Differentiation of Black-Box Combinatorial Solvers

Marin Vlastelica Pogančić[1]*, Anselm Paulus[1]*, Vít Musil[2], Georg Martius[1], Michal Rolínek[1]
[1] Max-Planck-Institute for Intelligent Systems Tübingen, Germany
[2] Università degli Studi di Firenze, Italy
{marin.vlastelica, anselm.paulus}@tuebingen.mpg.de
{georg.martius, michal.rolinek}@tuebingen.mpg.de
vit.musil@unifi.it

Abstract

Deep learning has been shown to be a powerful method for feature extraction, but falls short in solving certain problems of combinatorial nature. Such problems can be solved efficiently given dedicated solvers. We present a method that fuses deep learning with black-box combinatorial solvers in an end-to-end manner, allowing efficient gradient propagation through the solvers. We also provide both experimental and theoretical backing of our approach.

1 Introduction

Deep learning approaches excel at learning rich feature representations, most notably in the area of computer vision. Such representations have proven to be very useful in various problems, such as object recognition, object detection, reinforcement learning and more.

The flexibility and composability of neural networks invite designing new types of building blocks. This is particularly interesting when the building blocks can easily solve relevant problems that are difficult for standard networks. Examples of such relevant problems are SAT-solving and constrained optimization. The corresponding building blocks SATNet and OptNet were introduced in [1] and [2].

Our mission is to introduce building blocks capable of solving discrete optimization problems. The goal is to leverage decades of successful research that led to efficient graph algorithms (for problems such as SHORTEST PATH [3], MIN-COST-PERFECT-MATCHING, *etc*. [4]) as well as strong heuristical solvers for NP-Hard problems (*e.g.*TRAVELING SALESMAN and many types of graph cuts).

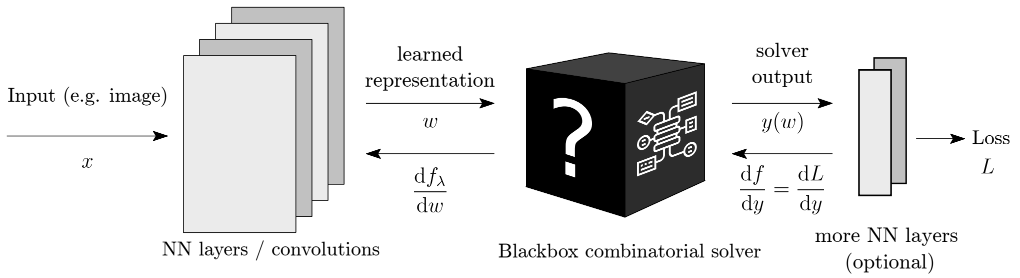

Input (e.g. image)
x
learned representation
w
$\frac{\mathrm{d}f_\lambda}{\mathrm{d}w}$
solver output
$y(w)$
$\frac{\mathrm{d}f}{\mathrm{d}y} = \frac{\mathrm{d}L}{\mathrm{d}y}$
Loss
L
NN layers / convolutions
Blackbox combinatorial solver
more NN layers (optional)

Figure 1: Architecture design enabled by Theorem 1. Blackbox combinatorial solvers can be naturally embedded into a neural network.

*These authors contributed equally.

With our method we are able to implement a backward pass for large classes of combinatorial optimization problems and the corresponding algorithms and solvers. Our main contribution is that the method is tight in the two following ways:

- It operates with **blackbox** solvers and embeds them **without any modification**. In particular, one can run *the most suitable solver* for the task and hence get the best possible performance in the forward pass. This is in contrast with usual techniques, that typically find a differentiable way of solving some relaxation and thus inevitably take a hit in performance, scalability, theoretical soundness *etc*.
- The computational cost of the introduced backward pass matches the cost of the forward pass (in particular it also amounts to one call to the solver).

We provide mathematical as well as experimental validation.

2 Method

Let us first formalize the notion of a combinatorial solver. We expect the solver to receive continuous input $w \in W \subseteq \mathbb{R}^N$ (*e.g.* edge weights of a fixed graph) and return discrete output y from some finite set Y (*e.g.* all travelling salesman tours on a fixed graph) that minimizes some cost $\mathbf{c}(w, y)$ (*e.g.* length of the tour). More precisely, the solver maps

$$w \mapsto y(w) \quad \text{such that} \quad y(w) = \arg\min_{y \in Y} \mathbf{c}(w, y). \tag{1}$$

We will restrict ourselves to objective functions $\mathbf{c}(w, y)$ that are **linear** in w for every y.

For incorporating such a combinatorial solver into a neural network, we will think of w as output of some intermediate layer and assume that y has some vector representation that is passed further down the network.

