An adaptive CP method for TSP solving

Nicolas Isoart, Jean-Charles Régin

An adaptive CP method for TSP solving

Nicolas Isoart and Jean-Charles Régin

Université Côte d'Azur - I3S - CNRS
2000, route des Lucioles - Les Algorithmes- Euclide B
BP 121 - 06903 Sophia Antipolis Cedex - France
{isoart,regin}@univ-cotedazur.fr

Abstract. M. Sellmann showed that CP-based Lagrangian relaxation gave good results but the interactions between the two techniques were quite difficult to understand. There are two main reasons for this: the best multipliers do not lead to the best filtering and each filtering disrupts the Lagrangian multiplier problem (LMP) to be solved. As the resolution of the TSP in CP is mainly based on a Lagrangian relaxation, we propose to study in detail these interactions for this particular problem. This article experimentally confirms the above statements and shows that it is very difficult to establish any relationship between filtering and the method used to solve the LMP in practice. Thus, it seems very difficult to select a priori the best method suited for a given instance. We propose to use a multi-armed bandit algorithm to find the best possible method to use. The experimental results show the advantages of our approach.

Keywords: Lagrangian Relaxation · Filtering Algorithms · TSP

1 Introduction

Lagrangian relaxation (LR) is a relaxation method which approximates a difficult problem of constrained optimization by a simpler problem [4]. It consists in removing difficult constraints by integrating them into the objective function. It is therefore appropriate for solving problems where the constraints can be partitioned into two parts: a set of constraints that can be easily solved and a set that contains the other constraints. The constraints of the second group are moved to the objective, so it remains only constraints that are easy to solve. The satisfaction of difficult constraints is achieved by penalizing them in the objective by introducing a cost for each constraint that measures the distance to satisfaction and by multiplying this cost by a multiplier. For each set of multipliers the optimal solution of the LR is a lower bound of the optimal solution of P, the initial problem, and there is a multiplier set for which this lower bound is equal of the optimal value of P. Thus, an optimal solution of P can be found by searching for some multipliers of LR. We can also use the lower bounds produced by the LR to accelerate the search for an optimal solution of P by more traditional means such as the use of a branch-and-bound algorithm.

LR is an effective method to solve many combinatorial optimization problems. In particular, it has proven its efficiency in solving the Traveling Salesman

Problem (TSP), which consists of finding a simple cycle of minimum weight traversing all the nodes of a graph [14]. The LR of the TSP can be defined as follows. The TSP can be seen as the search for a 1-tree (i.e. a node associated with two arcs joining a spanning tree) of minimum weight such that each node of the 1-tree has a 2 degree. The search for a minimum weight 1-tree is equivalent to the search for a minimum spanning tree. This problem can be solved in polynomial time and therefore forms a constraint that we know how to solve. However, we do not know how to effectively combine it with the constraint on degrees, since the TSP is NP-Complete. The Lagrangian relaxation transfers these degree constraints into the objective. Thus for each node v, the expression $\mu_i(degree(v) - 2)$ with $\mu \geq 0$ is added to the objective, where the degree of v is expressed as the sum of the arcs taken with v as an endpoint.

The fact that for any set of multipliers μ, the optimal value of the LR is a lower bound of the optimal value of P can be efficiently used for removing some values of variables. Consider UB, an upper bound of the optimal solution of P (for example any solution of P, therefore not necessarily optimal), and $x = a$ an assignment, if for $x = a$ the optimal value of the LR is greater than UB then we can remove a from $D(x)$ since we know that $x=a$ does not belong to the optimal solution. From this idea, Fahle and Sellmann introduced the CP-based Lagrangian relaxation [22] which has been used successfully to solve many problems [23–25, 15, 17, 11, 6–8]. It consists in modeling the problem so that one or more cost based filtering algorithms can be used on the easy part of the problem. Difficult constraints are moved to the objective function and these filtering algorithms are used when looking for good multipliers.

Sellmann then became more interested in the relationship between the LR and filtering algorithms [21]. He made two important observations:

- Suboptimal multipliers can be more efficient for filtering than the optimal multipliers for the original problems. Since the search for good multipliers is important for determining good lower bound, it is necessary to do it. Thus, it makes sense to perform cost-based filtering during the optimization of Lagrangian multipliers.
- It is not clear whether domain reduction should actually take place during the optimization of the Lagrangian multipliers (i.e. as early as possible or not?), because the standard approach for the optimization of the Lagrangian multipliers are not guaranteed to be robust enough to enable a change of the underlying subproblem during the optimization.

In this paper, we study the impact of suboptimal multipliers on filtering algorithms for the TSP, and propose an approach to determine relevant ones.

We can immediately make another observation for the CP-based Lagrangian relaxation approach. The optimal solution of the initial problem is obtained by using a branch-and-bound algorithm, thus we are mainly interested in good lower bounds given by the LR and not by the optimal solutions. In addition, since we need suboptimal multipliers, we can accept to not compute (or even to not converge to) the optimal multipliers. This means that it is reasonable to consider

subgradient optimization algorithms for determining multipliers, because they give access to suboptimal multipliers and are fast.

Subgradient algorithms work in steps and reoptimize locally the multipliers according to a certain precision. Thus, determining the appropriate multipliers is the same as determining the type of precision and the number of internal multiplier recomputations we want, that is two parameters.

We start by showing that the permanent use of filtering algorithms does not lead to good results and reveals a rather disturbing property. Normally, in CP, when F_2, a filtering algorithm, is added to F_1, another filtering algorithm, all values eliminated by F_1 are also eliminated by the combination of F_1 and F_2. However, it turns out that this is not the case when filtering algorithms are called more often with CP based-Lagrangian relaxation. This does not help in determining the right level of filtering to use! In addition, we show that the number of values eliminated by filtering algorithm is, in fact, rather erratic. In other words, introducing more precision, or making more internal recalculations, is sometimes better and sometimes worse. This therefore seems to eliminate any a priori determination of the criteria. Also, we introduce a multi-armed bandit algorithm to learn the right combination of criteria during solving. The results we obtain with this method are quite good since they improve the best combination of parameters that could be obtained globally on a set of instances and are competitive with the best method for each instance.

