L'algorithme Quadripartition appliqué pour la segmentation de l'image

INF TC1

Version Élèves

2015-16

I Modalités

1. Objectifs de ce BE

- Apprentissage des algorithmes et du langage Python
- Acquisition des bases pour la suite des enseignements d'Informatique à l'ECL.

2. Principe de ce BE

Le principe d'un certain algorithme de segmentation d'image vous est proposé. L'objectif est d'analyser le problème posé produire un algorithme; avant de l'implémenter en langage python et d'analyser son comportement. Il vous est demandé aussi de répondre à un certain nombre de question.

3. Travaux à rendre

- Proposez un algorithme. Implémentez celui-ci en langage python et analysez son comportement en l'appliquant notamment sur l'image fournie et en variant le paramètre seuil.
- Répondre aux questions posées.
- Ces travaux doivent être rendus dans **un délai de 2 semaines** ouvrables après la séance.
- Créer une archive (zip, rar, etc.) contenant les codes des exercices ainsi que votre compte rendu traitant les points précédents.

II Segmentation

1. Description du problème

La segmentation d'image est le processus qui consiste à découper une image en régions connexes présentant une homogénéité selon un certain critère, comme par exemple la couleur, la texture, la profondeur, le mouvement, etc. L'image initiale est retrouvée par l'union de ces régions.

La segmentation est une étape importante dans l'extraction des informations qualitatives d'une image. Elle apporte une description de haut niveau sémantique. Les régions voisines sont connectées à travers un graphe ou chaque région porte une étiquette donnant des informations qualitatives discriminantes comme sa taille, sa couleur, sa forme, son mouvement ou encore son orientation. L'image est réduite à un graphe la représentant, où des nœuds étiquetés contiennent presque toutes les informations utile au système. Les arcs de ce graphe sont généralement valués précisant si deux régions connectées sont en simple contact ou si l'une est incluse dans l'autre. D'autres informations topologiques peuvent également être stockées comme par exemple le positionnement relatif : une région est au-dessous d'une autre par exemple. La construction de ce graphe peut être plus ou moins complexe et dépend des techniques de segmentation utilisées.

Les algorithmes de segmentation sont généralement groupés en trois grandes catégories :

1. Segmentation à base de pixels
2. Segmentation à base de régions
3. Segmentation à base de contours

La première catégorie utilise souvent les histogrammes de l'image. Par seuillage, les différentes couleurs représentatives de l'image sont identifiées, l'algorithme construit ainsi des classes de couleurs qui sont ensuite projetées sur l'image.

La deuxième catégorie correspond aux algorithmes d'accroissement de régions ou de partitionnement de régions. Le premier est une approche bottom-up : on part d'un ensemble de petites régions uniformes dans l'image, d'un, voire de quelques pixels, et on regroupe les régions adjacentes d'une même couleur. Ce regroupement s'arrête quand aucun regroupement n'est plus possible. Le deuxième, est une méthode top-down: on part de l'image entière que l'on va subdiviser récursivement en plus petites régions tant que ces régions ne sont pas suffisamment homogènes. Un mélange de ces deux méthodes est l'algorithme de split and merge que nous allons détailler et étudier par la suite.

Finalement, la troisième catégorie utilise l'information de contours des objets. Comme la plupart de ces algorithmes fonctionnent au niveau du pixel, ils sont considérés comme des algorithmes locaux. Ce type d'algorithme est formellement proche des techniques d'accroissement de régions fonctionnant au niveau du pixel et sont purement locales et en général limitées pour traiter des images complexes et bruitées.

2. Approche du type "split and merge"

L'algorithme split and merge a été proposé par Pavlidis et Horowitz en 1974. Selon cet algorithme, deux étapes de split, partionnement, et de merge, fusion, sont opérées.

2.1 Split

La méthode de découpage de l'image utilisée dans cet algorithme est basée sur la notion de *quadripartition*, *quadtree* en anglais. Une structure de données du type *arbre quaternaire* peut permettre de stocker l'image à plusieurs niveaux de résolution. On part souvent de la totalité de l'image. Si cette image vérifie un certain critère d'homogénéité de couleur, l'algorithme s'arrête. Sinon, on découpe cette région en quatre parties de la même taille et on relance, récursivement, cette procédure dans chacune des quatre parties. La région initiale est considérée comme un nœud dans un graphe et les sous parties comme les quatre fils de ce nœud.

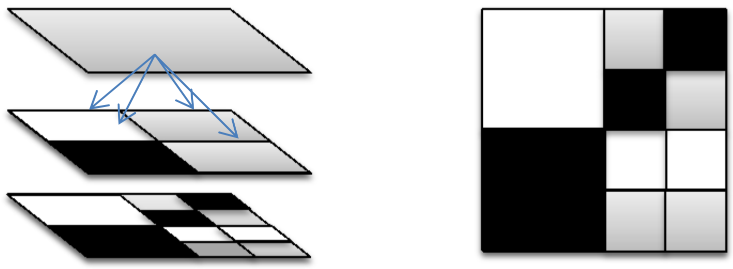


Figure 1. Découpage par *quadripartition* d'une image 4x4.

