(x)| = 1$ then $p(c)(D) = \emptyset$ if θ_D is a solution of c and $\{false\}$ otherwise. A propagation solver $prop(P, D)$ applied to a set of propagators P and a domain D repeatedly applies the propagators $p \in P$ until $p(D') = \emptyset$ for $p \in P$, and returns D'.

A *constraint satisfaction problem (CSP)* is a constraint C, often broken into a domain constraint and the remainder $C \Leftrightarrow D \wedge C'$. A *constraint optimization problem (COP)* is of the form $z = \min\{e \mid C\}$, where e is an expression to be minimized and C is a constraint.

In *lazy clause generation (LCG)* solvers [3] propagators are also required to return explanations for each new consequence $l \in p(c)(D)$, that is an explanation clause $e \equiv l_1 \wedge \cdots l_n \rightarrow l$ where $\forall 1 \leq i \leq n, D \models l_i$ and $c \models e$. LCG solvers, like SAT solvers, create an implication graph, where every new consequence is attached to a reason. On failure this is used to create a *nogood* by repeatedly replacing literals in the explanation of failure until only one literal that became true after the last decision remains. This nogood is guaranteed to generate new propagation information. See [3] for more details.

2.2 Lagrangian Decomposition

Lagrangian decomposition is a well understood application of Lagrangian relaxation in order to decompose an optimization problem into parts. Consider an optimization problem of the form $z = \min\{cx \mid C_1(x) \wedge C_2(x)\}$ where z is the objective value, c are the coefficients and x the decisions of the linear objective, and $C_1(x)$ and $C_2(x)$ are arbitrary constraints, then we can provide a lower bound on the objective using

$$\begin{aligned} z = \min\{cx \mid C_1(x) \wedge C_2(x)\} &= \min\{cx \mid C_1(x) \wedge C_2(y) \wedge x = y\} \\ &= \min\{cx + \lambda(x - y) \mid C_1(x) \wedge C_2(y) \wedge x = y\} \\ &\geq \min\{cx + \lambda(x - y) \mid C_1(x) \wedge C_2(y)\} \\ &= \min\{(c + \lambda)x \mid C_1(x)\} + \min\{-\lambda y \mid C_2(y)\} \end{aligned}$$

The problem is decomposed by duplicating the variables and adding a Lagrange multiplier penalty λ to try to force the duplicate variables to be the same.

The above reasoning shows how to break a problem into two parts, the approach straightforwardly generalizes into $n+1$ parts by creating n copies of the variables and n sets of equations relating the copied variables with the original variables.

The correctness of the lower bound holds regardless of the values of λ, but we can get stronger bounds by solving the Lagrangian dual to obtain the best values for λ. In the CP space, since the constraints $C_i(x)$ are arbitrary the usual approach to do this is the subgradient method [4].

In CP many integer variables represent different choices, and the order of the integers is irrelevant, hence applying a Lagrangian penalty like $x_i - y_i$ makes no sense since if it is non-zero it simply represents that two different choices are made of (original) variable x_i. Hence Lagrangian decomposition approaches for CP usually break such integer variables into separate 0-1 variables representing which choice is taken. Given $D_{init}(x_i) = [l..u]$ we replace x_i by 0-1 variables $x_i^j, l \leq j \leq u$ where $x_i^j = 1 \leftrightarrow x_i = j$. We then replace $x_i = y_i$ by the conjunction $\bigwedge_{l \leq j \leq u} x_i^j = y_i^j$. The key advantage for Lagrangian decomposition is that we have separate Lagrange multipliers λ for each such equation.

The CP based Lagrangian decomposition approaches, solve the original problem $z = \min\{cx \mid C_1(x) \wedge C_2(x)\}$ by effectively solving the problem $\min\{z \mid C_1(x) \wedge C_2(x) \wedge z = cx \wedge z \geq z_1 + z_2 \wedge z_1 = \min\{(c+\lambda)x \mid C_1(x)\}\} \wedge z_2 = \min\{-\lambda y \mid C_2(y)\}\}$. Each of the constraints in the master problem are represented by propagators. This requires a propagation algorithm for the each of optimization sub-problem constraints. This is a distinct restriction on the approaches. Some practical approaches to building these propagation algorithms are:

- restrict to a well understood problem: e.g. $z = \min\{-\lambda y \mid dy \leq d_0\}$ is an instance of the knapsack problem, for which many algorithms are known, and indeed quick approximation algorithms are available.
- encode the problem using an existing global: e.g. $z = \min\{-\lambda y \mid C_2(y)\}$ can be represented as a cost-MDD constraint, where the MDD encodes $C_2(y)$. As long as the MDD constraint is not too large then we can use the global cost-MDD propagation algorithms [5, 6].

In the end the difficulty of creating efficient propagators for the optimization sub-problem can severely limit the applicability of Lagrangian decomposition to CP.

3 Objective Splitting Lagrangian Decomposition

The existing approaches to Lagrangian decomposition decompose the problem in a constraint centric way, splitting up the constraints into disjoint subsets and assigning the corresponding part of the objective to each subset. We advocate a decomposition based on breaking up the objective function directly and assigning the corresponding parts of the constraints to each part of the objective. Unlike normal Lagrangian decomposition where each constraint can only belong to one sub-problem, we project the original constraints onto each sub-problem, meaning

that each original constraint could have a projection in more than one sub-problem.