The article is organized as follows. We recall some definitions. Then we present the subgradient algorithm we used. We then show the erratic side of the results we obtain. Next, we introduce the multi-armed bandit algorithm. Finally, we conclude. Many experimental results are given throughout this article.

2 Preliminaries

Most of the presentations come from [1, 21].

Lagrangian Relaxation. The Lagrangian relaxation (LR) procedure uses the idea of relaxing some difficult constraints by bringing them into the objective function with associated Lagrangian multipliers $\mu \geq 0$. The application of LR to a mixed integer program can be defined as follows.

$$Z = \min c \cdot x \quad \text{s.t.} \begin{cases} A_1 \cdot x \leq b_1 \\ A_2 \cdot x \leq b_2 \\ x \in X \end{cases} \longrightarrow \quad Z_{LR}(\mu) = \min c \cdot x + \mu(A_1 \cdot xn - b_1) \quad \text{s.t.} \begin{cases} A_2 \cdot x \leq b_2 \\ x \in X \end{cases}$$

We will denote by $LR(P)$ the Lagrangian relaxation of the problem P and by $LR(P, \mu)$ the LR of P associated with the multiplier set μ.

Assume that the constraint $A_1 \cdot x \leq b_1$ is difficult to solve whereas constraint $A_2 \cdot x \leq b_2$ is easy. LR moves the first one into the objective. If $A_1 \cdot x \leq b_1$ is violated then $A_1 \cdot x > b_1$ and so $d = A_1 \cdot x - b_1 > 0$. This value d measures the distance to the satisfaction of this constraint. The further away the d value

is from satisfaction, the more the solution must be penalized, so the more the value of the objective must be increased. On the contrary, the closer it is to satisfaction, the less the objective should be penalized. As we consider a problem of minimization, penalizing the objective means increasing it. This result is obtained by adding the value $(A_1x - b_1)$ in the objective. Lagrangian relaxation proposes to use a positive or zero multiplier μ for each constraint introduced in the objective. The interest of the multipliers is shown by the following property:

Property 1 *For any vector μ, the value of $Z_{LR}(\mu)$ is a lower bound of Z.*

The Lagrangian multiplier problem (LMP) consists of searching for the best multipliers. The two most popular types of methods for solving it are the subgradient and the bundle methods [12]. This second type of method converges faster than the previous one. Since we need to use suboptimal multipliers to filter we will focus our attention on the first type.

CP-based Lagrangian Relaxation. According to Sellmann [21] CP based LR consists in the following procedure: Assuming we are given a linear optimization problem that consists in the conjunction of two constraint families A and B for which an efficient filtering algorithm $prop(B)$ is known, we try to optimize Lagrangian multipliers for A and use $prop(B)$ for filtering in each Lagrangian subproblem $LR(P, \mu)$.

It is not necessary for constraints A or B to be linear (something that is not imposed in CP). We need to ensure that the relaxation we calculate for any multiplier set is a P relaxation. So we just need to be able to make sure that prop(B) remains valid when the objective becomes that of the LR.

Sellmann defined a particular consistency based on the continuous relaxation of P, but it does not matter in this paper. He also defined the following property:

Property 2 *Suboptimal multipliers can be more efficient for filtering than the optimal multipliers for the original problems.*

This property is explained by the fact that a value $x = a$ can be removed when the optimal value of $P \wedge (x = a)$ is greater than UB, a given upper bound. By considering the Lagrangian relaxation we consider the problem $LR(P)$ and not $LR(P \wedge (x = a))$ and there is no reason why the best multipliers for $LR(P)$ should also be the best for $LR(P \wedge (x = a))$.

In CP, it is also possible to express the violation of the constraint in different ways, we can also decide not to measures the distance to the violation. Fontaine et al. have proposed to avoid counting any value of a relaxed constraint when it is satisfied [11]. This is an interesting idea, but since we will relax only equality constraints we will not detail it here.

TSP model. The TSP consists of the searching for an Hamiltonian path whose the sum of the cost of its edges is minimum.

The model we use is based on the weighted circuit constraint (WCC) [5] with some additional structural constraints [2]. The circuit constraint is based on the famous Held and Karp Lagrangian relaxation of the TSP [14]:

A 1-tree of a graph G is formed by a node x, two edges having x as an extremity and a spanning tree of $G - x$ (the graph G in which x has been removed). Held and Karp proposed to represent the TSP as the search for a 1-tree whose all vertices have a degree two and whose sum of the costs of the edges it contains is minimum. Searching for a minimum 1-tree is an easy task because it is related to the search for a minimum spanning tree. However, the constraints on the degree modify the complexity of the problem. Held and Karp proposed to use the Lagrangian relaxation on these constraints (the degree of a node x is expressed as the sum of the arcs taken with x as an endpoint).

Decomman et al. [9] tested different models and concluded that the weighted circuit constrained gave the best results for the TSP.

Fages et al. [10] tested different strategies and concluded that three strategies gave similar results that are better than the others. Among them, the best strategy with the additional structural constraints is LCFirstMinReplacementCost. It consists in selecting the edges by their increasing replacement costs [5] with the LCFirst policy, which keeps one of the two extremities of the last branching edge and selects the edges from the neighborhood of the kept node by their increasing replacement costs.

It can therefore reasonably be considered that the WCC constraint with structural side constraints used in conjunction with the LCFirstMinReplacementCost strategy is the state of the art of TSP modeling by the CP-based Lagrangian Relaxation.

Experiments. The algorithms have been implemented in Java 11 in a locally developed CP solver. The experiments were performed on a Windows 10 machine using an Intel Core i7-3930K CPU @ 3.20 GHz and 64 GB of RAM. The reference instances are from the TSPLib [20], a library of reference graphs for the TSP and the set of instances is the same as in [10]. All instances considered are symmetrical graphs. The name of each instance is suffixed by its number of nodes.