La Figure 1 montre une image en noir et blanc 4x4 et le découpage correspondant en trois niveaux. La structure d'arbre associée à ce découpage est illustrée Figure 2.

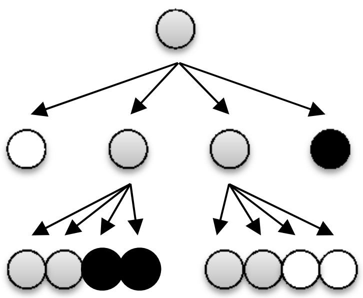


Figure 2. Arbre quaternaire à partir de l'image de la Figure 1.

Dans cet exemple, le critère d'homogénéité est absolu. Une zone est dite homogène si elle ne contient que des pixels de la même couleur (seuil d'homogénéité = 100%). En pratique on est plus tolérant et considère qu'une zone est homogène dès que plus de 75% d'une couleur domine.

De manière plus générale, on va appliquer ce principe de réduction à des images colorées. Chaque pixel d'une image en couleur est représenté par trois intensités en rouge, vert et bleu. Chaque intensité est codée sur un octet, sa valeur varie de 0 à

255. Le critère d'homogénéité est fixé par un seuil. En dessous de ce seuil, la région est conservée et constitue une feuille, nœud terminal, de l'arbre. On lui attribue alors la couleur de la moyenne des pixels la constituant. Au-dessus de ce seuil, la région est découpée en quatre.

2.2 Merge

La procédure de découpage décrite précédemment aboutit à un nombre de régions parfois trop élevé. Il se peut qu'elle coupe de cette façon une zone homogène en deux ou quatre parties.

La solution, qui correspond à la phase fusion, merge, de l'algorithme, est de procéder à une fusion de régions après le découpage. L'implémentation la plus simple de cette fusion consiste à rassembler, fusionner, les couples de régions adjacentes repérées, de couleur proche, dans l'arbre issu de la phase split.

Pour réaliser cette fusion, il faut d'abord tenir à jour une liste des contacts entre régions (chaque région dispose d'une liste de régions avec lesquelles elle est en contact). On obtient ainsi un graphe d'adjacence de régions ou *Region Adjacency Graph – RAG* (Ce graphe de contact doit se construire en même temps que l'arbre de découpage). Ensuite, l'algorithme va marquer toutes les régions comme non-traitées, et choisir la première région R non traitée disponible.

Les régions en contact avec R sont empilées et examinées les unes après les autres pour savoir si elles doivent fusionner avec R. Si c'est le cas, la couleur moyenne de R est mise à jour et les régions en contact avec la région fusionnée sont ajoutées à la pile des régions à comparer avec R. La région fusionnée est marquée comme traitée. Une fois la pile vide, l'algorithme choisi la prochaine région marquée comme non traitée et recommence, jusqu'à ce que toutes les régions soient traitées.

2.3 Exemple

Un exemple d'image traitée par l'algorithme *quadripartition* est illustré par la Figure 3. L'image originale est traitée avec des valeurs de seuil de plus en plus petites augmentant le nombre de régions détectées.