3.1 Problem Decomposition

We consider a similar Lagrange decomposition based on splitting on the objective for $z = \min\{cx + du \mid C(x, u, v)\}$. We consider three classes of variables: x and u appear in the objective and v are auxiliary variables; and split the constraints $C(x, u, v)$ into three categories: $C_1(x, v)$ are constraints only involving x and auxiliaries, $C_2(u, v)$ are constraints only involving u and auxiliaries and $C_0(x, u, v)$ are the remaining constraints. In practice if we have auxiliaries v only related to x we can add them to x, treating them as having coefficient 0 in the objective, similarly for auxiliaries only related to u. This reduces the number of Lagrange multipliers required.

The objective splitting decomposition is based on the following reasoning:

$$
\begin{aligned}
z &= \min\{cx + du \mid C(x, u, v)\} \\
&= \min\{cx + du \mid C_0(x, u, v) \wedge C_1(x, v) \wedge C_2(u, v)\} \\
&= \min\{cx + du \mid C_1(x, v) \wedge C_{02}(x, u, v) \wedge C_2(u, v) \wedge C_{01}(x, u, v)\} \\
&= \min\{cx + du \mid C_1(x, v) \wedge C_{02}(x, u, v) \wedge C_2(u, v') \wedge C_{01}(x, u, v') \wedge v = v'\} \\
&= \min\{cx + du + \lambda(v - v') \mid C_1(x, v) \wedge C_{02}(x, u, v) \wedge C_2(u, v') \wedge C_{01}(x, u, v') \wedge v = v'\} \\
&\geq \min\{cx + du + \lambda(v - v') \mid C_1(x, v) \wedge (\exists u.C_{02}(x, u, v)) \wedge C_2(u, v') \wedge (\exists x.C_{01}(x, u, v'))\} \\
&= \min\{cx + \lambda v \mid C_1(x, v) \wedge \exists u.C_{02}(x, u, v)\} + \min\{du - \lambda v' \mid C_2(u, v') \wedge \exists x.C_{01}(x, u, v')\} \\
&= \min\{cx + \lambda v \mid C_1(x, v) \wedge \exists u.C_{02}(x, u, v)\} + \min\{du - \lambda v \mid C_2(u, v) \wedge \exists x.C_{01}(x, u, v)\} \\
&= \min\{cx + \lambda v \mid \exists u.C_1(x, v) \wedge C_{02}(x, u, v)\} + \min\{du - \lambda v \mid \exists x.C_2(u, v) \wedge C_{01}(x, u, v))\} \\
&= \min\{cx + \lambda v \mid \exists u.C(x, u, v)\} + \min\{du - \lambda v \mid \exists x.C(x, u, v)\} \\
&\geq \min\{cx + \lambda v \mid \bar{\exists} u.C(x, u, v)\} + \min\{du - \lambda v \mid \bar{\exists} x.C(x, u, v)\}
\end{aligned}
$$

where $C_{02}(x, u, v) \Leftrightarrow C_0(x, u, v) \wedge C_2(u, v)$, $C_{01}(x, u, v) \Leftrightarrow C_0(x, u, v) \wedge C_1(x, v)$, and the *quasi projection*, defined later, $\bar{\exists} y.C$ is a formula such that $\exists y.C \Rightarrow \bar{\exists} y.C$. The two weakening steps hold since weakening the constraints can only reduce the minimum value.

The resulting CP optimization problem is $\min\{z \mid C(x, u, v) \wedge z = cx + du \wedge z \geq z_1 + z_2 \wedge z_1 = \min\{cx + \lambda v \mid \bar{\exists} u.C(x, u, v)\} \wedge z_2 = \min\{du - \lambda v \mid \bar{\exists} x.C(x, u, v)\}\}$. Notice that all sub-problems use the same variables, and all constraints are present (in a quasi projected form) in every sub-problem.

We can generalize this to separating into m components

$$
\begin{aligned}
z &= \min\{c_1x_1 + \cdots c_mx_m \mid C(x, v)\} \\
&\geq \min\{c_1x_1 + (\lambda_2 + \cdots \lambda_m)v \mid \bar{\exists} x_2 ... x_m.C(x, v)\} + \\
&\quad \min\{c_2x_2 - \lambda_2 v \mid \bar{\exists} x_1 x_3 ... x_m.C(x, v)\} + \cdots + \\
&\quad \min\{c_mx_m - \lambda_m v \mid \bar{\exists} x_1 ... x_{m-1}.C(x, v)\}
\end{aligned}
$$

Now the resulting problem appears far more complex than the original problem, since we have m sub-problems that appear to be (almost) copies of the original. The advantage that arises is that we have weakened the constraints in the sub-problem and still get correct bounds. Of course if we weaken them too much the bounds will be useless.

The objective based decomposition makes use of existential quantification to allow us to separate constraints that involve objective variables from different classes. Since projection is impractical to compute we weaken the projection. While the logic holds for an arbitrary weakening we will use a certain form we call *quasi projection*.

Given a constraint C with $vars(C) = \mathcal{V}$ a *quasi projection* of C onto variables V, written $\bar{\exists}\{\mathcal{V} - V\}.C$ or $\bar{\exists}_{-V}.C$, is the set of solutions θ over variables V such that $prop(\{p(c) \mid c \in C\}, D_\theta)$ does not return a false domain. We call V the *local variables* of the qausi projection.

