Constraint-Based Lagrangian Relaxation

Daniel Fontaine[1], Laurent Michel[1], and Pascal Van Hentenryck[2]

[1] University of Connecticut, Storrs, CT 06269-2155, USA
[2] NICTA and Australian National University, Australia

Abstract. This paper studies how to generalize Lagrangian relaxation to high-level optimization models, including constraint-programming and local search models. It exploits the concepts of constraint violation (typically used in constraint programming and local search) and constraint satisfiability (typically exploited in mathematical programming). The paper considers dual and primal methods, studies their properties, and discusses how they can be implemented in terms of high-level model combinators and algorithmic templates. Experimental results suggest the potential benefits of Lagrangian methods for improving high-level constraint programming and local search models.

1 Introduction

Lagrangian relaxation (e.g., [1]) is a significant optimization paradigm that typically applies to models that feature both easy and hard constraints. Its idea is to relax the hard constraints into the objective, using Lagrangian multipliers to adjust the coefficients of these relaxed constraints. Consider, for instance, the application of Lagrangian relaxation to a mixed integer program:

$$Z = \texttt{min}\ c \cdot x \quad \texttt{s.t.} \begin{cases} A_h x \geq b_h \\ A_e x \geq b_e \\ x \in \{0,1\} \end{cases} \quad \longrightarrow \quad Z_{LR}(\lambda) = \texttt{min}\ c \cdot x + \lambda^T \cdot (b_h - A_h x) \quad \texttt{s.t.} \begin{cases} A_e x \geq b_e \\ x \in \{0,1\} \end{cases}$$

The Lagrangian relaxation can be used to find both dual and primal bounds. The value $Z_{LR}(\lambda)$ is a lower bound of Z for any $\lambda \geq 0$ and primal bounds can be obtained by searching for saddle points to the Lagrangian relaxation (in the x and λ space).

The idea of Lagrangian relaxation is largely independent of how the model is expressed (and solved). It is heavily used in continuous optimization (e.g., [8]) and in mixed integer programming (e.g., [5]). It had attracted some attention in constraint satisfaction problems and SAT in the late 1990s (e.g., [17,3]) but has not been a topic of much research since then.

This research originated from an attempt to build model combinators for Lagrangian relaxation that would apply to arbitrary optimization models. The design of these combinators required a systematic investigation of the semantics of

B. O'Sullivan (Ed.): CP 2014, LNCS 8656, pp. 324–339, 2014.

Lagrangian relaxation over these models, which revealed some interesting modeling, computational, and implementation issues. It also raised the question about the potential benefits of Lagrangian relaxation for constraint programming, hybrid methods, and constraint-based local search, a topic which has been largely neglected in the constraint-programming community.

This paper reports some findings in this direction. It starts by introducing the concept of *satisfiability degree* that provides an alternative to the notion of constraint violation used in constraint programming (e.g., [2]) and constraint-based local search [4,9,14]. The paper then provides natural generalizations of traditional Lagrangian relaxation concepts, including the Lagrangian duals, subgradient optimization [5], surrogate subgradient optimization [18], and primal Lagrangian methods (e.g., [17,3]). These generalizations then makes it possible to design model combinators for Lagrangian relaxation that apply to arbitrary models and algorithmic templates for Lagrangian methods that are independent of the solving technology. The paper also presents some preliminary experimental results indicating the potential of Lagrangian methods for constraint programming and local search, as well as synergies between mathematical programming, constraint programming, and large-neighborhood search.

The remainder of the paper is organized as follows. Section 2 introduces the generalized Lagrangian relaxation idea. Sections 3 and 4 discuss dual and primal Lagrangian methods respectively. Section 5 sketches the implementation in Objective-CP. Section 6 presents the experimental results techniques. Section 7 discusses related work and Section 8 concludes the paper.

2 Generalized Lagrangian Relaxation

This section explores how the traditional formulation of Lagrangian relaxations and Lagrangian duals can be systematically generalized.

2.1 Violation and Satisfiability Degrees

Generalized Lagrangian relaxations are based on the concepts of violation and satisfiability degrees. The violation degree of a constraint is a key concept in constraint programming (e.g., [2]) and constraint-based local search (e.g., [9,14,4]). The violation degree is constraint-dependent and exploits the constraint structure. Intuitively, the violation degree denotes how violated the constraint is.

Definition 1 (Violation Degree). *The violation degree of constraint* $c : \Re^n \rightarrow$ *Bool is a function* $\nu_c : \Re^n \rightarrow \Re^+$ *such that*

$$c(v_1, \ldots, v_n) \equiv \nu_c(v_1, \ldots, v_n) = 0.$$

In contrast, the satisfiability degree of a constraint captures both how much the constraint is violated and the constraint slackness when it is satisfied. It generalizes the Lagrangian relaxation typically used in mathematical programming. Intuitively, when the satisfiability degree is strictly positive, it denotes how much

the constraint is violated; when it is negative, the constraint is satisfied and the satisfiability degree denotes the *slack* of the constraint. When the satisfiability degree is zero, the constraint is satisfied but tight.

Definition 2 (Satisfiability Degree). *The satisfiability degree of constraint* $c : \Re^n \rightarrow Bool$ *is a function* $\sigma_c : \Re^n \rightarrow \Re$ *such that*

$$c(v_1, \ldots, v_n) \equiv \sigma_c(v_1, \ldots, v_n) \leq 0.$$

Example 1 ($A \cdot x \geq b$*).* Consider the constraint $c(x)$ defined by $A \cdot x \geq b$. Its satisfiability degree is given by the function $\sigma_c(x) = b - A \cdot x$. Observe that, when $\sigma_c(v) \leq 0$, $A \cdot v \geq b$ and $c(v)$ is satisfied. When $\sigma_c(v) > 0$, $A \cdot v < b$ and $\sigma_c(v)$ represents how much the constraint is violated. When $\sigma_c(v) = 0$, the constraint is satisfied at equality. When $\sigma_c(v) < 0$, $\sigma_c(v)$ captures the slackness of the inequality.