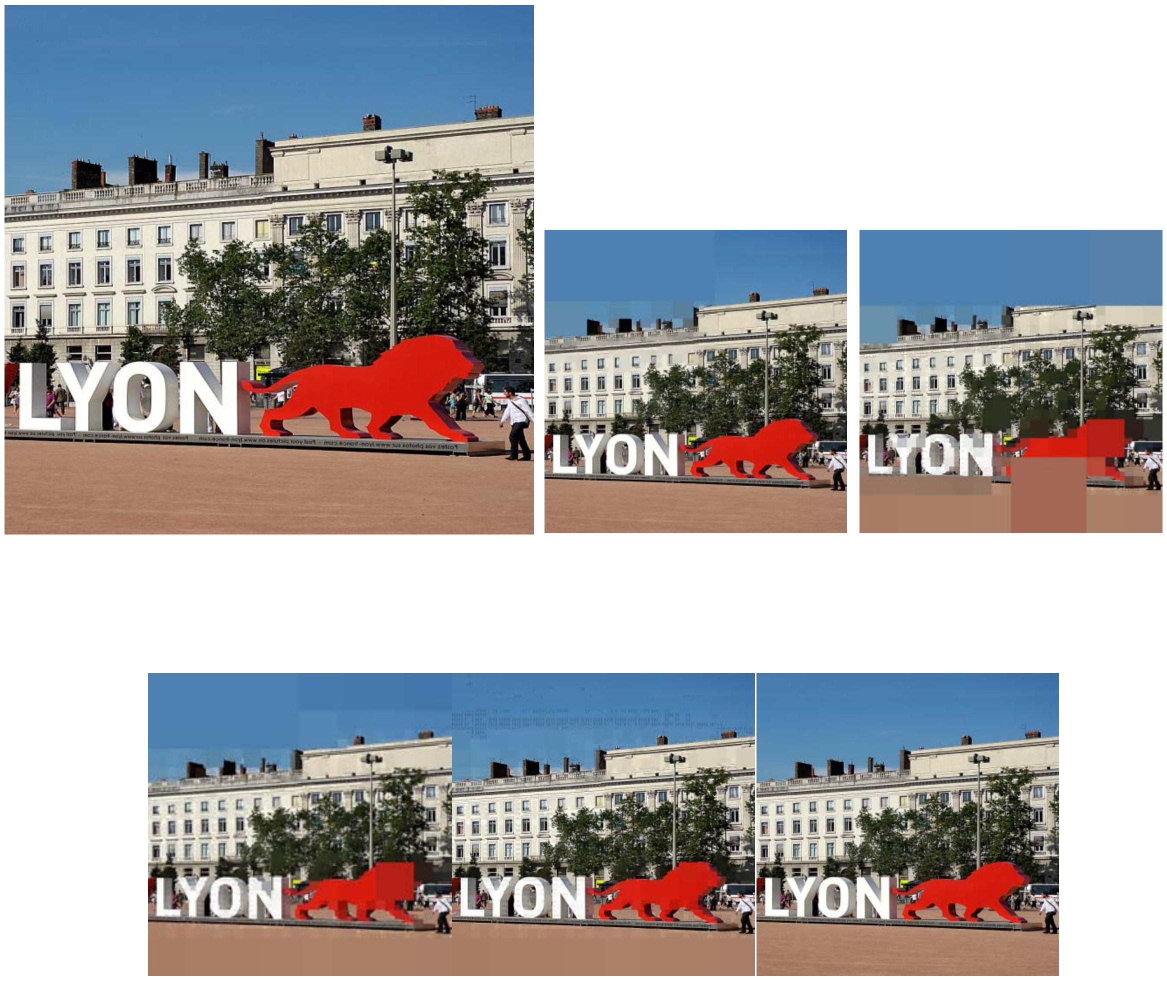


Figure 3. Image de Lyon et les résultats obtenus avec différentes valeurs de seuil. En haut à gauche, l'image originale non segmentée. Les suivantes avec des valeurs de seuil de plus en plus basses augmentant le nombre de régions.

3. Travail à réaliser

- On souhaite réaliser l'algorithme de split qui utilise la stratégie *quadripartition.*
- Proposez une fonction qui estime l'homogénéité des pixels d'une région rectangulaire de l'image.
- Proposez un algorithme récursif permettant de reproduire la stratégie *quadripartition.*
- Selon vous, la réalisation récursive d'un *arbre quaternaire*, est-elle indispensable pour l'étape de split ? Qu'en est-il pour les deux étapes split et merge ?
- Implanter une fonction permettant de lire la valeur d'un pixel.
- Implanter une fonction permettant d'affecter une couleur à un pixel.
- Implanter une fonction permettant d'affecter une couleur à une région rectangulaire de l'image.
- Implanter les algorithmes de split en langage python.

La réalisation du travail demandé nécessite l'utilisation de la librairie PILLOW fourni avec Anaconda. Nous avons notamment besoin de lire une image à partir de son nom dans le répertoire courant :

```
im = Image.open("Image8.bmp").
```

Importer les données des pixels sous forme d'une matrice px :

```
px = im.load() #Importation des pixels de l'image
```

Obtenir la taille de cette image :

```
w,h=im.size
```

Utiliser, et donc importer, la fonction mathématique racine carrée, sqrt, qui se trouve dans la librairie math :

```
from math import sqrt
```

Vous devez donc exécuter la séquence suivante au début de votre programme.

```
from PIL import Image #Importation de la librairie d'image PIL
im = Image.open("Image8.bmp") #Ouverture du fichier d'image
px = im.load() #Importation des pixels de l'image

from math import sqrt #Importation de la fonction sqrt de la librairie math

w,h=im.size
```

Par la suite on peut accéder au pixel px[x,y] de coordonnées x, y par la commande python suivante :

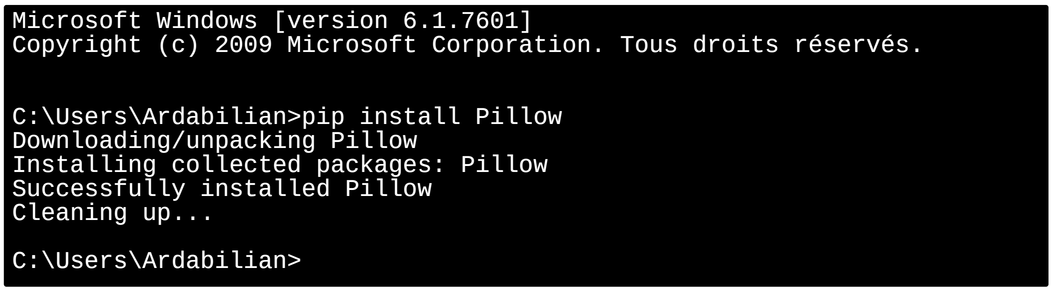
```
p=px[x,y]
```

Pour affecter une couleur r,g,b au pixel p[x,y], on utilisera la commande :

```
px[x,y] = r,g,b
```

où r,g et b sont des variables.