The task to solve during back-propagation is the following: we receive the gradient $\mathrm{d}L/\mathrm{d}y$ of the global loss L with respect to solver output $\hat{y}$, from that we construct the linearization around the solver output $\hat{y}$ from the forward pass as $f(y) = L(\hat{y}) + (y - \hat{y})\mathrm{d}L/\mathrm{d}y$; and are expected to return $\mathrm{d}f\big(y(w)\big)/\mathrm{d}w = (\mathrm{d}y/\mathrm{d}w) \cdot (\mathrm{d}L/\mathrm{d}y) = \mathrm{d}L/dw$; the gradient of the loss with respect to solver input w.

Since Y is finite, there are only finitely many values of $f\big(y(w)\big)$. In other words, this function of w is **piecewise constant** and the gradient is identically zero or does not exist at all (at points of jumps). This should not come as a surprise; if one does a small perturbation to edge weights of a graph, one *usually* does not change the optimal TSP tour. This has an important consequence:

> The fundamental problem with differentiating through combinatorial solvers is not the lack of differentiability; the gradient exists *almost everywhere*. However, this gradient is a constant zero and as such is useless for optimization.

Accordingly, we will *not rely* on standard techniques for gradient estimation (see [5] for a nice survey). Our strategy will rather be to compute the gradient of some $f_\lambda(w)$ which (in a precise sense) is a **continuous interpolation** of $f\big(y(w)\big)$, where $\lambda > 0$ is a parameter controlling the trade-off between "informativeness of gradient" and "faithfulness to the original function".

2.1 Construction and properties of f_λ

Before we give the exact definition of the function f_λ, we formulate several requirements on it. This will help us understand why $f_\lambda(w)$ is a reasonable replacement for $f\big(y(w)\big)$ and, most importantly, why its gradient allows to optimize function f. The simplest condition is the following.

Property A1. For each $\lambda > 0$, f_λ is continuous and piecewise affine.

The second property describes the trade-off induced by changing the value of λ. For $\lambda > 0$, we define sets W^λ_{eq} and W^λ_{diff} as the sets where $f\big(y(w)\big)$ and $f_\lambda(w)$ coincide and where they differ, *i.e.*

$$W^\lambda_{\text{eq}} = \big\{ w \in W : f_\lambda(w) = f\big(y(w)\big) \big\} \quad \text{and} \quad W^\lambda_{\text{diff}} = W \setminus W^\lambda_{\text{eq}}.$$

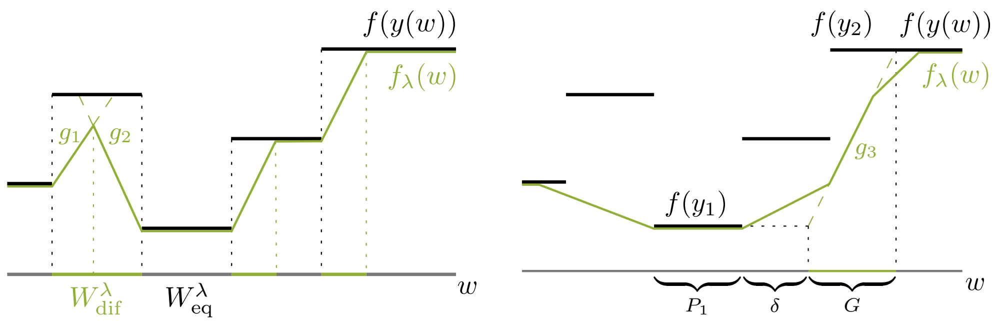


Figure 2: Continuous interpolation of a piecewise constant function. Left: f_λ for a small value of λ; the set W^λ_{eq} is still substantial and only two interpolators g_1 and g_2 are incomplete. Also, all interpolators are 0-interpolators. Right: f_λ for a high value of λ; most interpolators are incomplete and we also encounter a δ-interpolator g_3 (between y_1 and y_2) which attains the value $f(y_1)$ δ-away from the set P_1. Despite losing some local structure for high λ, the gradient of f_λ is still informative.

Property A2. The sets W^λ_{diff} are monotone in λ and they vanish as $\lambda \to 0^+$, *i.e.*

$$W^{\lambda_1}_{\text{diff}} \subseteq W^{\lambda_2}_{\text{diff}} \quad \text{for } 0 < \lambda_1 \le \lambda_2 \quad \text{and} \quad W^{\lambda}_{\text{diff}} \to \emptyset \quad \text{as } \lambda \to 0^+.$$

In other words, Property A2 tells us that λ controls the size of the set where f_λ deviates from f but where also f_λ has meaningful gradient.

In the third and final property, we want to capture the interpolation behavior of f_λ. For that purpose, we define a δ-*interpolator* of f. We say that g, defined on a set $G \subset W$, is a δ-interpolator of f between y_1 and $y_2 \in Y$, if

- g is non-constant affine function;
- the image $g(G)$ is an interval with endpoints $f(y_1)$ and $f(y_2)$;
- g attains the boundary values $f(y_1)$ and $f(y_2)$ at most δ-far away from where $f(y(w))$ does. In particular, there is a point $w_k \in G$ for which $g(w_k) = f(y_k)$ and $\operatorname{dist}(w_k, P_k) \le \delta$, where $P_k = \{w \in W : y(w) = y_k\}$, for $k = 1, 2$.