3 Subgradient algorithm

As mentioned above, we are trying to calculate multipliers that allow filtering algorithms to prune values. Also, we decided to base our study on the search for multipliers by the subgradient method.

We need to have a fine control of the subgradient method in order to be able to study the relationship between multipliers and filtering algorithms. This is why we propose a particular calculation of multipliers that is strongly inspired by the Beasley algorithm [4] which is one of the most widely used. This algorithm is depicted in Algorithm 1.

In order to be able to measure the impact of subgradient optimization, we propose to simplify this algorithm, in particular the different loops, and to make it parametric. The algorithm will be used with a branch-and-bound procedure, so we do not need to find the optimal multipliers.

Algorithm 1: subgradientSolve algorithm of Beasley

```
subgradientSolve(LR(P), Z_ub)
    π ← 2 // subgradient agility
    k ← 0; noImprovedCount ← 0; ∀r ∈ R : μ_r^0 ← 0
    Z_max ← −∞ // best lower bound so far
    do
        x^k ← solve LR(P, μ) to optimality
        Z^k ← obj(x^k) + Σ_{r∈R} μ_r^k obj_r(x^k)        // optimal value of LR(P)
        Δ^k ← π(Z_ub − Z^k) / Σ_{r∈R} (obj_r(x^k))^2      // compute step
        ∀r ∈ R : μ_r^{k+1} ← max(0, μ_r^k + Δ^k × obj_r(x^k))   // update multipliers
        if Z^k > Z_max then
            Z_max ← Z^k; noImproveCount ← 0
            if Z_max = Z_ub then return Z_max    //the optimum has been found
        else noImproveCount ← noImproveCount + 1
        if noImproveCount > 30 then
            π ← π/2; noImproveCount ← 0
        k ← k + 1
    while π > 0.005
    return Z_max
```

After many tests, we propose Function parametrizedSubgradient (See Algorithm 2) which has only two simple loops. The main loop (variable i) deals with the subgradient agility: at each iteration the parameter is divided by 4. The number of iterations is given by the n parameter. Inside this loop, so for an agility value, multipliers are calculated and the filtering algorithms are called, via Funtion runPropagation of the solver. It is during the calculation of these multipliers that a second loop is used (variable j). The content of this loop is a classical application of the subgradient optimization. The number of iterations performed is defined by the m parameter. It is important to note that during these calculations the filtering algorithms are not called. By using the m value we can determine when to call the filtering. If $m = 1$ then the filtering will be called systematically after calculating multipliers, whereas with a higher value we will do calculations without filtering, so without changing the problem for which the multipliers are calculated.

The subgradient algorithm (FLR) used by Fages et al [10] in their experiments corresponds to the values of the parameters $n = 5$ and $m = 30$ of Algorithm 2 and makes the agility different since it uses the following update formula: $\pi \leftarrow \pi/\beta$; $\beta \leftarrow \beta/2$ with $\beta = 1/2$ at initialization. It should also be noted that Fages et al. repeats the call to the algorithm as long as the lower bound of the 1-tree is increased, which we do not do, as no experiment has shown a significant gain with this additional repetition.

Table 1 shows that our approach produces much better results. We reproduce here the best combination of pair (n, m). The gain potential is quite high since we gain on average a factor of 5.5 in time and 9 in number of backtracks if the best combination of parameters is found.

We therefore propose to focus on the determination of effective parameters.

Algorithm 2: PARAMETERIZEDSUBGRADIENT algorithm

```
PARAMETERIZEDSUBGRADIENT(P, Z_ub, μ, n, m)
    π ← 2 // subgradient agility
    k ← 0
    ∀r ∈ R : μ_r^0 ← μ_r // We start with the current values of multipliers
    Z_max ← COMPUTELOWERBOUND(LR(P), μ)
    for each i = 1..n do
        for each j = 1..m do
            x^k ← solve LR(P, μ) to optimality
            Z^k ← obj(x^k) + Σ_{r∈R} μ_r^k obj_r(x^k) // optimal value of LR(P)
            Δ^k ← π(Z_ub − Z^k) / Σ_{r∈R} (obj_r(x^k))^2 // compute step
            ∀r ∈ R : μ_r^{k+1} ← max(0, μ_r^k + Δ^k × obj_r(x^k)) // update multipliers
            if Z^k > Z_max then Z_max ← Z^k
            if Z_max = Z_ub then return x^k
            k ← k + 1
        RUNPROPAGATION(P, x^{k−1}, Z_ub, μ) // trigger the filtering algorithms
        if solver failed then return nil
        π ← π/4
    return nil
```

Instance	FLR time (ms)	FLR #bk	parameterized time	parameterized #bk
gr96	3113	1372	931	152
rat99	278	46	185	12
kroA100	7305	3726	1792	830
kroB100	23181	10812	1489	622
kroC100	4451	2070	592	90
kroD100	778	240	283	46
kroE100	5604	2316	2684	1112
eil101	337	74	176	18
pr107	9	10	873	90
gr120	1791	578	815	88
pr124	2387	582	1325	374
bier127	728	180	728	88
ch130	10243	3682	1586	394
pr136	160126	48370	68198	10736
gr137	13664	4208	2225	618
pr144	1892	316	941	62
ch150	12350	3514	3110	574
kroA150	63307	17526	5865	1744
kroB150	1194191	319360	335182	79648
brg180	56323	267004	428	30
rat195	732018	178312	78353	13550
d198	93713	24048	32374	2184
kroB200	1393679	288336	134291	16522
gr202	7073	1906	2492	444
pr264	7194	278	6629	326
mean	151829	47155	27342	5214
geo mean	8681	2761	2734	426
sum	3795735	1178866	683457	130354

Table 1. Comparison between LR based on FLR's subgradient algorithm and LR based on PARAMETERIZEDSUBGRADIENT algorithm; #bk stands for the number of backtracks

The subgradient algorithm (FLR) used by Fages et al [10] in their experiments corresponds to the values of the parameters $n = 5$ and $m = 30$ of Algorithm 2 and makes the agility different since it uses the following update formula: $\pi \leftarrow \pi/\beta$; $\beta \leftarrow \beta/2$ with $\beta = 1/2$ at initialization. It should also be noted that

Fages et al. repeats the call to the algorithm as long as the lower bound of the 1-tree is increased, which we do not do, as no experiment has shown a significant gain with this additional repetition.