Si les commandes de lecture d'image et d'accès aux pixels ne sont pas disponibles, c'est que la librairie PILLOW n'est pas installée. Pour l'installer vous devez exécuter la commande suivante dans une fenêtre de commande Windows.



```
Microsoft Windows [version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation. Tous droits réservés.


C:\Users\Ardabilian>pip install Pillow
Downloading/unpacking Pillow
Installing collected packages: Pillow
Successfully installed Pillow
Cleaning up...

C:\Users\Ardabilian>
```

4. Split & Merge suite

Une fois l'algorithme de split finalisé et implanté, il reste à traiter la partie fusion, merge, décrite plus haut dans la section 2.2.

La partie fusion demande d'une part la connaissance des régions de l'image (obtenue par l'algorithme de split), la liste de contact entre ces régions, RAG, et d'autre part l'algorithme de fusion qui rassemble les régions de couleurs proches.

Traitons un exemple concret, l'image ''test2.bmp'' de 4x4 pixels et constituée de 2 régions. Procédons alors par étape.

– Commençons par proposer un algorithme qui permet de rassembler dans une structure de données, les données des régions monoblocs de couleur homogène de l'image, obtenues à l'étape de split.

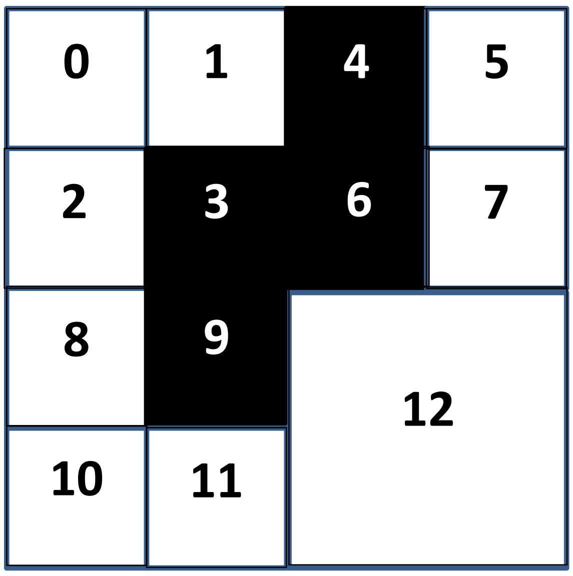
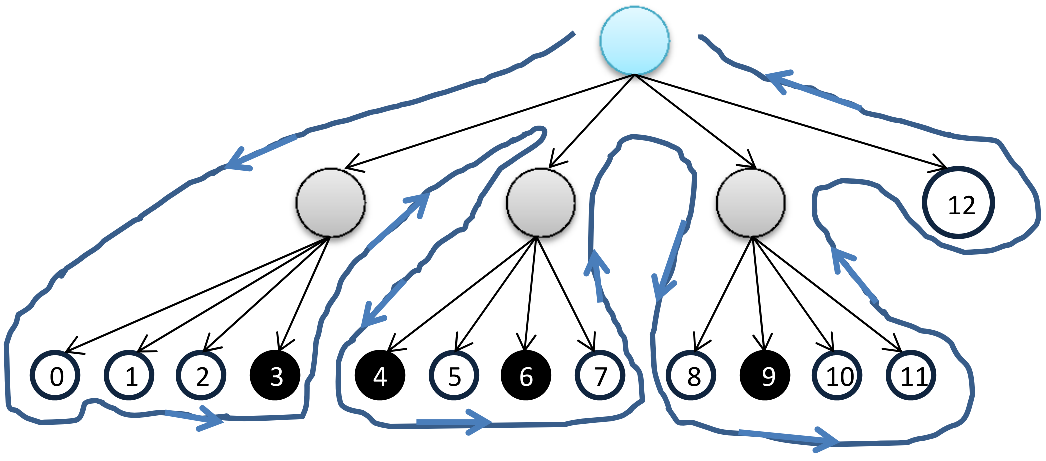


Figure 4. En haut – l'image test2.bmp constituée de 16 pixels. En bas – L'arbre parcouru en appliquant la stratégie *quadripartition*.



Soit l'image de la Figure 4, en suivant la stratégie de *quadripartition*, nous parcourons en profondeur un arbre quaternaire dont les feuilles correspondent aux régions homogènes. Ces feuilles sont numérotées dans l'ordre de leur visite permettant ainsi de numéroter les régions correspondantes de l'image dont on retiendra pour chacune d'elle, par exemple, les coordonnées du coin gauche supérieur, sa largeur qui égale sa hauteur dans notre cas, ainsi que sa couleur. D'autres possibilités existent pour caractériser une région. Nous vous laissons d'opter dans la suite pour le choix le plus judicieux. A présent, il ne reste plus qu'à placer ces informations dans l'ordre croissant des feuilles, dans une structure de donnée.

– Modifiez votre précédent programme pour apporter cette fonctionnalité. A l'issu de cette étape de réalisation, votre programme doit pouvoir retourner une liste, L__regions, de ces régions homogènes.

Une fois les régions homogènes listées, nous devons retrouver pour chacune d'elle ses voisines, les régions adjacentes, et les placer dans une structure adéquate. On pourra s'inspirer de la Figure 5.