Example 2 (`disjunctive`*(*s_1, d_1, s_2, d_2*)).* Constraint `disjunctive`(s_1, d_1, s_2, d_2) holds if $s_1 + d_1 \leq s_2 \vee s_2 + d_2 \leq s_1$. Its degree of satisfiability is given by

$$\sigma_d(s_1, d_1, s_2, d_2) = \min(s_1 + d_1, s_2 + d_2) - \max(s_1, s_2).$$

When $\max(s_1, s_2) < \min(s_1 + d_1, s_2 + d_2)$, the two intervals $[s_1, s_1 + d_1[$ and $[s_2, s_2 + d_2[$ overlap, the disjunctive constraint is violated and $\sigma_d(s_1, d_1, s_2, d_2) > 0$. Similarly, if $\max(s_1, s_2) \geq \min(s_1 + d_1, s_2 + d_2)$, the two intervals are temporally separated and $|\sigma_d(s_1, d_1, s_2, d_2)|$ is the temporal slack separating the end of the first activity from the start of the second activity.

The following example illustrates that, for some constraints, the satisfiability degree reduces to the violation degree.

Example 3 (`permutation`*(*$x_1, \cdots, x_n$*)).* Constraint `permutation`$(x_1, \cdots, x_n)$ holds if $x_1, \cdots, x_n$ is a permutation of the values in interval $1..n$. Its degree of satisfiability is given by

$$\sigma_p(v_1, \cdots, v_n) = \sum_{j=1}^{n} \max\left(0, \left(\sum_{i=1}^{n} (x_i = j)\right) - 1\right).$$

When $\sigma_p(v_1, \cdots, v_n) > 0$, the constraint is violated. When $\sigma_p(v_1, \cdots, v_n) = 0$, each value is selected exactly once and the constraint is satisfied. However, $\sigma_p(v_1, \cdots, v_n)$ is never negative as all permutations are equally good: None satisfies the constraint more than the others.

Observe that the violation degree ν_c of a constraint c can always be defined in terms of its satisfiability degree σ_c by stating

$$\nu_c(x_1, \ldots, x_n) = \max(0, \sigma_c(x_1, \ldots, x_n)).$$

We make this assumption in the rest of this paper, when comparing relaxations based on ν_c and σ_c.

2.2 Generalized Lagrangian Relaxations

This section considers Lagrangian relaxations based on violation and satisfiability degrees. It first defines the concepts of constraint softening and constraint relaxation.

Definition 3 (Constraint Softening). *The softening of a constraint c over $\Re^n$ is a constraint $soft(c)$ over $\Re^{n+1}$ defined as*

$$soft(c)(x_1, \cdots, x_n, y) \equiv y = \nu_c(x_1, \cdots, x_n).$$

Definition 4 (Constraint Relaxation). *The relaxation of a constraint c over $\Re^n$ is a constraint $relax(c)$ over $\Re^{n+1}$ defined as*

$$relax(c)(x_1, \cdots, x_n, y) \equiv y = \sigma_c(x_1, \cdots, x_n).$$

We are now in a position to define generalized and soft Lagrangian relaxations.

Definition 5 (Generalized and Soft Lagrangian Relaxations). *Consider the optimization problem*

$$\begin{array}{l} P = \min_x f(x) \\ \textit{subject to} \begin{cases} c_h(x) & (h \in H) \\ c_e(x) & (e \in E) \end{cases} \end{array}$$

where H and E denote, respectively, the index sets of hard and easy constraints. The generalized Lagrangian relaxation of P for a set of Lagrangian multipliers $\lambda_h \geq 0$ is given by

$$\begin{array}{ll} GLR(\lambda) & = \min_x f(x) + \sum_{h \in H} \lambda_h \cdot \sigma_h \\ \textit{subject to} & \begin{cases} c_e(x) & (e \in E) \\ relax(c_h)(x, \sigma_h) & (h \in H) \end{cases} \end{array}$$

The soft Lagrangian relaxation of P for a set of Lagrangian multipliers $\lambda_h \geq 0$ is given by

$$\begin{array}{ll} SLR(\lambda) & = \min_x f(x) + \sum_{h \in H} \lambda_h \cdot \nu_h \\ \textit{subject to} & \begin{cases} c_e(x) & (e \in E) \\ soft(c_h)(x, \nu_h) & (h \in H) \end{cases} \end{array}$$

The definitions of the generalized and soft Lagrangian relaxations are compositional and constraint-driven. This makes it possible to define model combinators that systematically obtain a Lagrangian relaxation from a high-level model as discussed in Section 5. The following (direct) lemma establishes the soundness of the approach.

Lemma 1 (Relaxations). *Consider the optimization problem P defined above, an optimal solution x^* of P, and the generalized and soft relaxations $GLR(\lambda)$ and $SLR(\lambda)$ for a vector $\lambda \geq 0 \in \mathbb{R}^{|H|}$. Then, $GLR(\lambda)$ and $SLR(\lambda)$ are relaxations of P, i.e., $GLR(\lambda) \leq f(x^*)$ and $SLR(\lambda) \leq f(x^*)$.*

Proof. Each feasible solution of P satisfies $\nu_h = 0$ and $\sigma_h \leq 0$ for all $h \in H$. Hence, in $SLR(\lambda)$ (resp. $GLR(\lambda)$), the objective value of a feasible solution is the same as (resp. no greater than) the objective value of a feasible solution in P. The results follows since the λ_h are nonnegative. □

The following (also direct) lemma shows that the soft relaxation is at least as strong as the generalized relaxation.