Table 1 shows that our approach produces much better results. We reproduce here the best combination of pair (n, m). The gain potential is quite high since we gain on average a factor of 5.5 in time and 9 in number of backtracks if the best combination of parameters is found.

We therefore propose to focus on the determination of effective parameters.

4 LR and filtering

The parameterizedSubgradient algorithm allow us to study the relationship between the calculation of multipliers and filtering by defining the parameters n and m and by observing the solving time and the number of backtracks obtained to solve the instances of the TSP problems we considered. After some tests it became clear that we can restrict the values of the parameters n and m to the set of values $\{6, 9, 12, 15\}$. This means that 16 pairs are possible. Surprisingly, values below 6 do not seem interesting, as well as values above 15 (we often find $n = 5$ and $m = 30$ in the literature).

Consider instance ch150 of the TSPLib. If Algorithm 2 with $n = 6$ and $m = 9$ is used in conjunction with a static strategy (the arcs are assigned according to the decreasing value of their cost) then 1778 backtracks are needed to find the optimal solution and to prove the optimality. If the filtering algorithms are used all the time, that is when Function RUNPROPAGATION is called after each computation of multipliers in Algorithm 2, then 1868 backtracks. This result shows that **using systematically the filtering algorithms can degrade the performance in term of pruned values!** (The time increases from 4,011 to 12,504 ms). This is counter-intuitive (although explicable in this case) because usually when you add one filter to another filter you potentially eliminate more values.

The question now is: can a clear relationship between the parameters n and m and solving time and the number of backtracks be defined?

First, we can compute for each pair (n, m) the mean and geometric mean for all the instances. The following table does not show any particular trend:

n	m	mean	geo	n	m	mean	geo
6	6	49987	4100	12	6	30628	3775
6	9	33765	3569	12	9	25195	3690
6	12	33517	3804	12	12	28061	3994
6	15	46999	4037	12	15	29833	3917
9	6	45493	4084	15	6	15134	3268
9	9	23838	3166	15	9	22440	4529
9	12	30855	3775	15	12	20268	4095
9	15	32782	3622	15	15	21554	3847

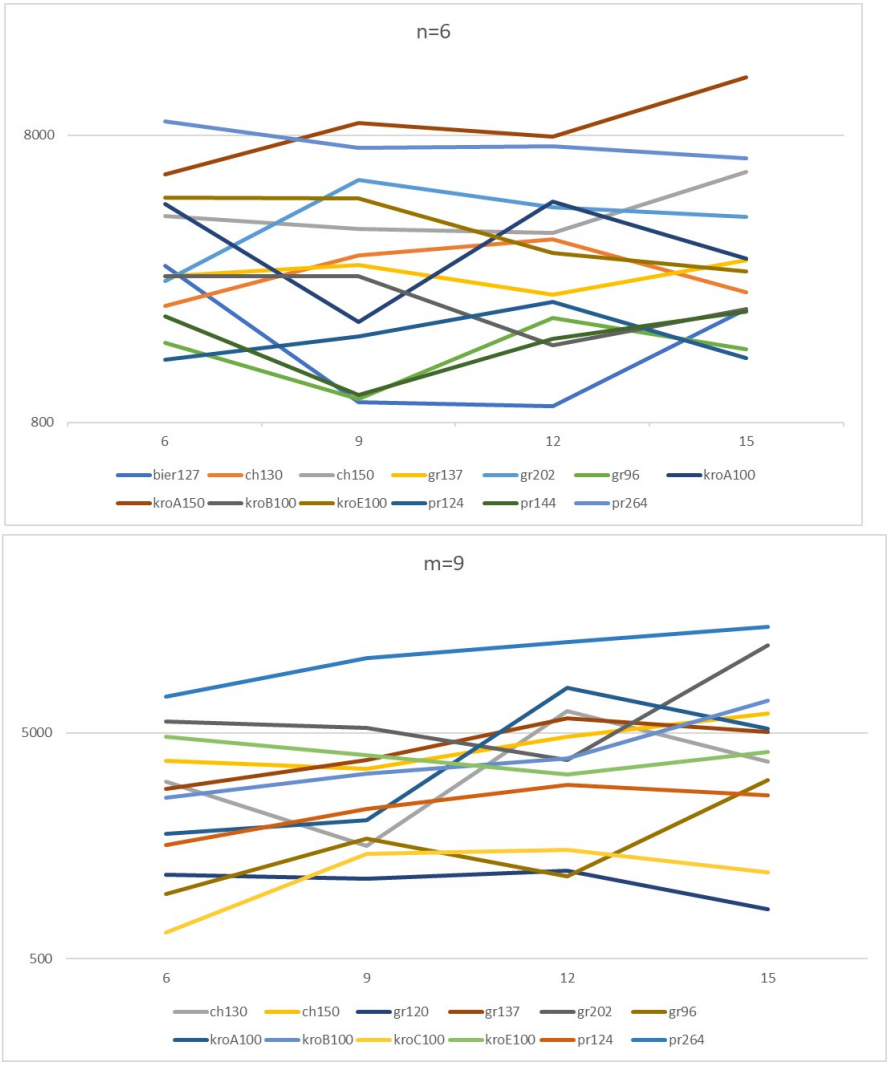

Fig. 1. Solving times (in ms with a logarithmic scale) for different instances when: (top) $n = 6$ and $m \in \{6, 9, 12, 15\}$; (bottom) $n \in \{6, 9, 12, 15\}$ and $m = 9$.