– Proposez alors un algorithme qui, à partir des données de deux régions et une valeur de seuil raisonnable, fournie par l'utilisateur, teste si ces régions sont adjacentes. Souvenez-vous du choix judicieux pour représenter une région !

– Proposer un algorithme qui prend en entrée la liste L__regions et l'indice, i, d'une région dans cette liste et teste l'adjacence de toutes les autres régions de la liste avec la région i. En sortie, votre algorithme fournit sous forme de liste, pour chaque région, les régions adjacentes.

L'étape suivante consiste à étendre le travail précédent pour réaliser une liste de régions adjacente. Celle-ci est illustrée par la Figure 4. Il s'agit de RAL qui n'est autre que la Liste des Régions Adjacentes. Une solution simple et pratique pour représenter les régions adjacentes de l'image est d'utiliser un graphe. Chaque région apparaît comme nœud de graphe. Les liens entre les nœuds expriment la relation adjacence. Ce graphe peut être implémenté par une Liste de Régions Adjacentes, Figure 4. Dans cette liste, [0 1 3] signifie que les régions (nœuds) 1 et 3 sont adjacentes à la région 0.

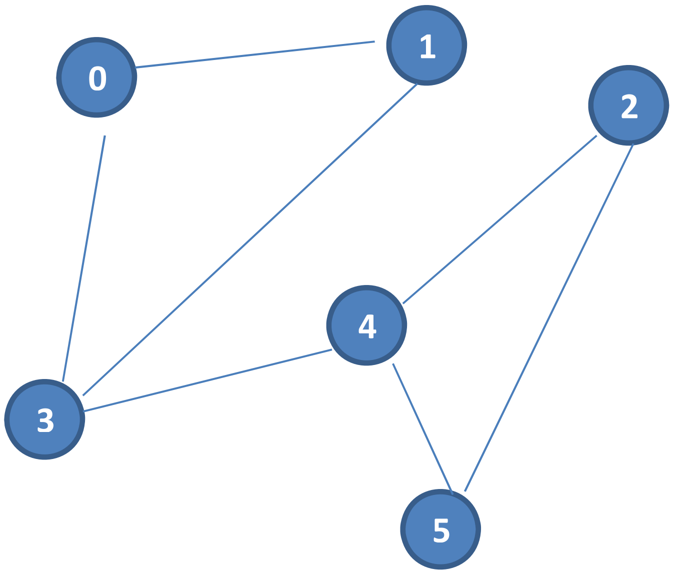


Figure 5. Ici, une solution simple et pratique pour représenter les régions adjacentes de l’image est de s’inspirer d’un graphe où chaque nœud est une région de l’image. Dans ce cas un lien entre deux nœuds indique un lien d’adjacence. On peut alors représenter le lien d’adjacence entre le nœud 0 et les nœuds 1 et 3 par [0 1 3]. En allant jusqu’au bout de ce raisonnement, nous obtenons une liste d’adjacence des régions (en anglais *Region Ajacency List* - RAL) semblable au RAG (*Region Ajacency Graph*), mais plus simple et pratique.

RAL = [[0 1 3][1 0 3][2 4 5][3 0 1][4 2 3 5] [5 2 4]]

RAL: *Region Adjacency List*

L’ultime étape est la réalisation de la fusion et son algorithme. Ici, la condition pour fusionner deux régions est qu’ils soient adjacentes et de couleurs proches. Une ébauche de cet algorithme est introduite dans la section 2.2.

Nous partons d’une RAL et l'algorithme marque toutes les régions comme non-traitées, et choisir la première région R non traitée disponible. Les régions en contact avec R sont examinées les unes après les autres pour savoir si elles doivent fusionner avec R. En comparant leurs couleurs par exemple. Si c'est le cas, la couleur moyenne de R est mise à jour et les régions en contact avec la région fusionnée sont à leur tour comparées avec R. La région fusionnée est marquée comme traitée. Une fois qu’il n’y a plus de région non traitée en contact avec R et de couleur proche, l'algorithme choisi la prochaine région marquée comme non traitée et recommence, jusqu'à ce que toutes les régions soient traitées.

Prenons le cas de la Figure 4. Le RAL de cette image est la suivante.

[[0, 1, 2, 3],
[1, 0, 2, 3, 4, 6],
[2, 0, 1, 3, 8, 9],
[3, 0, 1, 2, 4, 6, 8, 9, 12],
[4, 1, 3, 5, 6, 7],
[5, 4, 6, 7],
[6, 1, 3, 4, 5, 7, 9, 12],
[7, 4, 5, 6, 12],
[8, 2, 3, 9, 10, 11],
[9, 2, 3, 6, 8, 10, 11, 12],
[10, 8, 9, 11],
[11, 8, 9, 10, 12],
[12, 3, 6, 7, 9, 11]]

L'algorithme commence avec la première ligne qui indique que la région zéro est en contact avec les régions 1, 2 et 3. L'algorithme trouve que la région 1 et en suite 2 ont des couleurs similaires à la région 0. Mais pas la région 3. (Voir la figure 4). A chaque fois qu'une région adjacente de même couleur que la région de départ est traitée, elle est marquée dans la liste des régions adjacentes (dans l'exemple ci-dessous par un D).