Lemma 2 (GLR versus SLR). *For any $\lambda \geq 0$, we have $GLR(\lambda) \leq SLR(\lambda)$.*

This lemma seems to suggest the use of $SLR(\lambda)$ instead of $GLR(\lambda)$ (which generalizes the traditional mathematical approach), since it is a stronger relaxation. Indeed, $SLR(\lambda)$ could be defined as

$$\begin{array}{ll} SLR(\lambda) & = \min_x f(x) + \sum_{h \in H} \lambda_h \cdot \nu_h \\ \text{subject to} & \begin{cases} c_e(x) & (e \in E) \\ relax(c_h)(x, \sigma_h) & (h \in H) \\ \nu_h \geq 0 & (h \in H) \\ \nu_h \geq \sigma_h & (h \in H) \end{cases} \end{array}$$

which does not change the theoretical complexity of the relaxation if $GLR(\lambda)$ is a linear program or a mixed integer program. The experimental results in Section 6 will shed some light on this issue.

3 Generalized Lagrangian Duals

Lagrangian relaxation is often used to find tight dual bounds to optimization problems. The aim is thus to determine the set of Lagrangian multipliers λ that gives the strongest dual bound. This section focuses on the generalized Lagrangian relaxation but the results apply to the soft Lagrangian relaxation as well. The generalized Lagrangian dual can then be defined as

$$GLR^* = \max_{\lambda \geq 0} GLR(\lambda).$$

The generalized Lagrangian dual satisfies the following property.

Theorem 1 (Optimality Test). *Let $\hat{x}$ be an optimal solution to $GLR(\lambda)$ for some $\lambda \geq 0$ such that*

1. *$c_h(\hat{x})$ holds for all $h \in H$.*
2. *$\lambda_h \cdot \sigma_h = 0$ for all $h \in H$.*

Then, $\hat{x}$ is an optimal solution to P and $GLR^* = GLR(\lambda)$.

Proof. By condition (1) and the definition of $GLR(\lambda)$, $\hat{x}$ is a feasible solution. Moreover, by condition (2), $GLR(\lambda) = f(\hat{x}) + \sum_{h \in H} \lambda_h \cdot \sigma_h = f(\hat{x})$. Since $f(\hat{x})$ is both a lower and an upper bound, the result follows. □

Note that, for *SLR*, condition (1) implies condition (2).

```
1  function SubgradientSolve(GLR(λ), Z_UB)
2      π = 2
3      k = 0
4      λ^0 = 0
5      Z_best = −∞
6      noImproveCount = 0
7      do
8          x^{k+1} = solve(GLR(λ^k))
9          Z^{k+1} = f(x^{k+1}) + Σ_{h∈H} λ_h^k · σ_h(x^{k+1})
10         Δ^{k+1} = π(Z_UB − Z^{k+1})/||σ(x^{k+1})||^2
11         forall (h ∈ H) λ_h^{k+1} = max(0, λ_h^k + Δ^{k+1} * σ_h(x^{k+1}))
12         if Z^{k+1} > Z_best
13             Z_best = Z^{k+1}
14             noImproveCount = 0
15         else noImproveCount = noImproveCount + 1
16         if noImproveCount > 30
17             π = π/2
18             noImproveCount = 0
19         k = k + 1
20     while the termination criterion is not met;
21     return Z_best
```

Fig. 1. The *subgradient* Algorithm Template

Subgradient Optimization. The generalized Lagrangian dual can be rewritten explicitly as

$$\begin{aligned} &\max_{\lambda \geq 0} w \\ &\text{subject to} \\ &\quad w \leq f(x) + \lambda^T \cdot \sigma_h(x) \quad \forall x, e \in E : c_e(x). \end{aligned}$$

This formulation has exponentially many constraints but it can be solved by a subgradient method which iterates two steps

$$\begin{aligned} x^{k+1} &= solve(GLR(\lambda^k)) \\ \lambda_h^{k+1} &= \max\left(0, \lambda_h^k + \Delta^{k+1}\ \sigma_h(x^{k+1})\right)\ (h \in H) \end{aligned}$$

where Δ^k is the step size at iteration k. What remains to determine is the initial value of the multipliers and the step size at each iteration. The algorithmic schema for subgradient optimization is depicted in Figure 1 and is independent of the model and the solving technology. Its input is a parametric Lagrangian model $GLR(\lambda)$ and an initial upper bound to the original problem P. The algorithmic template also uses an agility parameter π used to compute the step sizes. The subgradient optimization sets the initial multipliers λ^0 to 0. Lines 8–19 repeatedly solves the parametric model with the current multipliers λ^k and store its solution in x^{k+1} and its objective value in Z^{k+1} (lines 8–9). Lines 10 computes the step function Δ^{k+1} used on line 11 to compute the next multipliers λ^{k+1}. Lines 12–19 record the current best objective and update the agility parameter π when there is no improvement over some time. Observe that the template does not prescribe any technology for solving $GLR(\lambda^k)$.

Generalized Surrogate Optimization. The surrogate gradient method was introduced in [18] and refined in [13] to solve a Lagrangian dual featuring loosely

coupled subproblems. By relaxing the coupling constraints, the subproblems can then be optimized independently.