We can also study the impact of some parameters problem by problem (See Fig. 1). Again, no particular behavior seems to be describable.

The best pair value for each problem is given in Table 2. All pairs appear to be the best combination for at least one instance. All these observations lead us to deduce that **the behavior of the CP-based Lagrangian relaxation seems erratic.**

	time (ms)		#bk		best	
Instance	min	max	min	max	n	m
gr96	931	4855	152	1320	15	12
rat99	185	305	12	26	9	6
kroA100	1792	7949	830	2610	6	9
kroB100	1489	6945	622	1914	6	12
kroC100	592	2447	90	376	12	12
kroD100	283	673	46	72	6	12
kroE100	2684	9796	1112	3818	6	15
eil101	176	354	18	50	12	15
pr107	873	1848	90	186	6	12
gr120	815	2024	88	358	15	15
pr124	1325	3932	374	530	6	6
bier127	728	5124	88	1042	9	15
ch130	1586	6230	394	1414	9	9
pr136	68198	134391	10736	25898	15	9
gr137	2225	7310	618	1458	6	12
pr144	941	2292	62	246	9	12
ch150	3110	14050	574	2186	12	6
kroA150	5865	22593	1744	3542	6	6
kroB150	335182	901294	79648	138884	9	9
brg180	428	2010	30	3106	6	15
rat195	78353	243035	13550	36710	9	6
d198	32374	200688	2184	25566	15	15
kroB200	134291	318964	16522	35638	15	6
gr202	2492	12224	444	1164	6	6
pr264	6629	16866	326	508	6	15
mean	27342	77128	5214	11545		
geo mean	2734	9086	426	1507		
sum	683547	1928199	130354	288622		

Table 2. Min and max results and best pairs of parameters for each instance.

5 Multi-armed bandit approach

Since we cannot determine which pair (n, m) will lead to the most value deletion in Function PARAMETERIZEDSUBGRADIENT, we propose a Multi-Armed Bandit (MAB) approach, similar to the one proposed by Palmieri et al. [18], to determine a good pair (n, m).

The Multi-Armed Bandit selector is based on a model defined on a set of k arms and a set of rewards $R_i(j)$, where $R_i(j)$ is the reward delivered when an arm i has been chosen at time j. A reward reflects the performance of choosing that arm. The selection value is based on this reward and the sequence of previous trials. Usually, the arm having the largest selection value is selected. Two rules should be respected: if a bad choice is made then the selection value should prevent us to make this choice at the next step; and if we made good choice then the selection should help us to make the same choice.

In our case, each pair (n, m) corresponds to an arm. Since $n \in \{6, 9, 12, 15\}$ and $m \in \{6, 9, 12, 15\}$ there are 16 possibles pairs and so 16 arms. The reward function is related to the number of deleted values, which correspond to our goal. We propose to use the UCB1 policy defined in [3], which selects the arm i that maximizes $a(i) = \overline{R}_i + \sqrt{\frac{2ln(s)}{s_i}}$, where s is the current number of selection, s_i the number of times i has been selected and $\overline{R}_i$ is the mean of the past rewards

of the i arm. This policy prefers the most rewarded arm but also biases the selection toward less frequently selected arms (this bias factor increases along the iterations).

The main difficulty is the definition of the reward function. We adapt the one of Gagliolo and Schmidhuber [13] which is designed for resource allocation and defined by: $\frac{\ln(t_{\max})-\ln(t_i)}{\ln(t_{\max})-\ln(t_{\min})}$, where $t_{\max}$ and $t_{\min}$ are respectively the maximum and minimum solving time and t_i is the time for solving problem i. The logarithms help to moderate extreme cases.

Experimentally, we obtained the best results with the reward function:

$$R_i = \frac{\ln(p_{\max}) - \ln(p_i)}{\ln(p_{\max}) - \ln(p_{\min})} \tag{1}$$

where

- p_i is the number of pruned values if it is greater than 0; otherwise it is equal to p_{max}, in order to obtain a reward equals to 0.
- $\overline{p}$ is the mean of the computed p_i.
- $p_{max} = 10\ \overline{p}$
- $p_{min} = \overline{p}/10$

The extreme values p_{min} and p_{max} are defined in relation to the mean because fewer and fewer values are deleted as variables are instantiated during the search. Factor 10 was empirically determined.

The detailed results obtained by this algorithm are given in Table 3. The best and worst pairs are considered per instance and not globally.

Fig. 2 compares the number of backtracks for the parameter pair that gives the best results on average (i.e. $n = 6$ and $m = 9$) with the MAB approach. Results are given in term of ratio. The gain is clear (a factor 1.5 for the number of backtracks). However, the solving times gain according to the best pair for each instance is weaker (around 10%; See Fig. 3). It is important to note that we know that the pair $(6, 9)$ gives the best results only after we have done all the experiments. This pair is the best for this given set of instances and it seems difficult to generalize this result. The MAB approach does not make any assumption a priori and can be used for any set of instances. Thus, being able to obtain results improving the best possible global method without making any assumption is quite interesting.

The difference in gain between the number of backtracks and the solving time shows that the MAB approach is too focused on the number of pruned values and sometimes filtering is not necessary. The combination of solving time and pruned values is not easy to manage because giving weight to time locally causes the bandit to select more values with a lower filtering potential. We have not found a satisfactory solution combining these two aspects.

Finally, Table 4 compares this part with the LR proposed in Fages et al. [10] which is the current state of the art. Except for the pr107 instance which is resolved very quickly with both approaches, a factor of 4.7 is gained for the average time and 7.3 for the average number of backtracks with the MAB approach.