Les régions adjacentes aux régions 1 et 2, 1^{e} et 2^{e} lignes sont successivement ajoutées à la première ligne.

[[0, 1, 2, 3, 1, 0, 2, 3, 4, 6, 2, 0, 1, 3, 8, 9],
[1, 0, 2, 3, 4, 6],
[2, 0, 1, 3, 8, 9],
[3, 0, 1, 2, 4, 6, 8, 9, 12],
[4, 1, 3, 5, 6, 7],
[5, 4, 6, 7],
[6, 1, 3, 4, 5, 7, 9, 12],
[7, 4, 5, 6, 12],
[8, 2, 3, 9, 10, 11],
[9, 2, 3, 6, 8, 10, 11, 12],
[10, 8, 9, 11],
[11, 8, 9, 10, 12],

[12, 3, 6, 7, 9, 11]]

En supprimant les redondances et les cycles, 0 est adjacente à 0, ainsi que la région 3 différente de couleur, nous obtenons.

```
[[0, 1, 2, 4, 6, 8, 9],
 [D, 0, 2, 3, 4, 6],
 [D, 0, 1, 3, 8, 9],
 [3, 0, 1, 2, 4, 6, 8, 9, 12],
 [4, 1, 3, 5, 6, 7],
 [5, 4, 6, 7],
 [6, 1, 3, 4, 5, 7, 9, 12],
 [7, 4, 5, 6, 12],
 [8, 2, 3, 9, 10, 11],
 [9, 2, 3, 6, 8, 10, 11, 12],
 [10, 8, 9, 11],
 [11, 8, 9, 10, 12],
 [12, 3, 6, 7, 9, 11]]
```

Les régions 4, 6 et 9 ont des couleurs différentes de celle de la région 0. Elles sont supprimées de la première ligne. Mais pas la région 8, de couleur blanche. Les régions en contact avec 8 sont ajoutées. Ce sont les régions 10 et 11. Elles sont de même couleur que la région 0.

```
[[0, 1, 2, 8, 10, 11],
 [D, 1, 0, 2, 3, 4, 6],
 [D, 2, 0, 1, 3, 8, 9],
 [3, 0, 1, 2, 4, 6, 8, 9, 12],
 [4, 1, 3, 5, 6, 7],
 [5, 4, 6, 7],
 [6, 1, 3, 4, 5, 7, 9, 12],
 [7, 4, 5, 6, 12],
 [D, 8, 2, 3, 9, 10, 11],
 [9, 2, 3, 6, 8, 10, 11, 12],
 [D, 10, 8, 9, 11],
 [D, 11, 8, 9, 10, 12],
 [12, 3, 6, 7, 9, 11]]
```

En fin des itérations pour la première couleur, nous obtenons.

```
[[0, 1, 2, 8, 10, 11, 12, 5, 7],
 [D, 1, 0, 2, 3, 4, 6],
 [D, 2, 0, 1, 3, 8, 9],
 [3, 0, 1, 2, 4, 6, 8, 9, 12],
 [4, 1, 3, 5, 6, 7],
 [D, 5, 4, 6, 7],
 [6, 1, 3, 4, 5, 7, 9, 12],
 [D, 7, 4, 5, 6, 12],
```

[D, 8, 2, 3, 9, 10, 11],
[9, 2, 3, 6, 8, 10, 11, 12],
[D, 10, 8, 9, 11],
[D, 11, 8, 9, 10, 12],
[D, 12, 3, 6, 7, 9, 11]]

Le travail continue avec la prochaine région non traitée, la région 3.

[[0, 1, 2, 4, 6, 8, 9, 10, 11, 12, 5, 7],
[D, 1, 0, 2, 3, 4, 6],
[D, 2, 0, 1, 3, 8, 9],
[3, 4, 6, 9],
[D, 4, 1, 3, 5, 6, 7],
[D, 5, 4, 6, 7],
[D, 6, 1, 3, 4, 5, 7, 9, 12],
[D, 7, 4, 5, 6, 12],
[D, 8, 2, 3, 9, 10, 11],
[D, 9, 2, 3, 6, 8, 10, 11, 12],
[D, 10, 8, 9, 11],
[D, 11, 8, 9, 10, 12],
[D, 12, 3, 6, 7, 9, 11]]

Une autre variante de cet algorithme place les régions adjacentes à une région R, dans une pile. Une fois que la pile est vide. C'est qu'il ne reste plus de région adjacente à R. On passe alors à la région non traitée suivante.

- Proposez un algorithme réalisant la fusion des régions en vous appuyant sur les structures et les données mises en place plus haut, ainsi que l'exemple simple traité.
- Implémenter chaque algorithme en python.

5. Travail à réaliser

Résumons le travail à faire.

- Modifiez votre précédent programme pour produire une structure qui contient les régions homogènes. A l'issu de cette étape de la réalisation, votre programme doit pouvoir retourner une liste, L_regions, de ces régions homogènes.

- Proposez un algorithme qui, à partir des données de deux régions et une valeur de seuil raisonnable, fournie par l'utilisateur, teste si ces régions sont adjacentes. Souvenez-vous du choix judicieux pour représenter une région !