Consider the following simple IP minimization problem:

$$Z = \texttt{min} \ \sum_{i=1}^{4} x_i$$
$$\texttt{s.t.} \begin{cases} x_1 + x_2 \geq b_1 \\ x_3 + x_4 \geq b_2 \\ x_1 + x_3 + x_4 \geq b_3 \end{cases}$$

Relaxing the coupling constraint produces the Lagrangian dual:

$$Z_{LD} = \texttt{min} \ \sum_{i=1}^{4} x_i + \lambda(b_3 - (x_1 + x_3 + x_4))$$
$$\texttt{s.t.} \begin{cases} x_1 + x_2 \geq b_1 \\ x_3 + x_4 \geq b_2 \end{cases}$$

The objective function of Z_{LD} can be simplified algebraically in order to obtain two separable subproblems:

$$min[x_1(1-\lambda) + x_2] + [(x_3 + x_4)(1-\lambda)]$$

Such rewritings are not always possible when using arbitrary models featuring global constraints. But the surrogate subgradient algorithm can be generalized to arbitrary models by using ideas from large neighborhood search. At each iteration, a subproblem can be chosen and all the variables not appearing in this subproblem are fixed to their values in the incumbent solution. A subproblem $GLR(\lambda, \hat{x}, V)$, where $\hat{x}$ is an incumbent solution and V is the set of variables associated with one of the subproblems, can be defined as

$$GLR(\lambda, \hat{x}, V) = \min_x f(x) + \sum_{h \in H} \lambda_h \ \sigma_h(x)$$
$$\text{subject to} \quad \begin{cases} c_e(x) \quad (e \in E) \\ x_i = \hat{x}_i \ (i \notin V) \end{cases}$$

With this idea in mind, the generalized surrogate gradient template is presented in Figure 2. It receives as inputs the parametric model and the set of variables appearing in each subproblem. Observe that line 6 solves the initial model entirely before starting the subproblem optimization. Once again, the template does not prescribe any technology for solving $GLR(\lambda^k, \hat{x}, V)$.

4 Generalized Lagrangian Primal Methods

Primal Lagrangian methods are ubiquitous in continuous optimization. In the late 1990s, some of their main concepts were elegantly transferred to discrete optimization [12,17]. Focusing on violation degrees, the resulting Lagrangian primal methods (SPLR) can be viewed as the iteration of two steps:

$$x^{k+1} = \text{argmin}_{x \in \mathcal{N}(x^k)} SLR(\lambda^k, x^k)$$
$$\lambda^{k+1} = \lambda^k + \nu(x^{k+1})$$

```
1  function SurrogateSolve(GLR(λ), Z_UB, {V_1, ..., V_k})
2    k = 0
3    λ^0 = 0
4    Z_best = −∞
5    noImproveCount = 0
6    x^0 = Solve(GLR(λ^0))
7    do
8         Z^k = f(x^k) + Σ_{h∈H} λ_h^k σ_h(x^k)
9         Δ^k = (Z_UB − Z^k)/||σ(x^k)||^2
10        forall (h ∈ H)  λ_h^{k+1} = max(0, λ_h^k + Δ^k * σ_h(x^k))
11        if  Z^k > Z_best
12             Z_best = Z^k
13             noImproveCount = 0
14        else noImproveCount = noImproveCount + 1
15        select  i ∈ 1..k
16        y = Solve(GLR(λ^{k+1}, x^k, V_i))
17        obj = f(y) + Σ_{h∈H} λ_h^{k+1} σ_h(y)
18        if  obj < z^k
19             x^{k+1} = y
20        else  x^{k+1} = x^k
21        k = k + 1
22   while  the termination criterion is not met;
23   return  Z_best
```

Fig. 2. The *Surrogate Subgradient* Algorithm Template

where $\mathcal{N}(x)$ is the neighborhood around x, i.e., a set of points satisfying the easy constraint and including x, and $SLR(\lambda, x) = f(x) + \lambda\nu(x)$. Such primal Lagrangian methods thus descend in the x-space and ascend in the λ-space. Such primal Lagrangian methods were applied to SAT [12] and constraint satisfaction problems [3], with neighborhoods changing the value of one variable. However, they have not attracted much attention in the constraint-programming community since then. It is useful to state the main theoretical results from [17], since they shed some light on the search algorithm.

Definition 6 (Discrete Saddle Point). *A pair* (λ^*, x^*) *is a discrete saddle point of SLR if* $SLR(\lambda, x^*) \leq SLR(\lambda^*, x^*) \leq SLR(\lambda^*, x)$ *for all* λ *and* $x \in \mathcal{N}(x^*)$.

The left condition in the definition can be shown to be equivalent to $\nu(x^*) = 0$. The following theorem is a direct adaptation to our setting of the main results in [17].

Theorem 2 (Saddle Point Theorem). *Point* x^* *is a local minimum to the original problem* P *if and only if there exists* $\lambda^* \geq 0$ *such that* (λ^*, x^*) *is a discrete saddle point. Moreover,* (λ^*, x^*) *is a saddle point if and only if* $x^* = argmin_{x \in \mathcal{N}(x^*)} SLR(\lambda^*, x^*)$ *and* $\nu(x^*) = 0$.

Observe also that if (λ^*, x^*) is a saddle point, so is (λ, x^*) for $\lambda \geq \lambda^*$ [17]. Hence, in theory, there is no need to decrease the Lagrangian multipliers when searching for a saddle point. It is thus unclear whether the satisfiability degree is useful in primal Lagrangian methods.