Proposition 1. $\exists W.C \Rightarrow \bar{\exists} W.C$

Proof. By definition each solution σ of $\exists W.C$ is such that there exists θ solution of C where $\sigma = \{x \mapsto \theta(x) \mid x \in \mathcal{V} - W\}$. The call $prop(\{p(c) \mid c \in C\}, D_\sigma)$ cannot return a false domain since this would eliminate the solution θ erroneously. Consider the propagator that did this, i.e. $\theta \in D'$ and $D'' = p(D')$ where $\theta \notin D''$. Now $\theta \models D \wedge c$ but $\theta \not\models p(D)$ which contradicts the definition of a propagator. Hence $\sigma \in \bar{\exists} W.C$. □

Example 1. Consider the constraint $C \equiv x < y \wedge y < z \wedge y \mod 3 = 0 \wedge x \in 0..4 \wedge y \in 0..4 \wedge z \in 0..4$. Assuming bounds propagators for the inequalities and a mod propagator that only wakes when y is fixed, the quasi projection onto $\{x, z\}$ is $\{\{x \mapsto 0, z \mapsto 3\}, \{x \mapsto 0, z \mapsto 4\}, \{x \mapsto 1, z \mapsto 4\}, \{x \mapsto 2, z \mapsto 4\}\}$. Note how $\{x \mapsto 0, z \mapsto 2\}$ causes failure since propagation fixes y to 1 where the mod constraint then fails. The actual projection eliminates the first solution. If the propagator for $y \mod 3 = 0$ was stronger, changing the domain of y to $\{3\}$ then the quasi projection would return the projection. □

Example 2. Consider a nurse rostering problem. We have n nurses working for m days and on each day we must choose a shift type in S for each nurse (including a day off). The model has complex restrictions on the sequence of shifts that each nurse can undertake, typically encoded by a `regular` constraint using some finite automata FA, and vectors of upper u and lower l limits on the number of nurses assigned to each shift type on a day, typically encoded by a global cardinality constraint. Finally each nurse i has a preferred shift p_{ij} for each day j, and the aim is to maximize the number of preferences that are met by the schedule. Let x_{ij} represent the shift type chosen for nurse i on day j, then the model is

$$\begin{array}{l} z = \text{minimise} - \sum_{j=1}^{m} \sum_{i=1}^{n} (x_{ij} = p_{ij}) \\ \text{subject to } \texttt{gcc\_low\_up}([x_{ij} | i \in 1..n], S, l, u),\ j \in 1..m \\ \qquad\qquad \texttt{regular}([x_{ij} | j \in 1..m], FA), \qquad i \in 1..n \end{array}$$

We decompose the objective into days, arriving at the following m sub-problems $P(j)$, of the form $y_j = \min\{-\sum_{i=1}^{n}(x_{ij} = p_{ij}) \mid \bar{\exists}_{-V_j}.C\}$ where $V_j = \{x_{ij} \mid i \in 1..m\}$ and C is all the constraints. Note that the constraint $\texttt{gcc\_low\_up}([x_{ij}|i \in 1..n], S, l, u)$ will certainly be satisfied by any solution of the quasi-projection since none of its variables are projected away. In this problem since there are no auxiliary variables we need no Lagrange multipliers. □

Example 3. Consider the problem of max density still life, building a $2m \times 2m$ square which is stable under the Conway's Game of Life rules, and has the maximum number of live cells. The best model [7] for this minimizes wastage (wasted opportunities for placing live cells) which can be computed from each 3×3 subsquare. Let $x_{ij}, 1 \leq i, j \leq 2m$ be the 01 decisions for each cell: 1 is live, 0 is dead. Let $w_{ij}, 2 \leq i, j \leq 2m-1$ be the wastage for the 3×3 subsquare centered at (i, j). A (simplified for ease of exposition) model for the problem is

$$\begin{array}{c} z = \text{minimise} \sum_{i=2}^{2m-1} \sum_{j=2}^{2m-1} w_{ij} \\ x_{i-1,j-1}, x_{i-1,j}, \quad x_{i,j+1}, \\ \text{subject to } \texttt{table}([\ x_{i,j-1}, \quad x_{i,j}, \quad x_{i,j+1}, \ w_{ij}], T), i, j \in 2..2m-1 \\ x_{i+1,j-1}, x_{i+1,j}, x_{i+1,j+1}, \end{array}$$

where T is a table relating 3×3 patterns to their wastage. We will use $wastage_{ij}$ as shorthand for the table constraint. We consider a decomposition of the objective into 4 quadrants $z = \text{minimise} \sum_{i=2}^{m} \sum_{j=2}^{m} w_{ij} + \sum_{i=2}^{m} \sum_{j=m+1}^{2m-1} w_{ij} + \sum_{i=m+1}^{2m-1} \sum_{j=2}^{m} w_{ij} + \sum_{i=m+1}^{2m-1} \sum_{j=m+1}^{2m-1} w_{ij}$ The auxiliary x variables for columns and rows m and $m+1$ are shared by sub-problems and need Lagrange multipliers, the remaining x variables only appear in one sub-problem and do not. The top left quadrant sub-problem has objective

$$\begin{array}{rl} & \sum_{i=2}^{m} \sum_{j=2}^{m} w_{ij} \\ z = \text{minimise} & + \sum_{i=2}^{m} \lambda_{i,m} x_{i,m} + \sum_{j=2}^{m} \lambda_{m,j} x_{m,j} \\ & - \sum_{i=2}^{m} \lambda_{i,m+1} x_{i,m+1} - \sum_{j=2}^{m} \lambda_{m+1,j} x_{m+1,j} - \lambda_{m+1,m+1} x_{m+1,m+1} \end{array}$$