			Ratios w.r.t. best pair		Ratios w.r.t. worst pair	
Instance	time	#bk	time	#bk	time	#bk
gr96	1454	588	0.6	0.3	3.3	2.2
rat99	172	24	1.1	0.5	1.8	1.1
kroA100	5830	2592	0.3	0.3	1.4	1.0
kroB100	2795	1088	0.5	0.6	2.5	1.8
kroC100	555	128	1.1	0.7	4.4	2.9
kroD100	302	50	0.9	0.9	2.2	1.4
kroE100	6440	2700	0.4	0.4	1.5	1.4
eil101	197	42	0.9	0.4	1.8	1.2
pr107	957	100	0.9	0.9	1.9	1.9
gr120	994	220	0.8	0.4	2.0	1.6
pr124	1763	400	0.4	0.2	2.9	2.6
bier127	1327	270	1.0	1.4	3.0	2.0
ch130	2797	672	0.6	0.6	2.2	2.1
pr136	75592	19376	0.9	0.6	1.8	1.3
gr137	2268	512	1.0	1.2	3.2	2.8
pr144	1354	196	0.7	0.3	1.7	1.3
ch150	4814	1022	0.6	0.6	2.9	2.1
kroA150	9867	2134	0.6	0.8	2.3	1.7
kroB150	344632	78552	1.0	1.0	2.6	1.8
brg180	505	54	0.8	0.6	4.0	57.5
rat195	110230	18474	0.7	0.7	2.2	2.0
d198	59912	9070	0.5	0.2	3.3	2.8
kroB200	156220	23108	0.9	0.7	2.0	1.5
gr202	2339	336	1.1	1.3	5.2	3.5
pr264	9302	508	0.7	0.6	1.8	1.0
mean	32105	6489	0.9	0.8	2.4	1.8
geo mean	3755	741	0.7	0.6	2.4	2.0
sum	802618	162216	1.2	0.8	2.4	1.8

Table 3. Multi-arm bandit approach results.

	FLR		MAB			
	time	#bk	time	#bk	time ratio	#bk ratio
gr96	3113	1372	1454	588	2.1	2.3
rat99	278	46	172	24	1.6	1.9
kroA100	7305	3726	5830	2592	1.3	1.4
kroB100	23181	10812	2795	1088	8.3	9.9
kroC100	4451	2070	555	128	8.0	16.2
kroD100	778	240	302	50	2.6	4.8
kroE100	5604	2316	6440	2700	0.9	0.9
eil101	337	74	197	42	1.7	1.8
pr107	9	10	957	100	0.0	0.1
gr120	1791	578	994	220	1.8	2.6
pr124	2387	582	1763	400	1.4	1.5
bier127	728	180	1327	270	0.5	0.7
ch130	10243	3682	2797	672	3.7	5.5
pr136	160126	48370	75592	19376	2.1	2.5
gr137	13664	4208	2268	512	6.0	8.2
pr144	1892	316	1354	196	1.4	1.6
ch150	12350	3514	4814	1022	2.6	3.4
kroA150	63307	17526	9867	2134	6.4	8.2
kroB150	1194191	319360	344632	78552	3.5	4.1
brg180	56323	267004	505	54	111.5	4944.5
rat195	732018	178312	110230	18474	6.6	9.7
d198	93713	24048	59912	9070	1.6	2.7
kroB200	1393679	288336	156220	23108	8.9	12.5
gr202	7073	1906	2339	336	3.0	5.7
pr264	7194	278	9302	508	0.8	0.5
mean	151829	47155	32105	6489	4.7	7.3
geo mean	8681	2761	3755	741	2.3	3.7
sum	3795735	1178866	802618	162216	4.7	7.3

Table 4. Comparison between FLR and Multi-arm bandit.

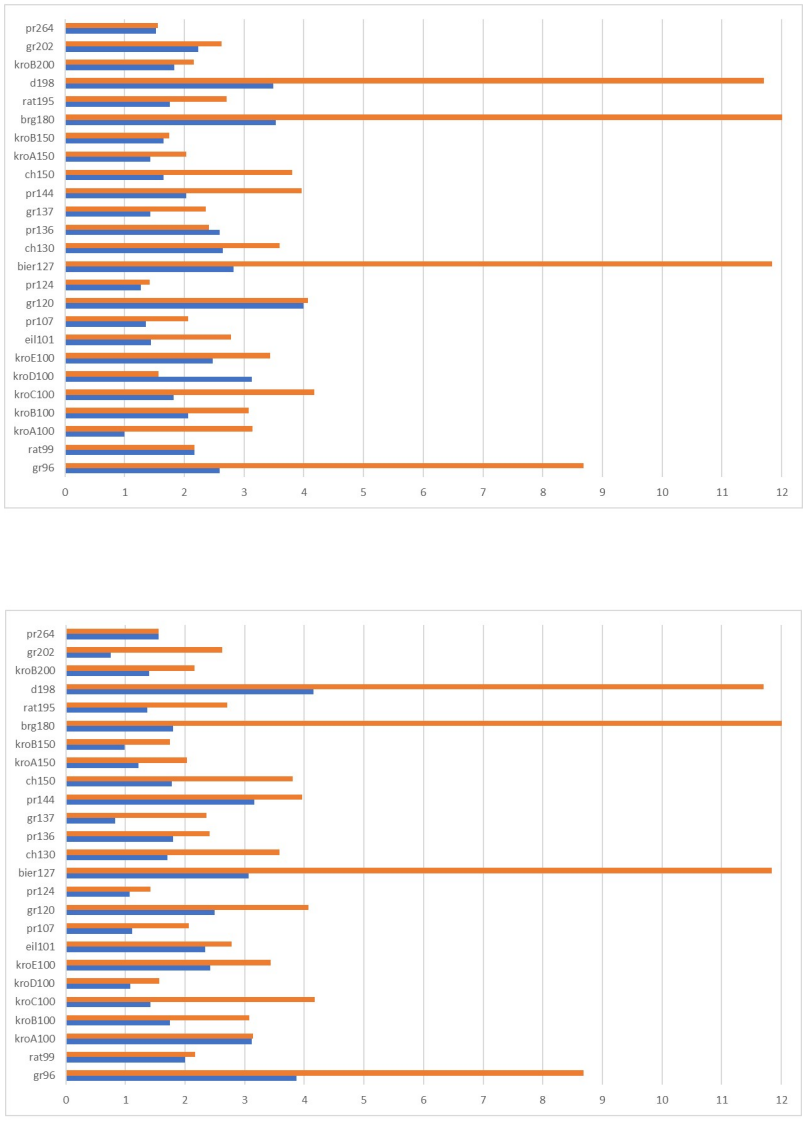

Fig. 2. Comparison of number of backtracks ratio between best and worst pairs and: (top) n=6 and m=9 (in blue); (bottom) the multi-armed bandit approach (in blue).