In contrast to earlier work, this paper studies whether primal Lagrangian methods can provide a simple, systematic, and principled way of boosting existing search methods, such as tabu search or large neighborhood search, when applied to high-level models. In other words, the neighborhood $\mathcal{N}$ in these primal Lagrangian methods is very large and defined by a neighborhood search technique over a high-level model.

5 Practical Implementation

The earlier sections defined a general framework for applying Lagrangian relaxation to high-level models. This section describes how this generality is supported in Objective-CP [15]. Intuitively, the implementation starts with a high-level model which is then relaxed by replacing the hard constraints with their relaxation and adding a new term in the objective function to capture the weighted sum of violations or satisfiability degrees. The hard constraints are identified either by users or automatically by a partitioning algorithms. The resulting Lagrangian model is then concretized into an optimization program, which can be a MIP solver, a constraint-programming solver, or a constraint-based local search. The concrete optimization program is then embedded in an algorithmic template (a runnable in Objective-CP's terminology [6], e.g., a surrogate dual or a primal Lagrangian methods. We now illustrate this methodology on a few code snippets. Consider the excerpt

```
1 id<ORModel>  P = [ORFactory createModel];
2 ...
3 id<ORIdArray>  H  = ... // array of hard constraints in P
4 id<ORModel>    L  = [ORFactory lagrangianRelax: P relaxingConstraints: H];
5 id<ORProgram>  O  = [ORFactory createMIPProgram: L];
6 id<ORRunnable> r  = [ORFactory subgradient: O];
7 [r run];
```

The code fragment starts by declaring a model P on line 1. Line 3 stores the set of constraints deemed hard in P in array H. Line 4 creates a parametric model L representing $GLR_P(\lambda)$. Line 5 concretizes $GLR_P(\lambda)$ into a parametric MIP program O, which is solved using a subgradient template in Lines 6–7. To switch to a `CP` solver, it suffices to change line 5 into

```
1 id<ORProgram> O = [ORFactory createCPProgram: L];
```

Similarly, to use violation degrees rather than satisfiability degrees, it is sufficient to edit line 4 to read

```
1 id<ORModel>    L = [ORFactory lagrangianRelax: P softeningConstraints: H];
```

Observe that, following [6], Objective-CP stores the fact that L is a relaxation of P and the runnable produces several products in agreement with a relaxation specification, including a stream of lower bounds. It can thus be composed naturally with a primal algorithm.

Consider now the application of a surrogate optimization scheme.

```
1 id<ORModel>       P = [ORFactory createModel];
2 ...
3 id<ORIdArray>     H = ... // array of hard constraints in P
4 id<ORPartition> Vs = [ORFactory autoPartition: P           accordingTo: H];
5 id<ORModel>       S = [ORFactory lagrangianRelax: P relaxingConstraints: H];
6 id<ORProgram>     O = [ORFactory createMIPProgram: S];
7 id<ORRunnable>    r = [ORFactory surrogate: O splitWith: Vs];
8 [r run];
```

Line 4 computes a partition of the variables in P from the hard constraints. Line 5 creates the Lagrangian relaxation of P with respect to H and line 6 creates a MIP program that is then used by a surrogate runnable in line 7. The partition in line 4 is the argument $\{V_1, \cdots, V_k\}$ appearing in the template in Figure 2.

Models with a natural decomposable or "block" structure are often difficult to decompose by hand, particularly for larger problems. Hence, it is useful to have the ability to automatically partition a problem based on sets of *coupling* constraints. Objective-CP makes use of a hyper-graph clustering algorithm [7] to provide an automatic decomposition. The variables of a model become nodes and each constraint defines an hyper-edge connecting it variables. The algorithm recursively clusters variables into disjoint sets until the maximal decomposition is achieved.

6 Empirical Results

This section reports some experimental results highlighting the concepts described in this paper. The goal is not to present state-of-the-art results on specific problems but to make the case that Lagrangian relaxation could play a larger role in the constraint-programming community. In addition, the experiments present some interesting perspectives on some design choices in Lagrangian methods. All experimental results are obtained using Objective-CP [15] unless specified otherwise. Mixed-Integer programs are solved using Gurobi 5.6.

6.1 Graph Coloring

Graph coloring aims at minimizing the number of colors necessary to color a graph so that no two adjacent vertices have the same color. The following is a typical CP formulation of the problem:

$$Z_{CP} = \texttt{min}\ m$$
$$\text{subject to} \begin{cases} v_i \leq m, & i \in 1..|V| \\ v_i \neq v_j, & (i,j) \in E \\ v_i \in 1..|V|, & i \in 1..|V| \\ m \in 1..|V| \end{cases}$$

Here, V is the set of vertices, E the set of edges, m a decision variable for the number of colors used and $\{v_i\}_{i \in 1..|V|}$ are decision variables representing the color assigned to the i-th vertex of V. In the Lagrangian relaxation, E is partitioned into a 'hard' and 'easy' edge set $E = E_e \cup E_h$, relaxing E_h. The experiments also use a MIP formulation automatically obtained from the above model