– Proposez un algorithme qui prend en entrée la liste L_regions et l'indice, i, d'une région dans cette liste et teste l'adjacence de toutes les autres régions de la liste avec la région i. En sortie, votre algorithme fournit sous forme de liste, pour chaque région les régions adjacentes.

– Proposez un algorithme réalisant la fusion des régions en vous appuyant sur les structures et les données mises en place plus haut, ainsi que l'exemple simple traité.

– Implantez chaque algorithme en python.

– Testez votre programme sur l'image test2.bmp et puis sur une image plus complexe, image10.bmp.

A vos plumes d'abords, à vos claviers ensuite !

6. Pour aller plus loin

La compression d'image et de vidéo fait appel à de nombreux algorithmes permettant de compresser les données des pixels dans une image, on parle alors de la compression spatiale. En étendant cette compression aux images successives d'une séquence d'image, on parle de la compression spatio-temporelle. Ces compressions se font en supprimant les données redondantes dans une image ou des images successives dans une séquence.

Pour cela, la première norme de compression, MPEG1, considère une image divisée en bloc de 8x8 pixels. Dans un tel bloc, les pixels sont dans la plupart des cas similaires. Si la norme MPEG1 visait les ordinateurs multimédia, MPEG2 vise un contexte plus large. Notamment, la télévision numérique, DVD, mais aussi les formats hautes résolutions. MPEG4 chamboule l'ordre en proposant une compression par objet. Chaque objet dans une image est segmenté et subit une compression. Il est par ailleurs possible de composer les objets au moment de diffusion. Dernièrement, le standard HEVC (High Efficiency Video Coding) propose par ISO/IEC Moving Picture Experts Group (MPEG) marque une nouvelle étape dans la capacité de compression des images animées. Ci-après, vous trouverez un article traitant ce sujet. C'est la stratégie quadripartition qui est utilisée. Fini les blocs de tailles figés.

Block Merging for Quadtree-Based Partitioning in HEVC

Philipp Helle, Simon Oudin, Benjamin Bross, *Student Member, IEEE,* Detlev Marpe, *Senior Member, IEEE,* M. Oguz Bici, Kemal Ugur, Joel Jung, Gordon Clare, and Thomas Wiegand, *Fellow, IEEE*

***Abstract*—The joint development of the upcoming High Efficiency Video Coding (HEVC) standard by ITU-T Video Coding Experts Group and ISO/IEC Moving Picture Experts Group marks a new step in video compression capability. In technical terms, HEVC is a hybrid video-coding approach using quadtree-based block partitioning together with motion-compensated prediction. Even though a high degree of adaptability is achieved by quadtree-based block partitioning, this approach has certain intrinsic drawbacks, which may result in redundant sets of motion parameters being transmitted. Previous work has shown that those redundancies can effectively be removed by merging the leafs of a particular quadtree structure. Following this concept, a block merging algorithm for HEVC is now proposed. This algorithm generates a single motion parameter set for a whole region of contiguous motion-compensated blocks. In this paper, we describe the various components of the proposed block merging algorithm and, using experimental evidence, demonstrate their benefits in terms of coding efficiency.**

***Index Terms*—Block merging, direct mode, High Efficiency Video Coding (HEVC), motion compensation, quadtree partition.**

I. Introduction

In IMAGE AND VIDEO CODING, pictures and their corresponding sets of sample arrays are usually decomposed into blocks such that each block is associated with a particular set of model or coding parameters. In hybrid video coding, e.g., each block is either spatially or temporally predicted and the resulting prediction residual is represented by using transform coding. For the purpose of partitioning, tree-structured schemes are very well suited for image coding, as they can be optimized in the rate-distortion (RD) sense by simple algorithms [1], [2]. However, the authors of [3] have shown that tree-structured partitioning may result in suboptimal RD performance when dependencies between leaf nodes of different parents are not exploited. Their proposed solution to this shortcoming is a so-called prune-join scheme that allows for joint coding of leaf nodes belonging to different parents [3]. Inspired by this study, the authors of [4] have shown that this scheme can improve quadtree-based motion models in the same manner. They have demonstrated significant improvements in RD performance by incorporating a leaf merging extension into the motion model of H.264/Advanced Video Coding (AVC) [5]. Also, in [6], a motion model similar to that of H.264/AVC was used, and it has been shown that its performance can be greatly improved by incorporating a leaf merging step subsequent to an RD-optimized tree pruning.

In our study, we present a coding tool called *block merging*, which was directly inspired by the scheme proposed in [4]. It has been designed as a part of a novel quadtree-based video coding approach [7], [8] and was maintained as a feature of the Fraunhofer HHI response [9] to the joint call for proposals (CfP) on High Efficiency Video Coding (HEVC) technology that was issued by ITU-T Video Coding Experts Group (VCEG) and ISO/IEC Moving Picture Experts Group [10]. The intention of the CfP was to identify video coding technology with substantially higher compression capabilities than the existing H.264/AVC standard. Block merging was adopted as a part of the initial HEVC test model under consideration (TMuC) in April 2010. Since then, many contributions to the HEVC standardization process led to improvement of the block merging algorithm in various aspects.