In the special case of a 0-interpolator g, the graph of g connects (in a topological sense) two components of the graph of $f\big(y(w)\big)$. In the general case, δ measures *displacement* of the interpolator (see also Fig. 2 for some examples). This displacement on the one hand loosens the connection to $f\big(y(w)\big)$ but on the other hand allows for less local interpolation which might be desirable.

Property A3. The function f_λ consists of finitely many (possibly incomplete) δ-interpolators of f on W^λ_{diff} where $\delta \le C\lambda$ for some fixed C. Equivalently, the *displacement* is linearly controlled by λ.

Before we introduce the function f_λ, we need to define a "perturbed solver" which, for given $\lambda > 0$, maps

$$w \mapsto y_\lambda(w) \quad \text{such that} \quad y_\lambda(w) = \arg\min_{y \in Y} \{\mathbf{c}(w, y) + \lambda f(y)\}. \tag{2}$$

We have then the following result.

Theorem 1. *Let $\lambda > 0$. The function f_λ defined by*

$$f_\lambda(w) = f\big(y_\lambda(w)\big) - \frac{1}{\lambda}\Big[\mathbf{c}\big(w, y(w)\big) - \mathbf{c}\big(w, y_\lambda(w)\big)\Big] \tag{3}$$

satisfies Properties A1, A2, A3.

Now, since f_λ is ensured to be differentiable, we have

$$\nabla f_\lambda(w) = -\frac{1}{\lambda}\Big[\frac{\mathrm{d}\mathbf{c}}{\mathrm{d}w}\big(w, y(w)\big) - \frac{\mathrm{d}\mathbf{c}}{\mathrm{d}w}\big(w, y_\lambda(w)\big)\Big]. \tag{4}$$

We then return ∇f_λ as a loss gradient. The only part of evaluating (4) that could potentially be computationally intensive is solving for y_λ in (2). The following proposition comes to rescue.

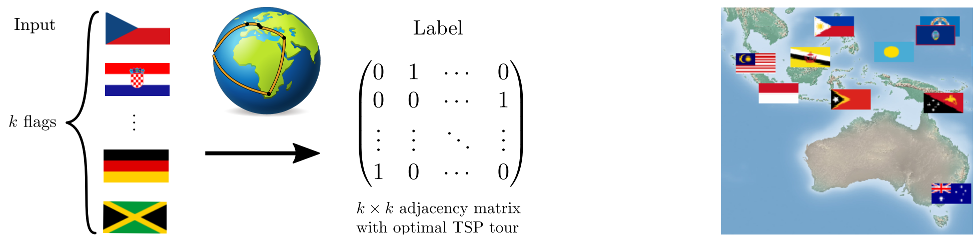

Input
k flags
Label
$\begin{pmatrix} 0 & 1 & \cdots & 0 \\ 0 & 0 & \cdots & 1 \\ \vdots & \vdots & \ddots & \vdots \\ 1 & 0 & \cdots & 0 \end{pmatrix}$
$k \times k$ adjacency matrix
with optimal TSP tour

Figure 3: The TSP(k) problem. Left: Dataset illustration. Each input is a sequence of k flags and the corresponding label is the adjacency matrix of the optimal TSP tour around the corresponding capitals. Right: Learned locations of 10 country capitals in southeast Asia and Australia, accurately recovering their true position.

Proposition 1. *Let $w \in W$ be fixed. If we set $w' = w + \lambda \frac{\mathrm{d}L}{\mathrm{d}y}(\hat{y})$, we can compute y_λ as*

$$y_\lambda(w) = \arg\min_{y \in Y} \mathbf{c}(w', y).$$

In other words, y_λ can be found by running the combinatorial solver on a different instance defined by weight vector w'. Proofs of these results, along with geometrical description of f_λ, can be found in section B.

3 Globe Travelling Salesman Problem

Problem input and output. The training dataset for problem TSP(k) consists of 10000 examples where the input for each example is a subset of k from a set of 100 country flags and the "label" is the shortest travelling salesman tour through the capitals of the corresponding countries. The optimal tour is represented by its adjacency matrix. On test time, the networks are presented unseen subsets of k country flags. We consider datasets TSP(k) for $k \in \{5, 10, 20, 40\}$

Architecture. Each of the flags is presented to a convolutional network that produces k three-dimensional representations. These vectors are projected onto the unit sphere in $\mathbb{R}^3$ which can be seen as a world map, as shown in Fig. 3. The TSP solver is presented with a matrix of pairwise distances of the k computed locations. The loss of the network is the Hamming distance between the true and the predicted TSP adjacency matrix. The architecture is expected to learn the correct representations of the flags (*i.e.* locations of their capitals on Earth, modulo rotations of the sphere).