Table 1. Experimental Results on *Graph Coloring*

Instances	Dual											Primal				
	GLR(MIP)				SLR(MIP)				SLR(CP)			MIP			CP	
	time	lb	ub	itr	time	lb	ub	itr	time	lb	itr	time	lb	up	time	ub
20-3-39	0.08	11*	11*	1	0.07	11*	11*	1	0.44	11*	10.95	0.06	11*	11*	**0.01**	11*
120-25-188	300	9	9	173	3.4	9*	9*	2	30.25	9*	8.35	**1.52**	9*	9*	1.98	9*
160-2-846	300	55	160	1	300	55	160	1	300	105	104.5	300	55	160	**0.07**	108*
160-30-187	300	9	9	103	6.77	9*	9*	2	0.11	9*	4.35	2.5	9*	9*	**0.03**	9*
80-10-176	300	13	13	266	2.15	13*	13*	2	8.25	13*	13.3	0.90	13*	13*	**0.02**	13*
200-10-281	300	30	30	30	23.48	30*	30*	2	12.14	30*	30.0	12.03	30*	30*	**11.03**	30*
120-3-465	300	53	53	33	300	53*	53*	34	37.25	53*	51.15	10.0	53*	53*	**0.05**	53*
160-4-498	300	62	62	16	40.4	62*	62*	2	54.04	62*	61.0	20.01	62*	62*	**6.92**	62*
120-5-938	300	34	34	49	300	34	34	54	300	34	33.5	**9.48**	35*	35*	25.94	35*
200-20-201	300	16	16	47	12.8	16*	16*	2	3.58	16*	16.15	5.77	16*	16*	**0.03**	16*
180-5-873	300	51	51	11	176.85	51*	51*	2	300	51*	49.5	**23.29**	51*	51*	37.04	51*
100-2-910	12.5	71*	71*	1	12.40	71*	71*	1	300	69	67.5	12.52	71*	71*	**0.03**	71*
150-5-1803	-	-	-	-	-	-	-	-	300	41	39.5	26.8	46*	46*	**11.11**	46*
140-12-1137	-	-	-	-	-	-	-	-	300	19	10.0	**12.04**	20*	20*	55.7	20*

by a linearization transformation. The Objective-CP linearization performs a *binarization* of the variables $\{v_i\}_{i\in 1..|V|}$ over their domains and transforms the (non-linear) disequality constraints into sets of inequalities.

The experiments consider three dual methods (GLR(MIP), SLR(MIP), and SLR(CP)), as well as two primal methods (MIP, CP). The methods are evaluated on randomly generated instances[1] which are built around collections of vertex cliques connected via *coupling edges*. More precisely, the vertex set is first partitioned into randomly sized cliques. Then a subset of vertices are chosen at random and *coupling edges* between these vertices are added. These instances are generated based on three parameters: *number of vertices* (nbv), *number of cliques* (nbc), *number of coupled vertices* (nbcv). In Figure 1, instances are referred to in the following format: 'nbv-nbc-nbcv'. The *relaxed edges* E_h used in GLR(MIP), SLR(MIP), and SLR(CP) are a subset of the *coupling edges*. The problem is first decomposed into independent sets (cliques in this case) using a standard hyper-graph partitioning algorithm. Edges which do not have a vertex in the maximal clique are relaxed. Initial upper bounds provided to the dual problem were about twice the optimal value.

Table 1 describes the results on 15 instances and it reports the runtime and the bounds produced by the various algorithms. Dual algorithms only report a lower bound *lb* while the MIP produces both lower and an upper bounds, and the CP program produces an upper bound only. Dual algorithms may terminate because of a timeout or because the step size is too small in which case the dual solution is typically primal-infeasible. Theorem 1 specifies when the Lagrangian dual produces an optimal solution. The table also reports the number of Lagrangian iterations. Bold entries correspond to the fastest implementation, while *starred*

[1] Python script for generating instances: `http://bit.ly/1jDCgJq`

Table 2. Experiments on *Set Covering* problems

Instances	GLR			SLR 5s			SLR 10s			SLR		
	time	itr	bnd	time	itr	bnd	time	itr	bnd	time	itr	bnd
inst 1	243.8	154	155.8	900	13	164.0	900	15	163.2	900	7	161.6
inst 2	900	109	149.2	900	12	153.1	900	12	155.3	900	9	153.5

entries indicate whether an optimal solution was found and proved optimal. A timeout of 300 seconds is used throughout.

The results bring some interesting conclusions. The dual MIP approaches perform poorly on these benchmarks and are strongly dominated by the primal MIP. **The dual CP is better than the dual MIP approaches but is typically bettered by the Primal CP formulation. The primal CP approach is the most effective approach on a number of benchmarks but is sometimes dominated by the primal MIP. These results seem to indicate that it would be valuable to investigate combinations of Lagrangian relaxation and constraint programming systematically on more applications. Note that the absence of lower-bounds and the large number of symmetries is detrimental to SLR(CP) which explores alternative selections of violated colorings.

6.2 GLR versus SLR

Lemma 2 indicated that SLR is a stronger relaxation than GLR, although GLR is the traditional Lagrangian relaxation in mathematical programming. Experimental results on graph coloring indicated that $SLR(MIP)$ systematically outperforms $GLR(MIP)$ on these instances and performs significantly fewer iterations. This section aims at confirming these results on set-covering instances. The results are based on random instances generated in separable blocks of random sizes which are extended with *coupling constraints*. The instances consider 1000 elements and 400 sets partitioned into 10 separable blocks and 250 coupling constraints (which are relaxed). The results for two representative instances are presented in Figure 3 and Table 2. The results again indicate that GLR and SLR behave very differently. SLR tends to have longer iterations but make larger jumps, while GLR features relatively rapid iterations but makes much less progress per iteration. It is also possible to speed-up SLR substantially by using a time limit. SLR then returns its best lower bound at the time limit when an improved lower bound has been found; otherwise, it continues until such a lower bound is found or the search is complete. Figure 3 shows the bound quality of GLR, SLR, and the time-limited SLR with limits of 5 seconds (SLR 5s) and 10 seconds (SLR 10s). A 15 minute (900s) time out is used.