The quasi projection (quasi)eliminates all variables not in top left quadrant except those included in the last line of the objective. All of the *wastage* constraints for the top left quadrant will be guaranteed to be solved since none of their variables are projected out, hence the bound will understand the effect of their interaction on the objective. □

How to split the objective expression into parts remains a question for all Lagrangian decomposition methods. In many problem classes, the partitioning is somewhat natural. It is often the case that problems have a certain amount of locality, where there are certain groups of variables which are strongly related to each other, but weakly related to other variables. We propose to partition the objective function into groups of closely related terms.

Example 4. Example 2 shows how we can meaningful split the nurse scheduling objective into individual days. This make sense for improving the lower bound

since the nursed preferred shifts will contradict the `gcc` constraint, and we will get a much better idea of how many it is possible to simultaneously satisfy. Another possibility is to split it into groups of consecutive days, since these are more tightly related by the `regular` constraint, so the sub-problems then learn about the interaction of `regular` and `gcc`. Alternatively, we could imagine splitting it into individual nurses, thus capturing the effect of the `regular` on the objective. □

Our objective based decomposition differs from the constraint based decomposition of earlier methods [1, 2], and has both advantages and disadvantages. Some points of interest are as follows:

- When a variable in the objective function appears in two or more sub-problems, it is not clear which sub-problem this objective term should be assigned to in order to maximize the effectiveness of the Lagrangian decomposition. In the constraint based decomposition, the constraint split does not completely tell us how to split the objective terms, and indeed we often have to make a second set of decisions as to how to split the objective terms. This decision is handled in a somewhat ad-hoc manner in [1, 2]. Sometimes one sub-problem gets the term, sometimes it is split into two or more parts. It is hard to understand what sort of assignment/split gives the best bound in general. In the objective based decomposition method however, this question does not arise, as the objective split has already fully decided which objective term belongs to each sub-problem, lowering the total amount of decisions that need to be made. Further, we have a good general policy for splitting the objective function, which is to split the objective terms into groups of strongly related terms.
- The constraint based decompositions allow objective terms to be split between sub-problems whereas our proposed objective based decomposition does not. This may be an advantage of the constraint based decomposition as there may be problem classes where splitting objective terms gives a better bound than we can if we are not allowed to split them. On the other hand, our objective based decomposition approach allows constraints to be split (projected) onto multiple sub-problems, whereas in the constraint based decomposition method, each constraint can only appear in one sub-problem. This may be an advantage of the objective based decomposition as many COPs have global constraints that may include all, or many of the variables defining the problem. If we place such a global constraint in only one sub-problem, then all the other sub-problems are substantially weakened. By projecting the global constraint into all of the sub-problems, all of them can get the pruning benefit of the relevant part of that global constraint.
- The constraint based decomposition approach can handle non-linear objective functions by for example assigning the entire objective function to one subproblem. This is not possible in general for the objective based decomposition. However, the objective based decomposition could also potentially handle certain forms of non-linear objective functions in a different way. For

```
bandb(D, V, P, z, S)
    u := max D(z)
    θ := ⊥
    repeat
        best := θ
        θ := search(D, V, P, z, {p(z ≤ u)}, S)
        u := θ(z) − 1
    until θ = ⊥
    return best

search(D, V, P, z, Q, S)
    D := propagate(D, V, P, Q, S)
    if (∃x ∈ V.D(x) = ∅) return ⊥
    if (∀x ∈ V.|D(x)| = 1)
        let θ = {x ↦ d_x | x ∈ V, D(x) = {d_x}}
        return θ
    else
        {c_1, ..., c_m} := branch(D, V)
        for i ∈ 1..m
            θ := search(D, V, P ∪ Q, z, {p(c_i)}, S)
            if (θ ≠ ⊥) return θ
        return ⊥

propagate(D, V, P, Q, S)
    P := P ∪ Q
    repeat
        while (∀x ∈ V.D(x) ≠ ∅ ∧ ∃p' ∈ Q)
            Q := Q − {p'}
            D' := D ∧ p'(D)
            Q := Q ∪ new(P, D, D')
            D := D'
        if (∃s ∈ S.Θ(s) ⊭ D)
            D' := subbound(s, D, P)
            Q := Q ∪ new(P, D, D')
            D := D'
    until Q = ∅
    return D

subbound(s, D, P)
    let s ≡ z = min([o|∃_{−V_s}.C ∧ S_s])
    θ := bandb(D, V_s, P, o, S_s)
    D := D ∧ z ≥ θ(o)
    Θ(s) := θ
    return D
```

Fig. 1. Pseudo-code for evaluating LD COPs.

example, a $min(x_1, \ldots, x_n)$ objective function could be split so that each of the x_i is the objective function for one subproblem.