Results. This architecture not only learns to extract the correct TSP tours but also learns the correct representations. Quantitative evidence is presented in Tab. 2, where we see that the learned locations generalize well and lead to correct TSP tours also on the test set and also on somewhat large instances (note that there are $39! \approx 10^{46}$ admissible TSP tours for $k = 40$). The baseline architecture fails to achieve anything but memorization of the training set. Additionally, we can extract the suggested locations of world capitals and compare them with reality. To that end, we present Fig. 3, where the learned locations of 10 capitals in southeast Asia are displayed.

Table 1: **Results for Globe TSP – Full Tour Accuracy.** Standard deviations are shown with respect to five restarts.

	Embedding TSP Solver		ResNet18	
k	Train	Test	Train	Test
5	$99.7 \pm 0.1\%$	$98.8 \pm 1.9\%$	$100.0 \pm 0.0\%$	$1.6 \pm 0.1\%$
10	$99.7 \pm 0.1\%$	$98.7 \pm 0.4\%$	$98.9 \pm 0.1\%$	$0.0 \pm 0.0\%$
20	$99.2 \pm 0.1\%$	$98.6 \pm 0.4\%$	$98.8 \pm 0.3\%$	$0.0 \pm 0.0\%$
40	$80.2 \pm 34.6\%$	$79.5 \pm 34.9\%$	$94.8 \pm 1.5\%$	$0.0 \pm 0.0\%$

References

[1] Po-Wei Wang, Priya L. Donti, Bryan Wilder, and Zico Kolter. SATNet: Bridging deep learning and logical reasoning using a differentiable satisfiability solver. *arXiv*, 1905.12149, 2019.

[2] Brandon Amos and J. Zico Kolter. Optnet: Differentiable optimization as a layer in neural networks. *arXiv*, 1703.00443, 2017.

[3] E. W. Dijkstra. A note on two problems in connexion with graphs. *Numer. Math.*, 1(1):269–271, December 1959.

[4] Jack Edmonds. Paths, trees, and flowers. *Canad. J. Math.*, 17:449–467, 1965.

[5] Shakir Mohamed, Mihaela Rosca, Michael Figurnov, and Andriy Mnih. Monte carlo gradient estimation in machine learning. *arXiv*, abs/1906.10652, 2019.

[6] Vladimir Kolmogorov. Blossom V: a new implementation of a minimum cost perfect matching algorithm. *Mathematical Programming Computation*, 1(1):43–67, Jul 2009.

A Additional Experiments

A.1 MNIST Min-cost Perfect Matching

Problem input and output. The training dataset for problem PM(k) consists of 10000 examples where the input to each example is a set of k^2 digits drawn from the MNIST dataset and arranged in a $k \times k$ grid. For computing the label, we consider the underlying $k \times k$ grid graph (without diagonal edges) and solve a MIN-COST-PERFECT-MATCHING problem, where edge weights are given simply by reading the two vertex digits as a two-digit number (we read downwards for vertical edges and from left to right for horizontal edges)

The optimal perfect matching is encoded by an indicator vector for the subset of the selected edges. This indicator vector is the label.

The test set consists of MNIST grid arrangements not used in the training set.

The mindset behind constructing the dataset is again that the input is not the "right representation" for the solver; this right representation (*i.e.* digit values) should be extracted via backpropagating through a composite architecture.

Architecture. The grid image is fed into a convolutional neural network which outputs a grid of vertex weights. These weights are transformed into edge weights as described above and fed into dedicated Blossom V solver [6]. The loss function is Hamming distance between solver output and the true label.

Results. Our method outperforms the ResNet18 baseline by a high margin in generalizing to unseen examples. The results show that even in the most trivial case of PM(4), the baseline is not capable of good generalization, indicated by the high test error. For cases where $k > 4$, the baseline cannot generalize at all to unseen examples, whereby our method does. Our method still does not achieve the perfect score, the reason for this is that the solutions to the problem are sometimes not unique. Therefore our method sometimes suggests one of the optimal solutions to the perfect matching problem, but a different one than the target label.

Table 2: **Preliminary Results for MNIST Min-weight perfect matching** Standard deviations are shown with respect to five restarts.

k	Train	Test
4	$96.97 \pm 0.94\%$	$92.1 \pm 1.61\%$
8	$93.25 \pm 1.63\%$	$88.6 \pm 1.69\%$

(a) Embedding PM Solver

k	Train	Test
4	$100.0 \pm 0.0\%$	$79.6 \pm 1.07\%$
8	$100.0 \pm 0.0\%$	$0.0 \pm 0.0\%$

(b) ResNet18

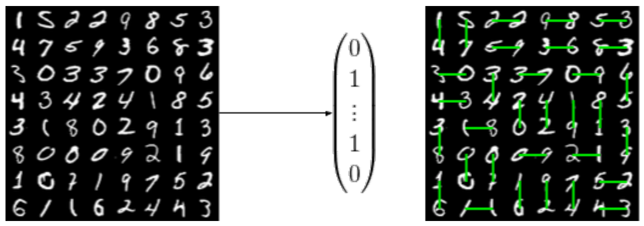


Figure 4: Each input is a full image of a $k \times k$ grid of digits drawn from the MNIST dataset (left). The output produced by the architecture is an indicator vector selecting the edges for the minimum cost perfect matching (right).