The results on these set-covering instances shed some light on the respective strengths of GLR and SLR. On the first instance, GLR terminates because the step size has become too small (due to lack of bound improvement). Once again, SLR approaches dominate GLR on these instances. Results on instance 2 is

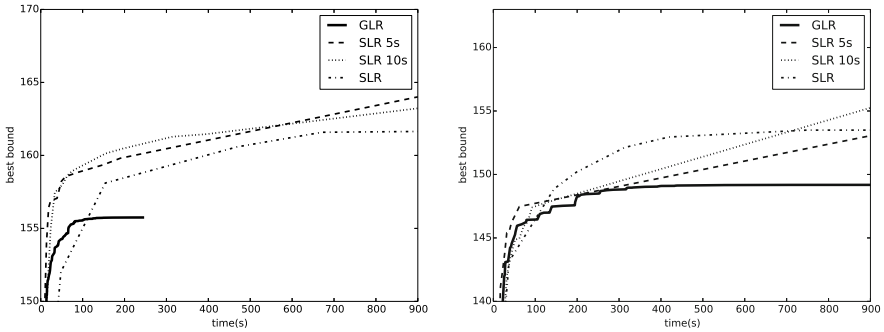

Fig. 3. Bound quality over time, GLR vs SLR

also revealing in that the smaller time limit gives better bounds early on but eventually falls behind and produces poorer bounds. Once again, these results seem to indicate that concepts from constraint programming, i.e., the degree of violations, could bring benefits into traditional Lagrangian dual methods.

6.3 Primal Lagrangian Tabu Search

This section describes the application of Lagrangian primal methods to boost the performance of a tabu-search algorithm. The tabu search is used to explore the neighborhood in the SPLR method; Upon completion, the Lagrangian multipliers are updated using the violation degrees and the tabu-search algorithm is restarted. The experiments are performed on the hardest instances of the progressive party problem using the model and the tabu search presented in [14] but with no restarting component and no manual tuning of the constraint weights. The progressive party problem features a variety of global constraints (e.g., *alldifferent* and *packing* constraints). It is thus fundamentally different from the benchmarks typically used to demonstrate the weighting schemes which uses SAT or binary CSPs, or binary constraints (e.g., [12,3,11]).

Figure 4 reports the experimental results. SPLR is a primal Lagrangian method using the tabu search (with no restart) to explore the neighborhood $\mathcal{N}$. SPLR updates the weights after n^2 iterations of the tabu search, where n is the number of variables. Tabu is the tabu search with restarts, i.e., the control case. W-Tabu is the Comet implementation of the tabu search with hand-chosen constraint weights and restarts (running on a slightly faster processor). All algorithms have a limit of 1,000,000 iterations and were executed 50 times on each instance. The table reports the configuration C, the number of periods P, the success percentage $\%S$ and the minimum, maximum, average, and standard deviation for the number of iterations and CPU time.

The results shows that SPLR significantly outperforms the tabu search in speed and success rate on these instances, indicating that Lagrangian primal

Algorithm	C	P	$\%S$	$m(I)$	$M(I)$	$\mu(I)$	$\sigma(I)$	$m(T)$	$M(T)$	$\mu(T)$	$\sigma(T)$
SPLR	3	9	98	63019	1000000	373515.82	233297.49	4.68	73.95	26.36	16.65
	4	9	94	73319	1000000	386478.46	246292.98	5.14	73.01	27.59	17.90
	6	7	94	40010	1000000	312445.44	280601.45	3.24	76.61	23.26	21.11
Tabu	3	9	30	22729	1000000	816440.38	311171.57	1.855	86.542	59.46	23.15
	4	9	48	17692	1000000	767466.30	313399,19	1.40	82.36	55.59	23.15
	6	7	64	130117	1000000	733028.38	285338.94	8.699	66.11	47.55	18.56
W-Tabu	3	9	96	23645	1000000	245331.96	249549.41	4.41	164.44	40.10	40.14
	4	9	94	47962	1000000	363842.50	273432.48	8.35	166.90	59.55	44.44
	6	7	88	19314	1000000	379072.73	328393.13	3.24	152.37	56.92	48.91

Fig. 4. Experimental Results for SPLR on the Progressive Party Problem

methods may provide a simple, systematic, and principled way to boost the performance of meta-heuristics and complex search procedures. Moreover, SPLR exhibits a performance similar to W-Tabu on instances (3,9) and (4,9) and outperforms it slightly on instance (6,7). This indicates that primal Lagrangian methods may find proper multipliers quickly (the theory indicates that such multipliers exist but not how fast they can be identified).

It is also interesting to report some additional insights on SPLR. Indeed, experimental results show that using the satisfiability is counter-productive in this setting, the search rarely converging to a feasible solution. Moreover, using a restarting component is also not productive, which is a surprise given the importance on restarts for tabu search on this benchmark. The Lagrangian multipliers are very effective in driving the search out of local minima on this problems.