3.2 Solving the sub-problems

The main difference between our approach and the recent approaches is that we do not require the sub-problems to be of a special form which can be solved via a specialized propagator. Instead, we are going to solve them via standard CP search. This means that our approach can be applied to virtually any CP problem with a linear objective function, rather than only to those which so happen to decompose into sub-problems of specialized forms.

Our approach is as follows. We decide on a splitting of the objective and create the Lagrangian optimization sub-problems. Note that since these problems also have a linear objective we can apply the splitting *recursively* constructing a nested Lagrangian decomposition.

First, we add a new variable z_j representing the objective value of each of those Lagrangian decomposed optimization sub-problems. We add a constraint $z \geq \sum_{i=1}^{m} z_j$ to relate the original objective to these variables. Finally we add the optimization sub-problems defining the z_j. Note that we do not create multiple copies of variables when they belong to multiple sub-problems. Instead, we only require the original copy. In addition, even if a constraint appears in multiple sub-problems, we only need to post one copy of that constraint in the CP solver. This is important because it increases the reusability of nogoods.

The solving of the Lagrangian decomposed COP $z = \min\{c.x \mid C\}$ is as follows. We begin by calling $\mathsf{bandb}(D_{init}, \mathcal{V}, \{p(c) \mid c \in P \cup \{z \geq \sum_{j=1}^{m} z_j\}\}, z, S)$

```
subbound(s, D, P)
    let s ≡ z = min([o|∃̄_{-V_s}.C ∧ S_s])
    l := min D(o)
    while (θ ≡ search(D, V_s, P, o, {p(o ≤ l)}, S_s) = ⊥)
        l := l + 1
    D := D ∧ z ≥ l
    Θ(s) := θ
    return D
```

Fig. 2. Subsearch using destructive lower bounding search.

(shown in Figure 1) where $s_j \in S$ is a Lagrangian decomposed optimization problem of the form $s_j \equiv z_j = \min\{o_j \mid \bar{\exists}_{-V_j}.C\}$.

Notice that each optimization sub-problem is of the same form as the original problem, with a different linear objective and some variables quasi projected. Hence we can apply Lagrangian decomposition on the sub-problems, nesting new Lagrangian decomposed problems within them. The algorithms in Figure 1 handle arbitrary depth of nesting of optimization sub-problems.

Branch and bound search calls search to search for a solution, repeatedly, and then adds constraints to search for better solutions, returning the best solution when it is proved optimal. The search routine is almost standard except: it passes around the sub-problem constraints S, it terminates when all the local variables V are fixed (as opposed to all variables in the problem $\mathcal{V}$), and the branching decisions returned by branch are restricted to only involve local variables V. This implements quasi projection.

The propagation routine propagate is standard, except that it wakes up a sub-problem s, when its incumbent optimal solution $\Theta(s)$ is no longer a support for the lower bound since it is incompatible with the current domain, using subbound to calculate a new lower bound. Initially the incumbent solution $\Theta(s)$ for each sub-problem s is set to $\bot$. We assume $\Theta(s)$ is a backtracking global variable.

The subbound procedure finds the optimal solution θ to the optimization sub-problem $s \equiv z = \min\{o \mid \bar{\exists}_{-V}, C \wedge S_s\}$ where S_s are optimization sub-problems local to S. It uses branch and bound search to minimize o. Crucially the variables of interest are limited to the objective variables for this sub-problem V. Note that the variable reduction is critical for solving the sub-problem more efficiently, since we only look for "solutions" where each local variable is fixed (and the propagators do not detect failure). It sets the lower bound of the sub-problem variable z to that value, as well as storing θ as the incumbent solution.

Alternatively we can use destructive lower bounding search to raise the lower bound of the sub-problem. Unlike normal branch and bound where we iteratively find better and better solutions, in destructive bounding, we start with the tightest bound on the objective function and repeatedly loosen that bound until we find a solution. Destructive bounding is more suitable than normal branch and bound for re-solving a sub-problem when the previous incumbent solution has become invalid. It will immediately try to find a replacement solution which is at least as good as the previous one, and if that fails, it will be able to strengthen

the bound proved and then try to find a solution which is one unit worse, etc. This is generally better than re-solving the sub-problem from scratch via normal branch and bound. Destructive bounding is described in Figure 2. Note that we can break the **while** loop at any time and still get a correct lower bound, although the there will be no incumbent solution in this case. This may be useful in cases where we want to put a time limit on solving the sub-problems. In practice we use normal branch and bound for the first solve of a sub-problem, and use destructive bounding for all re-solves.

3.3 Nogood Learning

Our search-based method integrates seamlessly with nogood learning [3]. In nogood learning, each propagation has to have an explanation clause which explains why that propagation is valid given the current domain. If we want to use nogood learning, then when we derive a bound on z_j via the sub-search on $z_j = \min\{o_j \mid \bar{\exists}_{-V_j}.C\}$, we have to be able to generate a clause which explains the bound on z_j given the current domain. Fortunately, this occurs naturally without any need to modify the nogood learning solver. When we enter a sub-search, any domain changes made by the master search will act as "assumptions" in the sub-search. Any literals representing those initial domain conditions which are relevant to failures in the sub-search will be kept in the nogoods derived during that sub-search. At the end of the proof of optimality phase of the sub-search, we will end up deriving a nogood of form: $l_1 \wedge \ldots \wedge l_n \wedge o_j < \theta(o_j) \rightarrow \mathit{false}$ where l_i are conditions on the variables $\mathcal{V}$ which are sufficient to force that bound on o_j. We can translate this to an explanation for the lower bound $l_1 \wedge \ldots \wedge l_n \rightarrow z_j \geq \theta(o_j)$

Another benefit of nogood learning is that many of the things learned during one sub-search are encapsulated in nogoods and can be reused in subsequent sub-searches of the same sub-problem. Thus we do not have to re-solve those sub-problems from scratch each time, but rather, much of the failed subspace is still encapsulated in the nogoods and can be immediately pruned.