B Proofs

Proof of Proposition 1. Let us write $L = L(\hat{y})$ and $\nabla L = \frac{\mathrm{d}L}{\mathrm{d}y}(\hat{y})$, for brevity. Thanks to the linearity of $\mathbf{c}$ and the definition of f, we have

$$\mathbf{c}(w, y) + \lambda f(y) = wy + \lambda\big(L + \nabla L(y - \hat{y})\big) = (w + \lambda \nabla L)y + \lambda L - \lambda \nabla L \hat{y} = \mathbf{c}(w', y) + \mathbf{c}_0,$$

where $\mathbf{c}_0 = \lambda L - \lambda \nabla L \hat{y}$ and $w' = w + \lambda \nabla L$ as desired. The conclusion about the points of minima then follows. $\square$

Before we prove Theorem 1, we make some preliminary observations. To start with, due to the definition of the solver, we have the fundamental inequality

$$\mathbf{c}(w, y) \geq \mathbf{c}\big(w, y(w)\big) \quad \text{for every } w \in W \text{ and } y \in Y. \tag{5}$$

Observation 1. *The function $w \mapsto \mathbf{c}\big(w, y(w)\big)$ is continuous and piecewise linear.*

Proof. Since $\mathbf{c}$'s are linear and distinct, $\mathbf{c}\big(w, y(w)\big)$, as their pointwise minimum, has the desired properties. $\square$

Analogous fundamental inequality

$$\mathbf{c}(w, y) + \lambda f(y) \geq \mathbf{c}\big(w, y_\lambda(w)\big) + \lambda f\big(y_\lambda(w)\big) \quad \text{for every } w \in W \text{ and } y \in Y \tag{6}$$

follows from the definition of the solution to the optimization problem (2).

A counterpart of Observation 1 reads as follows.

Observation 2. *The function $w \mapsto \mathbf{c}\big(w, y_\lambda(w)\big) + \lambda f\big(y_\lambda(w)\big)$ is continuous and piecewise affine.*

Proof. The function under inspection is a pointwise minimum of distinct affine functions $w \mapsto \mathbf{c}(w, y) + \lambda f(y)$ as y ranges Y. $\square$

As a consequence of above-mentioned fundamental inequalities, we obtain the following two-sided estimates on f_λ.

Observation 3. *The following inequalities hold for $w \in W$*

$$f\big(y_\lambda(w)\big) \leq f_\lambda(w) \leq f\big(y(w)\big).$$

Proof. Inequality (5) implies that $\mathbf{c}\big(w, y(w)\big) - \mathbf{c}\big(w, y_\lambda(w)\big) \leq 0$ and the first inequality then follows simply from the definition of f_λ. As for the second one, it suffices to apply (6) to $y = y(w)$. $\square$

Now, let us introduce few notions that will be useful later in the proofs. For a fixed λ, W partitions into maximal connected sets P on which $y_\lambda(w)$ is constant (see Fig. 5). We denote this collection of sets by $\mathcal{W}_\lambda$ and set $\mathcal{W} = \mathcal{W}_0$.

For $\lambda \in \mathbb{R}$ and $y_1 \neq y_2 \in Y$, we denote

$$F_\lambda(y_1, y_2) = \big\{w \in W : c(w, y_1) + \lambda f(y_1) = c(w, y_2) + \lambda f(y_2)\big\}.$$

We write $F(y_1, y_2) = F_0(y_1, y_2)$, for brevity. For technical reasons, we also allow negative values of λ here.

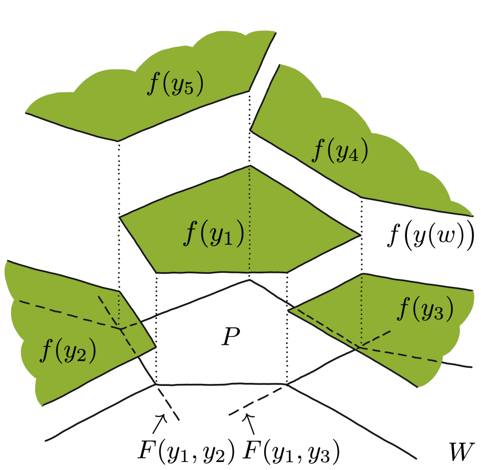
Note, that if $W = \mathbb{R}^N$, then F_λ is a hyperplane since $\mathbf{c}$'s are linear. In general, W may just be a proper subset of $\mathbb{R}^N$ and, in that case, F_λ is just the restriction of a hyperplane onto W. Consequently, it may happen that $F_\lambda(y_1, y_2)$ will be empty for some pair of y_1, y_2 and some $\lambda \in \mathbb{R}$. To emphasize this fact, we say "hyperplane in W". Analogous considerations should be taken into account for all other linear objects. The note "in W" stands for the intersection of these linear object with the set W.