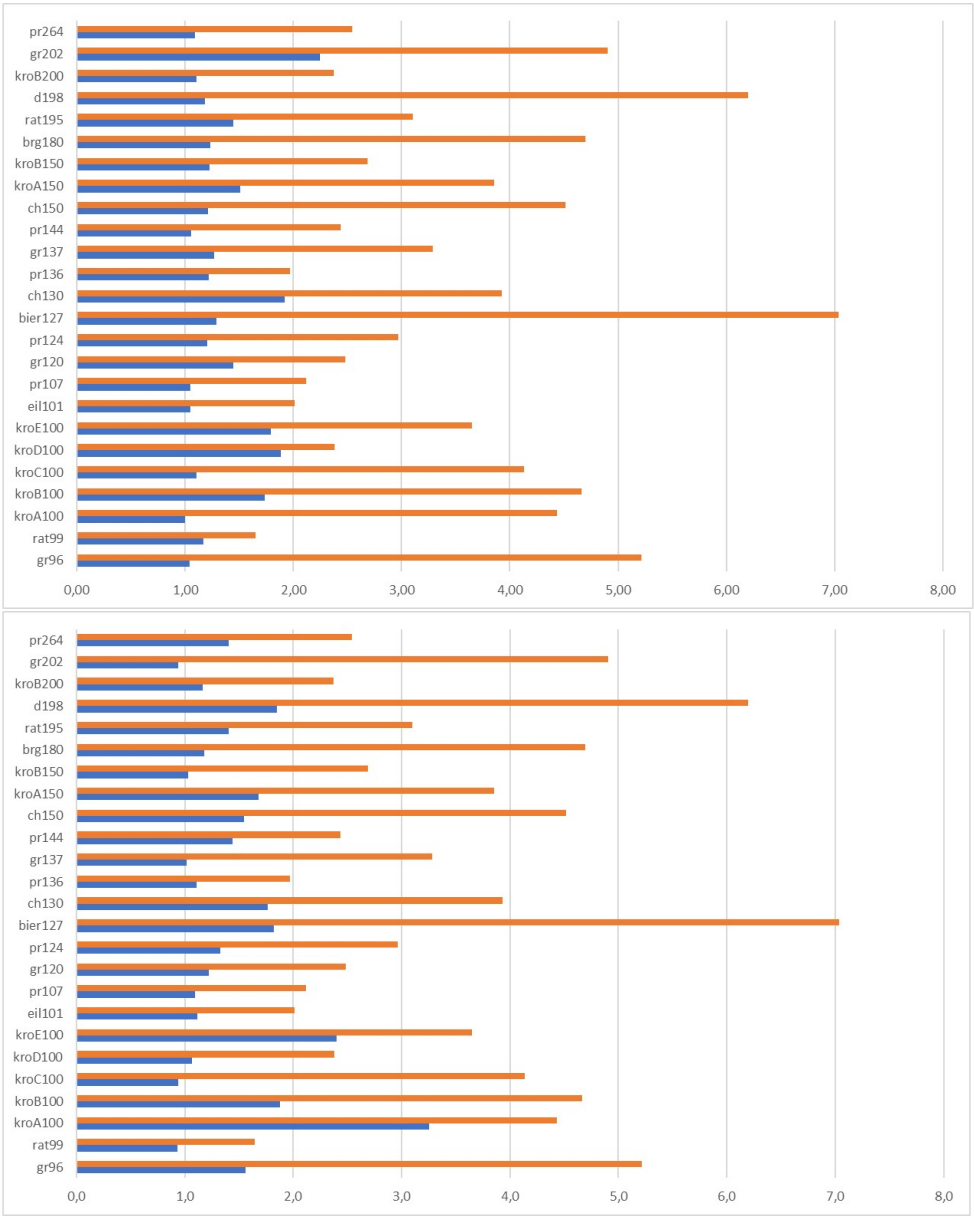

Fig. 3. Comparison of time ratio between the best and worst pairs and: (top) n=6 and m=9 (in blue); (bottom) the multi-armed bandit approach (in blue).

6 Related work

The best set of parameters for the subgradient algorithm could also have been determined with a sampling method similar to Parallel Search Strategy [18] which aims to determine a priori the best search strategy. This method proposes to decompose the initial problem into a large number of subproblems consistent with the propagation, as does the Embarrassingly Parallel Search (EPS) method [19, 16]. Then, it proceeds by sampling: it randomly draws a set of subproblems. Then, these subproblems are solved in parallel by setting a timeout corresponding to twice the time of the best method in order to limit the time spent with "bad" strategies. A Wilcoxon test is finally applied to eliminate the statistically worse strategies. All remaining strategies being equivalent, one is chosen that will be used by EPS to solve the other subproblems.

This approach is difficult to implement in our case because of the large number of subproblems it requires. Consider we have k methods to compare and we set a factor of 2 as timeout. With a confidence level of 95% and sample size equal to 30, which is not a good value in general but could be fine for our purpose, and if you accept to spend t % of the solving time in the selection of the best method then it means that the minimum number of elements in the population should be: $pop = \frac{2 \times 30 \times k}{t}$. For $t = 3\%$ and $k = 16$ we have $pop = 32,000$. Unfortunately, it requires a lot of time to decompose some TSP instances into 32,000 subproblems. For instance, the decomposition of kroB150 in more than 30,000 subproblems requires more than 100s, whereas the solving time is around 350s. This prevent us from using this method in practice for a lot of instances or a new way to decompose the instances should be found.