Example 5. Consider an instance of the nurse rostering problem of Example 2 with 8 nurses, requiring at least 3 on day shift (d) and at least 2 on night shift (n), where shift regulations require: no day shift immediately after a night shift, no more than 3 days shifts in a row, no more than 2 night shifts in a row, and no more than one dayoff (o) in a row. Consider a sub-problem instance, for a day j where all nurses request a day shift. Running the sub-problem at the root will discover that $z_j \geq -6$ since at most 6 nurses can get a day shift. When the branch and bound code searches with $o_j \leq -7$ the search fails with explanation $o_j \leq -7 \rightarrow \mathit{false}$, since this makes use only of globally true information. The resulting explanation of the bound is simply $z_j \geq -6$.

Now consider waking the sub-problem when on the previous day $j-1$ we have assigned the first four nurses to night shift, and the last 4 to day shift. Then only the last 4 nurses can be assigned to a day shift on day j. Branch and bound fails when we add $o_j \leq -5$ with explanation $x_{1j-1} = \mathsf{n} \wedge x_{2j-1} = \mathsf{n} \wedge x_{3j-1} = \mathsf{n} \wedge x_{4j-1} = \mathsf{n} \wedge o_j \leq -5 \rightarrow \mathit{false}$. Generating the explanation for $z_j \geq -4$ as $x_{1j-1} = \mathsf{n} \wedge x_{2j-1} = \mathsf{n} \wedge x_{3j-1} = \mathsf{n} \wedge x_{4j-1} = \mathsf{n} \rightarrow z_j \geq -4$. □

3.4 Lagrangian Multipliers

In order to take maximum advantage of Lagrangian multipliers for CP, we differentiate between two different types of integer variables; bounds type integer variables, and value type integer variables. Bounds-type variables are those which are mainly involved in bounds type constraints like linear inequalities. Whereas value-type variables are those which are mainly involved in value type constraints like `alldifferent`, `table` or `regular` constraints. For the latter class of variable we break them into separate 0-1 variables representing each possible value.

We could update Lagrangian multipliers at each call to search using the subgradient method. However, on the problem classes we tried, it appears that updating the Lagrangian multipliers at each node is usually not worth it. Instead we simply calculate the Lagrange multipliers at the root node and use the same multipliers throughout the computation, like [1]. This has an advantage for nogood learning, because the Lagrange multipliers are globally fixed we do not need to include any assumptions about them in explanations for propagation. Using the subgradient method [4] at the root, we update the Lagrangian multipliers for a fixed number of iterations or until the bound derived no longer improves.

3.5 Lazier Bounding

Re-solving the optimization sub-problems does not always yield a better bound on z_j. If the bound does not improve, then all the effort done in the sub-search is wasted. Thus we want to try to only resolve the sub-problems when we have a good chance of improving its bound. The pseudo-code naively re-solves a sub-problem each time its incumbent solution becomes inconsistent with the current domain of the master search, as there is a chance that the bound may be improved. However, this may be too costly.

We propose the following dynamic policy for determining whether to perform the sub-search. For each sub-problem s_j, we have an activation chance p_j which determines whether to re-solve the sub-problem when the incumbent solution becomes inconsistent with the master search. p_j starts at 1. Each time re-solving s_j yields a better bound, we increase p_j by α, capped at 1. Each time re-solving s_j does not yield a better bound, we decrease p_j by β, capped at 0.1. Some reasonable values for α and β are 0.1 and 0.05, and varying these values somewhat did not appear to make much difference. The main idea is that if re-solving the sub-problem often does not yield anything, then p_j will eventually decrease and we will rarely re-solve that sub-problem again, lowering the overhead of the method.

We propose another policy for reducing the overhead of the method. When the master search is searching on the variables of a particular sub-problem, the incumbent solution of that sub-problem will become invalid at almost every decision. If we follow the normal policy of re-solving a sub-problem whenever its incumbent solution becomes invalid, then we will end up re-solving a sub-problem at almost every node in the master search tree. This is clearly very

expensive. It is also often redundant work, because since the master search is searching on the variables of that sub-problem, the bound of that sub-problem will quickly be fixed by the master search anyway and worth trying to strengthen that bound through sub-search. Thus we modify our policy so that if the master search has just made a decision on variable v, then any sub-problem involving v will not be woken up for sub-search at that search node.

4 Experiments

In this section, we describe a few problems as well as how we partitioned their objective function.

Nurse Scheduling Problem This problem is described in Example 2. In the instance we use there are 4 possible shifts including a "day off" shift, and requirements on the number of nurses for each shift and minimum and maximum requirements on the number of holidays per time span. We partition the objective function into partial sums each representing one day, as described in Example 2.