Observation 4. *Let $P \in \mathcal{W}_\lambda$ and let $y_\lambda(w) = y$ for $w \in P$. Then P is a convex polytope in W, where the facets consist of parts of finitely many hyperplanes $F_\lambda(y, y_k)$ in W for some $\{y_k\} \subset Y$.*

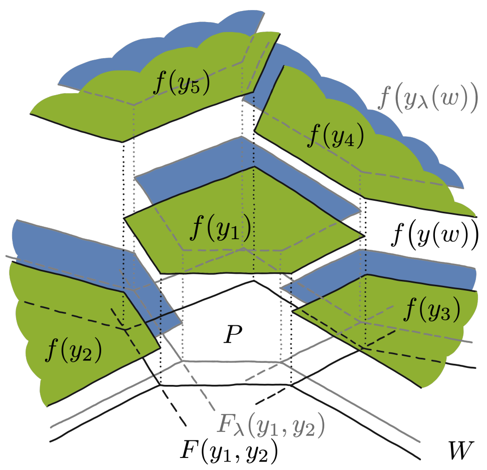
Proof. Assume that $W = \mathbb{R}^N$. The values of y_λ may only change on hyperplanes of the form $F_\lambda(y, y')$ for some $y' \in Y$. Then P is an intersection of corresponding half-spaces and therefore P is a convex polytope. If W is a proper subset of $\mathbb{R}^N$ the claim follows by intersecting all the objects with W. $\square$

Observation 5. *Let $y_1, y_2 \in Y$ be distinct. If nonempty, the hyperplanes $F(y_1, y_2)$ and $F_\lambda(y_1, y_2)$ are parallel and their distance is equal to $|\lambda| K(y_1, y_2)$, where*

$$K(y_1, y_2) = \frac{|f(y_1) - f(y_2)|}{\|y_1 - y_2\|}.$$



(a) The situation for $\lambda = 0$. We can see the polytope P on which $y(w)$ attains $y_1 \in Y$. The boundary of P is composed of segments of lines $F(y_1, y_k)$ for $k = 2, \ldots, 5$.



(b) The same situation is captured for some relatively small $\lambda > 0$. Each line $F_\lambda(y_1, y_k)$ is parallel to its corresponding $F(y_1, y_k)$ and encompasses a convex polytope in $\mathcal{W}_\lambda$.

Figure 5: The family $\mathcal{W}_\lambda$ of all maximal connected sets P on which y_λ is constant.

Proof. If we define a function $c(w) = \mathbf{c}(w, y_1) - \mathbf{c}(w, y_2) = w(y_1 - y_2)$ and a constant $C = f(y_2) - f(y_1)$, then our objects rewrite to

$$F(y_1, y_2) = \{w \in W : c(w) = 0\} \quad \text{and} \quad F_\lambda(y_1, y_2) = \{w \in W : c(w) = \lambda C\}.$$

Since c is linear, these sets are parallel and $F(y_1, y_2)$ intersects the origin. Thus, the required distance is the distance of the hyperplane $F_\lambda(y_1, y_2)$ from the origin, which equals to $|\lambda C| / \|y_1 - y_2\|$. □

As the set Y is finite, there is a uniform upper bound K on all values of $K(y_1, y_2)$. Namely

$$K = \max_{\substack{y_1, y_2 \in Y \\ y_1 \neq y_2}} K(y_1, y_2). \tag{7}$$

B.1 Proof of Theorem 1

Proof of Property A1. Now, Property A1 follows, since

$$f_\lambda(w) = \frac{1}{\lambda}\Big[\mathbf{c}\big(w, y_\lambda(w)\big) + \lambda f\big(y_\lambda(w)\big)\Big] - \frac{1}{\lambda}\mathbf{c}\big(w, y(w)\big)$$

and f_λ is a difference of continuous and piecewise affine functions. □

Proof of Property A2. Let $0 < \lambda_1 \leq \lambda_2$ be given. We show that $W_{\text{eq}}^{\lambda_2} \subseteq W_{\text{eq}}^{\lambda_1}$ which is the same as showing $W_{\text{diff}}^{\lambda_1} \subseteq W_{\text{diff}}^{\lambda_2}$. Assume that $w \in W_{\text{eq}}^{\lambda_2}$, that is, by the definition of $W_{\text{eq}}^{\lambda_2}$ and f_λ,

$$\mathbf{c}\big(w, y(w)\big) + \lambda_2 f\big(y(w)\big) = \mathbf{c}(w, y_2) + \lambda_2 f(y_2), \tag{8}$$

in which we denoted $y_2 = y_{\lambda_2}(w)$. Our goal is to show that

$$\mathbf{c}\big(w, y(w)\big) + \lambda_1 f\big(y(w)\big) = \mathbf{c}(w, y_1) + \lambda_1 f(y_1), \tag{9}$$

where $y_1 = y_{\lambda_1}(w)$ as this equality then guarantees that $w \in W_{\text{eq}}^{\lambda_1}$. Observe that (6) applied to $\lambda = \lambda_1$ and $y = y(w)$, yields the inequality "$\geq$" in (9).