7 Related Work

This section reviews related work and positions this research, complementing the citations already given in the paper. To the best of our knowledge, this paper presents the first implementation of the Lagrangian dual with constraint programming, showing the benefits of Lagrangian relaxation to improve dual bounds in constraint programming and speed up optimality proofs. The paper also shows how to generalize the surrogate method for solving the Lagrangian dual, using ideas from large neighborhood search. **This is the first** systematic, compositional, and technology-independent implementation of the surrogate method. The paper also suggests that it may be valuable to consider violation degrees instead of satisfiability degrees in a variety of applications. Primal Lagrangian methods were generalized from continuous to discrete optimization in [12,17] and applied to SAT and CSPs in [17,3]. The focus in this paper was to suggest that Lagrangian methods can boost the performance of existing local or large neighborhood search systematically and compositionally. In that sense, this resulting algorithmic template is close to guided local search [16]. There are some interesting differences however, including the fact that the updates of Lagrangian

multipliers use the violation degrees and that the underlying search can be arbitrary. Lagrangian relaxation was used to boost the coupling, communication, and propagation capabilities of two global propagators for optimization constraints in in [10]. The key insight is to solve a sequence of Lagrangian relaxations for the two propagators, using dual values at optimality of one propagator to seed the multipliers for the second optimization. This use of Lagrangian relaxation is rather different from this research where Lagrangian relaxation is used at the model level and existing methodologies are generalized to become compositional, to apply to high-level models, and to leverage multiple solution technologies.

8 Conclusion

The key contribution of this paper is to generalize Lagrangian relaxation to arbitrary high-level models. Lagrangian relaxations of such models can then be concretized into a variety of optimization technology (e.g., constraint programming, local search, or MIP). The resulting concrete optimization programs can be entrusted to algorithmic templates to solve Lagrangian duals or use Lagrangian primal methods. The paper also showed that Lagrangian relaxations can be built around the notion of satisfiability degrees (typical in mathematical programming) or violation degrees (typically in constraint programming and local search). Finally, the paper also indicated how to apply surrogate optimization systematically in a generic algorithmic template that optimizes independent problems separately. The experimental results show the versatility of Lagrangian relaxation for a variety of solver technologies and models. In particular, they show that

- The Lagrangian dual coupled with constraint programming is an effective method for some classes of graph coloring problems.
- The concept of violation degree is valuable to improve the quality and performance of the Lagrangian dual when solved with MIP solvers. It is not clear however whether satisfiability degrees can be valuable for constraint programming or local search.
- Primal Lagrangian methods may systematically boost the performance and solution quality of meta-heuristics in a principled way.

Overall, these results tend to indicate that Lagrangian methods could play a much more significant role in constraint programming and large neighborhood search and that further synergies between constraint programming and mathematical programming should be explored.

References

1. Ahuja, R.K., Magnanti, T.L., Orlin, J.B.: Network Flows: Theory, Algorithms, and Applications. Prentice Hall, Englewood Cliffs (1993)
2. Beldiceanu, N., Petit, T.: Cost evaluation of soft global constraints. In: Régin, J.-C., Rueher, M. (eds.) CPAIOR 2004. LNCS, vol. 3011, pp. 80–95. Springer, Heidelberg (2004)

3. Choi, K.M.F., Lee, J.H.-M., Stuckey, P.J.: A lagrangian reconstruction of genet. Artif. Intell. 123(1-2), 1–39 (2000)
4. Codognet, C., Diaz, D.: Yet Another Local Search Method for Constraint Solving. In: AAAI Fall Symposium on Using Uncertainty within Computation, Cape Cod, MA (2001)
5. Fisher, M.: The Lagrangian Relaxation Method for Solving Integer Programming Problems. Management Science 27, 1–18 (1981)
6. Fontaine, D., Michel, L., Van Hentenryck, P.: Model combinators for hybrid optimization. In: Schulte, C. (ed.) CP 2013. LNCS, vol. 8124, pp. 299–314. Springer, Heidelberg (2013)
7. Klimmek, R., Wagner, F.: A simple hypergraph min cut algorithm (1996)
8. Luenberger, D.G.: Introduction to Linear and Nonlinear Programming. Addison-Wesley, Reading (1973)
9. Nareyek, A.: Using global constraints for local search. In: Freuder, E.C., Wallace, R.J. (eds.) Constraint Programming and Large Scale Discrete Optimization. American Mathematical Society Publications, vol. 57, pp. 9–28. DIMACS (2001)
10. Sellmann, M., Fahle, T.: Constraint programming based lagrangian relaxation for the automatic recording problem. Annals of Operations Research 118(1-4), 17–33 (2003)
11. Selman, B., Kautz, H., Cohen, B.: Local search strategies for satisfiability testing. In: DIMACS Series in Discrete Mathematics and Theoretical Computer Science. American Mathematical Society Publications, vol. 26. DIMACS (1996)
12. Shang, Y., Wah, B.: A Discrete Lagrangian-Based Global-Search Method for Solving Satisfiability Problems. Journal of Global Optimization 12, 61–99 (1998)
13. Sun, T., Zhao, Q.C., Luh, P.B.: On the Surrogate Gradient Algorithm for Lagrangian Relaxation. Journal of Optimization Theory and Applications 133(3), 413–416 (2007)
14. Van Hentenryck, P.: Constraint-Based Local Search. The MIT Press, Cambridge (2005)
15. Van Hentenryck, P., Michel, L.: The Objective-CP Optimization System. In: Schulte, C. (ed.) CP 2013. LNCS, vol. 8124, pp. 8–29. Springer, Heidelberg (2013)
16. Voudouris, C., Tsang, E.: Partial constraint satisfaction problems and guided local search. In: Proc., Practical Application of Constraint Technology (PACT 1996), pp. 337–356 (1996)
17. Wah, B.W., Wu, Z.: The theory of discrete lagrange multipliers for nonlinear discrete optimization. In: Jaffar, J. (ed.) CP 1999. LNCS, vol. 1713, pp. 28–42. Springer, Heidelberg (1999)
18. Zhao, X., Luh, P.B., Wang, J.: The surrogate gradient algorithm for Lagrangian relaxation method. In: Proceedings of the 36th IEEE Conference on Decision and Control, pp. 305–310 (1997)