Maximum Density Still Life Problem This problem is described in Example 3, although there we only give a simplified model of the optimization problem, ignoring edge effects, and require the size n is even ($2m$). We partition the objective function by chopping the $n \times n$ region into 9 equal sized square chunks (rather than the 4 of Example 3).

Concert Hall Scheduling Problem In this problem, we have k concert halls and a bunch of orders. Each order hires a hall from a certain start time to a certain end time and gives a certain profit. The problem is to pick the subset of orders to satisfy such that we maximize the profit. Clearly, orders which are close together in terms of their time are more closely related then orders which are far apart in terms of their time. Thus we can quite naturally partition the objective function according to time. We partition the objective function by dividing the time span into 4 equal sized chunks and putting an order in a chunk based on their starting time.

Talent Scheduling Problem In this problem, we have some actors and some scenes. Each scene requires a subset of the actors. Each actor has a cost. The scenes are shot in a certain order. Each actor has to be on-scene from the first scene that they are in until the last scene that they are in is finished. For each day they are on-scene, they have to be paid their corresponding cost. The problem is to find the order of the scenes which minimizes the total cost of the actors. Again, terms which represent costs close in time are more closely related together than those far in time. Thus we partition the objective function into 4 equal sized chunks according to time.

Resource Constrained Project Scheduling Problem with tardiness In this problem, we have some tasks, each of which requires a certain amount of resources on each of the machines. Each machine has a maximum resource capacity. There are some precedences between the tasks. Each task has a due date. For each unit of time past the due date the task is finished by, there is a penalty. The problem is to find the schedule with the least penalty. Tasks which are closer together in terms of their time are more closely related. Thus we partition the objective function into 4 equal sized chunks based on time.

Sweatshop Scheduling Problem In this problem, we have rows of benches, each with some machines. Each machine is assigned a type of garment to make. Each type of garment will cost a different amount of power. Each bench and each bench column has limitations on the total amount of power used. Each person in a row has to work on a different garment. There are also global constraints on the minimum and maximum amount of each garment made. The problem is to find the the assignment of garments which maximizes the profit. Clearly, terms belonging to the same bench/row/column are more closely related to each other than to terms belong to different bench/row/columns. We partition the objective function based on rows.

Traveling Salesman Problem In this problem, we have a number of cities. We have a salesman which must tour all the cities and visit each one exactly once. The problem is to minimise the total amount of distance traveled. Cities which are geographically closer together are more closely related to each other. Thus we divide the objective function by partitioning the cities geographically into 9 square chunks.

The experiments were performed on Xeon Pro 2.4GHz processors using the state of the art LCG solver CHUFFED. We use 20 instances of each problem class, except for Still Life where we use just one. [1] We compare running the above problems with the new sub-problem search-based Lagrangian decomposition (LD via search), with the decomposition but no Lagrangian multipliers (D via search), and without any decomposition (Normal). We also try to compare against the cost-MDD based Lagrangian decomposition method [1] (Bergman et al.). Unfortunately, it is not very clear how that method can be applied to these particular problem classes, as most of them do not decompose into a set of MDD's. Thus we only compare against that method on the Nurse Scheduling problem, as the constraints in that problem can easily be modeled as MDD's. In all the new methods, we use the lazier bounding as described in Section 3.5. We use constructive brand and bound for the first solve of each subproblem and destructive bounding for all subsequent re-solves. We update the Lagrangian multipliers at the root for 100 iterations using the subgradient method or until the bound no longer improves. We use a time out of 10 minutes. The results are shown in Table 1 and Figure 3.

[1] Available from `people.unimelb.edu.au/pstuckey/lgadec`.

Table 1. Comparison between using and not using the search-based Lagrangian decomposition method.

Problem	Normal		LD via search		D via search		Bergman et al.	
	fails	*time*	*fails*	*time*	*fails*	*time*	*fails*	*time*
Nurse Scheduling	618537	56.32	54325	**7.23**	54325	**7.23**	**48630**	13.02
Still Life	478182	57.12	**23415**	3.45	24218	**3.37**	—	—
Concert Hall	389799	35.14	**45231**	**5.27**	158962	12.56	—	—
Talent Scheduling	**1535814**	**215.1**	2589576	417.82	2620582	428.07	—	—
RCPSP with tardiness	**234758**	**68.54**	1834758	487.29	1834758	487.29	—	—
Sweatshop Scheduling	682934	24.87	**124562**	**5.27**	**124562**	**5.27**	—	—
Traveling Salesman	256375	94.56	**185239**	**70.82**	**185239**	**70.82**	—	—