Let us show the reversed inequality. By Observation 3 applied to $\lambda = \lambda_1$, we have

$$f\big(y(w)\big) \geq f(y_1). \tag{10}$$

We now use (6) with $\lambda = \lambda_2$ and $y = y_1$, followed by equality (8) to obtain

$$\begin{aligned}
\mathbf{c}(w, y_1) + \lambda_1 f(y_1) &= \mathbf{c}(w, y_1) + \lambda_2 f(y_1) + (\lambda_1 - \lambda_2) f(y_1) \\
&\geq \mathbf{c}(w, y_2) + \lambda_2 f(y_2) + (\lambda_1 - \lambda_2) f(y_1) \\
&= \mathbf{c}\big(w, y(w)\big) + \lambda_2 f\big(y(w)\big) + (\lambda_1 - \lambda_2) f(y_1) \\
&= \mathbf{c}\big(w, y(w)\big) + \lambda_1 f\big(y(w)\big) + (\lambda_2 - \lambda_1)\big[f\big(y(w)\big) - f(y_1)\big] \\
&\geq \mathbf{c}\big(w, y(w)\big) + \lambda_1 f\big(y(w)\big)
\end{aligned}$$

where the last inequality holds due to (10).

Next, we have to show that $W^{\lambda}_{\text{diff}} \to \emptyset$ as $\lambda \to 0^+$, *i.e.* that for almost every $w \in W$, there is a $\lambda > 0$ such that $w \notin W^{\lambda}_{\text{diff}}$. To this end, let $w \in W$ be given. We can assume that $y(w)$ is a unique solution of solver (1), since two solutions, say y_1 and y_2, coincide only on the hyperplane $F(y_1, y_2)$ in W, which is of measure zero. Thus, since Y is finite, the constant

$$c = \min_{\substack{y \in Y \\ y \neq y(w)}} \big\{\mathbf{c}(w, y) - \mathbf{c}\big(w, y(w)\big)\big\}$$

is positive. Denote

$$d = \max_{y \in Y} \big\{f\big(y(w)\big) - f(y)\big\}. \tag{11}$$

If $d > 0$, set $\lambda < c/d$. Then, for every $y \in Y$ such that $f\big(y(w)\big) > f(y)$, we have

$$\lambda < \frac{\mathbf{c}(w, y) - \mathbf{c}\big(w, y(w)\big)}{f\big(y(w)\big) - f(y)}$$

which rewrites

$$\mathbf{c}\big(w, y(w)\big) + \lambda f\big(y(w)\big) < \mathbf{c}(w, y) + \lambda f(y). \tag{12}$$

For the remaining y's, (12) holds trivially for every $\lambda > 0$. Therefore, $y(w)$ is a solution of the minimization problem (2), whence $y_\lambda(w) = y(w)$. This shows that $w \in W^{\lambda}_{\text{eq}}$ as we wished. If $d = 0$, then $f\big(y(w)\big) \leq f(y)$ for every $y \in Y$ and (12) follows again. □

Proof of Property A3. Let $y_1 \neq y_2 \in Y$ be given. We show that on the component of the set

$$\{w \in W : y(w) = y_1 \text{ and } y_\lambda(w) = y_2\} \tag{13}$$

the function f_λ agrees with a δ-interpolator, where $\delta \leq C\lambda$ and $C > 0$ is an absolute constant. The claim follows as there are only finitely many sets and their components of the form (13) in $W^{\lambda}_{\text{diff}}$.

Let us set

$$h(w) = \mathbf{c}(w, y_1) - \mathbf{c}(w, y_2) \quad \text{for } w \in W$$

and

$$g(w) = f(y_2) - \frac{1}{\lambda} h(w).$$

The condition on $\mathbf{c}$ tells us that h is a non-constant affine function. It follows by the definition of $F(y_1, y_2)$ and $F_\lambda(y_1, y_2)$ that