The MAB approach can be more easily combined with other approaches. Here we use the bandit with Lagrangian relaxation, but we could also make a bandit with the choice of strategy or a bandit linked to another part of the model without any extra cost brought by the bandit algorithm. This is more complicated to do with sampling because each sampling takes a certain amount of time. In addition, the sampling combinations also require increasing the sampling size to stay within acceptable confidence intervals.

7 Conclusion

In this article we are interested in the CP-based Lagrangian relaxation and more particularly in the relationship between Lagrangian multiplying and filtering algorithms. We proposed a parameterized subgradient algorithm, thanks to which we compared Sellmann's results and observed that the relationship between multipliers and filtering seems erratic. We then used an approach based on a multi-armed bandit algorithm to calculate the best combination of parameters for each instance. Compared to the best choice that can be made a priori, the gain in number of backtracks is high, that in time a little less. Compared to the state of the art, the improvements are strong since we gain on average almost a factor of 5 in time and more than a factor of 7 in the number of backtracks.

8 Acknowlegments

This work was supported by the Agence Nationale de la Recherche (ANR) through the MULTIMOD project under the reference ANR-17-CE22-0016.

References

1. Ahuja, R., Magnanti, T., Orlin, J.: Network Flows. Prentice Hall (1993)
2. Anonymous: Integration of structural constraints into TSP models. Submitted to this Conference **1**, 1–2 (2019)
3. Auer, P., Cesa-Bianchi, N., Fischer, P.: Finite-time analysis of the multiarmed bandit problem. Machine Learning **47**(2-3), 235–256 (2002)
4. Beasley, J.E.: Lagrangian Relaxation, chap. 6, pp. 243–303. John Wiley & Sons, Inc., New York,NY, USA (1993)
5. Benchimol, P., van Hoeve, W.J., Régin, J., Rousseau, L., Rueher, M.: Improved filtering for weighted circuit constraints. Constraints **17**(3), 205–233 (2012)
6. Bergman, D., Cire, A.A., van Hoeve, W.J.: Improved constraint propagation via lagrangian decomposition. In: International Conference on Principles and Practice of Constraint Programming. pp. 30–38. Springer (2015)
7. Cambazard, H., Fages, J.G.: New filtering for atmostnvalue and its weighted variant: A lagrangian approach. Constraints **20**(3), 362–380 (2015)
8. Demassey, S.: Compositions and hybridizations for applied combinatorial optimization (2017), Habilitation Ã Diriger des Recherches
9. Ducomman, S., Cambazard, H., Penz, B.: Alternative filtering for the weighted circuit constraint: Comparing lower bounds for the TSP and solving TSPTW. In: Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence, February 12-17, 2016, Phoenix, Arizona, USA. pp. 3390–3396 (2016)
10. Fages, J., Lorca, X., Rousseau, L.: The salesman and the tree: the importance of search in cp. Constraints **21**(2), 145–162 (2016)
11. Fontaine, D., Michel, L.D., Hentenryck, P.V.: Constraint-based lagrangian relaxation. In: Principles and Practice of Constraint Programming - 20th International Conference, CP 2014, Lyon, France, September 8-12, 2014. Proceedings. pp. 324–339 (2014)
12. Frangioni, A.: Generalized bundle methods. SIAM J. on Optimization **13**(1), 117–156 (2002)
13. Gagliolo, M., Schmidhuber, J.: Learning dynamic algorithm portfolios. Annals of Mathematics and Artificial Intelligence **47**(3-4), 295–328 (2006)
14. Held, M., Karp, R.M.: The traveling-salesman problem and minimum spanning trees: Part ii. Mathematical Programming **1**(1), 6–25 (Dec 1971)
15. Khemmoudj, M.O.I., Bennaceur, H., Nagih, A.: Combining arc-consistency and dual lagrangean relaxation for filtering csps. In: International Conference on Integration of Artificial Intelligence (AI) and Operations Research (OR) Techniques in Constraint Programming. pp. 258–272. Springer (2005)
16. Malapert, A., Régin, J., Rezgui, M.: Embarrassingly parallel search in constraint programming. J. Artif. Intell. Res. (JAIR) **57**, 421–464 (2016)
17. Menana, J.: Automates et programmation par contraintes pour la planification de personnel. Ph.D. thesis, Université de Nantes (2011)
18. Palmieri, A., Régin, J., Schaus, P.: Parallel strategies selection. In: Principles and Practice of Constraint Programming - 22nd International Conference, CP 2016, Toulouse, France, September 5-9, 2016, Proceedings. pp. 388–404 (2016)

19. Régin, J.C., Rezgui, M., Malapert, A.: Embarrassingly parallel search. In: Principles and Practice of Constraint Programming. pp. 596–610. Springer (2013)
20. Reinelt, G.: TSPLIB–a traveling salesman problem library. ORSA Journal on Computing **3**(4), 376–384 (1991)
21. Sellmann, M.: Theoretical foundations of cp-based lagrangian relaxation. In: Principles and Practice of Constraint Programming - CP 2004, 10th International Conference, CP 2004, Toronto, Canada, September 27 - October 1, 2004, Proceedings. pp. 634–647 (2004)
22. Sellmann, M., Fahle, T.: Constraint programming based lagrangian relaxation for the automatic recording problem. Annals OR **118**(1-4), 17–33 (2003)
23. Shang, Y., Wah, B.W.: A discrete lagrangian-based global-search method for solving satisfiability problems. J. Global Optimization **12**(1), 61–99 (1998)
24. Wah, B.W., Wu, Z.: Discrete lagrangian methods for designing multiplierless two-channel PR-LP filter banks. VLSI Signal Processing **21**(2), 131–149 (1999)
25. Wah, B.W., Wu, Z.: The theory of discrete lagrange multipliers for nonlinear discrete optimization. In: Principles and Practice of Constraint Programming - CP'99, 5th International Conference, Alexandria, Virginia, USA, October 11-14, 1999, Proceedings. pp. 28–42 (1999)