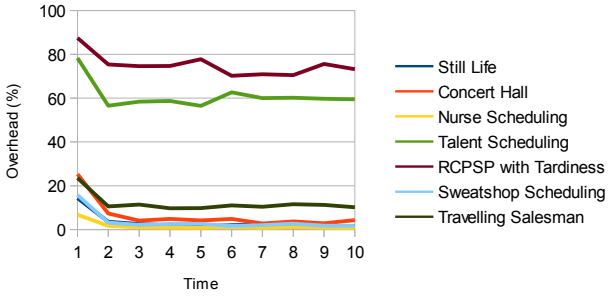

Fig. 3. Overhead of sub-searches per time partition

Table 1 shows the total number of fails and time spent on the benchmarks. These numbers include the fails and time spent in the master search and the sub-problem searches. In all of these benchmarks, the Lagrangian decomposition can give a better bound than that found through normal propagation. However, it is not always worth the overhead. We get significantly stronger bounds for the Nurse Scheduling, Still Life, Concert Hall and Sweatshop Scheduling problems, but relatively weak improvements to the bound for Talent Scheduling, RCPSP with tardiness and the Traveling Salesman Problem. It can be seen that when the bound we derive is significantly better, the overhead of the Lagrangian decomposition method is often worth it. Whereas when the improvement in bound is small, the reduction in the search space of the master search may well be swamped out by the overhead of the sub-searches. For example, in RCPSP with tardiness, the sub-searches occur frequently but the improvement in bound is too insignificant to be worth it. The Lagrangian multipliers are only useful for the Concert Hall problem. For the rest they either have no use because there are no shared variables between sub-problems, or their effect is statistically insignificant. We suspect that this is because Lagrangian multipliers do not work well when sub-problems are connected via value type integer variables, whereas they work far better when sub-problems are connected via bounds type integer variables. The approach of [1] requires less search, since instead of solving the sub-problems via search, the sub-problems are solved by the global propagator instead, which does not contribute to the node count. However, this costs more than the gain in run time compared to our approach.

Figure 3 shows the time overhead of the sub-searches as a percentage of the overall search when the search time is split evenly into 10 parts. It can be

seen that for some problem classes, much more time is spent on solving the sub-problems near the start of the search than in the rest of the search. There are several reasons. The first is that the first solve via branch and bound is often expensive, whereas subsequent re-solves using destructive bounding are often very quick. The second is that the learned nogoods which describe which boundary conditions force certain bounds for the sub-problem may often immediately propagate to avoid some redundant re-solving of sub-problems, making re-solving sub-problems quicker later in the search.

5 Related Work and Conclusion

We have already discussed the most closely related work on Lagrangian Decomposition for CP [1, 2]. The approach we present is closely related to Nested Constraint Programs (NCPs) [8], the optimization sub-problems can be seen as nested CPs where the domain of the variables are defining the sub-problem, for a new copy of the variables in the sub-problem. Because we use the same variables and constraints we can avoid much of the complexity of NCP. Similarly Russian doll search [9] can be seen as a special case of our approach to Lagrangian Decomposition, where there is exactly one optimization sub-problem per level, and no recomputation of the optimization sub-problems.

Lagrangian Decomposition is an exciting development for CP, allowing the same heterogeneous approach to satisfaction to be extended to optimization. In this paper we show how to create a very general scheme for Lagrangian Decomposition using sub-problem search, which, together with learning provides a powerful method for tackling optimization problems that can be meaningfully decomposed. We have shown that the new method can provide significant speedups on some realistic problem classes.

Acknowledgments NICTA is funded by the Australian Government through the Department of Communications and the Australian Research Council through the ICT Centre of Excellence Program. This work was partially supported by Asian Office of Aerospace Research and Development grant 15-4016.

References

1. Bergman, D., Ciré, A.A., van Hoeve, W.: Improved constraint propagation via lagrangian decomposition. In: Principles and Practice of Constraint Programming - 21st International Conference, CP 2015, Cork, Ireland, August 31 - September 4, 2015, Proceedings. (2015) 30–38
2. Ha, M.H., Quimper, C., Rousseau, L.: General bounding mechanism for constraint programs. In: Principles and Practice of Constraint Programming - 21st International Conference, CP 2015, Cork, Ireland, August 31 - September 4, 2015, Proceedings. (2015) 158–172
3. Ohrimenko, O., Stuckey, P., Codish, M.: Propagation via lazy clause generation. Constraints **14**(3) (2009) 357–391
4. Shor, N.: Minimization Methods for Non-differentiable Functions. Springer (1985)

5. Demassey, S., Pesant, G., Rousseau, L.: A cost-regular based hybrid column generation approach. Constraints **11**(4) (2006) 315–333
6. Gange, G., Stuckey, P.J., Hentenryck, P.V.: Explaining propagators for edge-valued decision diagrams. In Schulte, C., ed.: Proceedings of the 19th International Conference on Principles and Practice of Constraint Programming. Volume 8124 of LNCS., Springer (2013) 340–355
7. Chu, G., Stuckey, P.: A complete solution to the maximum density still life problem. Artificial Intelligence **184–185** (2012) 1–16
8. Chu, G., Stuckey, P.J.: Nested constraint programs. In O'Sullivan, B., ed.: Proceedings of the 20th International Conference on Principles and Practice of Constraint Programming. Volume 8656 of LNCS., Springer (2014) 240–255
9. Verfaillie, G., Lemaître, M., Schiex, T.: Russian doll search for solving constraint optimization problems. In: Proceedings of the Thirteenth National Conference on Artificial Intelligence and Eighth Innovative Applications of Artificial Intelligence Conference, AAAI Press / The MIT Press (1996) 181–187
10. Pesant, G., ed.: Principles and Practice of Constraint Programming - 21st International Conference, CP 2015, Cork, Ireland, August 31 - September 4, 2015, Proceedings. Volume 9255 of Lecture Notes in Computer Science., Springer (2015)