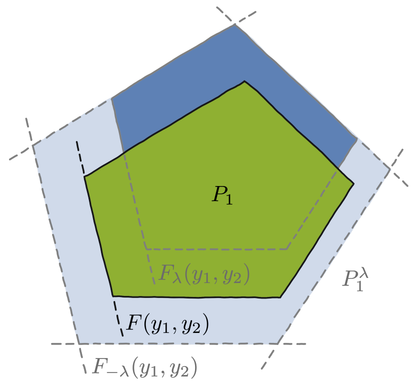
$$h(w) = 0 \quad \text{if and only if} \quad w \in F(y_1, y_2) \tag{14}$$

and

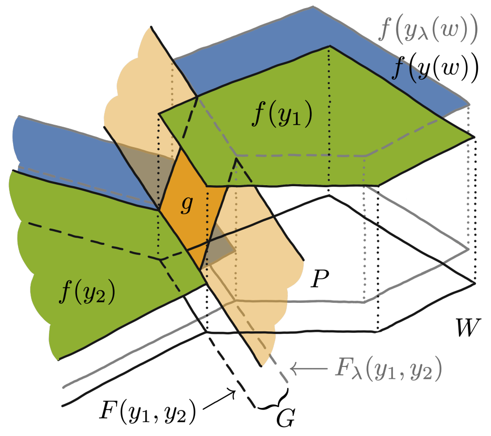
$$h(w) = \lambda\big(f(y_2) - f(y_1)\big) \quad \text{if and only if} \quad w \in F_\lambda(y_1, y_2). \tag{15}$$

By Observation 5, the sets F and F_λ are parallel hyperplanes. Denote by G the nonempty intersection of their corresponding half-spaces in W. We show that g is a δ-interpolator of f on G between y_1 and y_2, with δ being linearly controlled by λ.

We have already observed that g is the affine function ranging from $f(y_1)$ – on the set $F_\lambda(y_1, y_2)$ – to $f(y_2)$ – on the set $F(y_1, y_2)$. It remains to show that g attains both the values $f(y_1)$ and $f(y_2)$ at most δ-far from the sets P_1 and P_2, respectively, where $P_k \in \mathcal{W}$ denotes a component of the set $\{w \in W : y(w) = y_k\}$, $k = 1, 2$.



(a) The facets of P_1 consist of parts of hyperplanes $F(y_1,z_k)$ in W. Each facet $F(y_1,z_k)$ has its corresponding shifts F_λ and $F_{-\lambda}$, from which only one intersects P. The polytope P_1^λ is then bounded by those outer shifts.



(b) The interpolator g attains the value $f(y_1)$ on a part of $F_\lambda(y_1,y_2)$ – a border of the domain G. The value $f(y_2)$ is attained on a part of $F(y_1,y_2)$ – the second border of the strip G.

Figure 6: The polytopes P_1 and P_1^λ and the interpolator g.

Consider y_1 first. By Observation 4, there are $z_1,\dots,z_\ell \in Y$, such that facets of P_1 are parts of hyperplanes $F(y_1,z_1),\dots,F(y_1,z_\ell)$ in W. Each of them separates W into two half-spaces, say W_k^+ and W_k^-, where W_k^- is the half-space which contains P_1 and W_k^+ is the other one. Let us denote

$$c_k(w) = \mathbf{c}(w,y_1) - \mathbf{c}(w,z_k) \quad \text{for } w \in W \text{ and } k = 1,\dots,\ell.$$

Every c_k is a non-zero linear function which is negative on W_k^- and positive on W_k^+. By the definition of y_1, we have

$$\mathbf{c}(w,y_1) + \lambda f(y_1) \le \mathbf{c}(w,z_k) + \lambda f(z_k) \quad \text{for } w \in P_1 \text{ and for } k = 1,\dots,\ell,$$

that is

$$c_k(w) \le \lambda\big(f(z_k) - f(y_1)\big) \quad \text{for } w \in P_1 \text{ and for } k = 1,\dots,\ell.$$

Now, denote

$$W_k^\lambda = \big\{w \in W : c_k(w) \le \lambda\big|f(z_k) - f(y_1)\big|\big\} \quad \text{for } k = 1,\dots,\ell.$$

Each W_k^λ is a half-space in W containing W_k^- and hence P_1. Let us set $P_1^\lambda = \bigcap_{k=1}^{\ell} W_k^\lambda$. Clearly, $P_1 \subseteq P_1^\lambda$ (see Fig. 6). By Observation 5, the distance of the hyperplane $\big\{w \in W : c_k(w) = \lambda\big|f(z_k) - f(y_1)\big|\big\}$ from P_1 is at most λK, where K is given by (7). Therefore, since all the facets of P_1^λ are at most λK far from P_1, there is a constant C such that each point of P_1^λ is at most $C\lambda$ far from P_1.

Finally, choose any $w_1 \in P_1^\lambda \cap F_\lambda(y_1,y_2)$. By (15), we have $g(w_1) = f(y_1)$, and by the definition of P_1^λ, w_1 is no farther than $C\lambda$ away from P_1.

Now, let us treat y_2 and define the set P_2^λ analogous to P_1^λ, where each occurrence of y_1 is replaced by y_2. Any $w_2 \in P_2^\lambda \cap F(y_1,y_2)$ has desired properties. Indeed, (14) ensures that $g(w_2) = f(y_2)$ and w_2 is at most $C\lambda$ far away from P_2. $\square$