In this paper, we present this extended block merging algorithm and show how it fits into the coding architecture of HEVC, which offers much more versatile block partitioning capabilities than prior designs as, e.g., H.264/AVC. Compared to our previous work, the algorithm is much better adapted to this partitioning, e.g., by availability to all sizes and shapes of inter-picture predicted blocks and by an increased flexibility in the choice of neighboring blocks to be merged. Further RD-performance gains are achieved by including temporal prediction into the merging scheme. Also, a very efficient *skip* mode has been incorporated, which signals a merged block without transmitting a corresponding residual signal. In addition to the RD-performance improvements, we show how aspects of complexity, parsing robustness, error resilience, and friendliness to parallel implementations were considered

Manuscript received April 15, 2012; revised July 25, 2012; accepted August 21, 2012. Date of publication October 5, 2012; date of current version January 8, 2013. This paper was recommended by Associate Editor O. C. Au.

P. Helle, S. Oudin, B. Bross, and D. Marpe are with the Image and Video Coding Group, Fraunhofer Institute for Telecommunications-Heinrich Hertz Institute, Berlin 10587, Germany (e-mail: philipp.helle@hhi.fraunhofer.de; simon.oudin@hhi.fraunhofer.de; benjamin.bross@hhi.fraunhofer.de; detlev.marpe@hhi.fraunhofer.de).

K. Ugur is with Nokia Corporation, Tampere 33720, Finland (e-mail: kemal.ugur@nokia.com).

M. O. Bici is with Nokia Corporation, Tampere 33720, Finland, and also with the Tampere University of Technology, Tampere 33720, Finland (e-mail: ext-oguz.bici@nokia.com).

J. Jung and G. Clare are with Orange Labs Research and Development, Issy-les-Moulineaux 92794, France (e-mail: joelb.jung@orange.com; gordon.clare@orange.com).

T. Wiegand is with the Department of Image Processing, Fraunhofer Institute for Telecommunications-Heinrich Hertz Institute, Berlin 10587, Germany, and also with the Technical University of Berlin, Berlin 10587, Germany (e-mail: thomas.wiegand@hhi.fraunhofer.de).

Digital Object Identifier 10.1109/TCSVT.2012.2223051

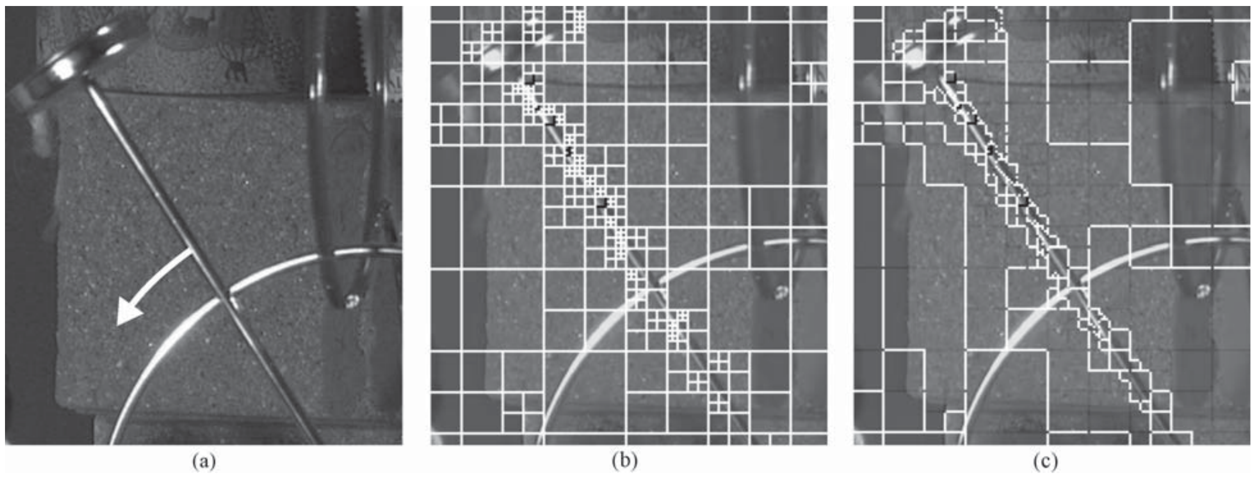


Fig. 1. (a) Details of a picture in the *Cactus* sequence containing a moving object in the foreground with motion indicated by the arrow. (b) Quadtree partitioning for motion parameters is indicated by the white squares. (c) Only those block borders are shown that separate blocks with different motion parameters.

for integrating the leaf merging concept into HEVC. We finally validate the specific design of block merging by providing analysis and experimental results beyond that found in associated JCT-VC standardization documents. Presentation of technical details as well as our experimental results are thereby based on the HEVC specification draft 8 (draft international standard, DIS) and the HEVC software test model (HM) version 8.0 [11], [12]. A general overview of the HEVC standard is given in [13] and an analysis of its coding efficiency is provided in [14].

The rest of this paper is organized as follows. The next section revisits quadtree-based partitioning and its related drawbacks, thereby motivating the establishment of a block merging scheme. Section III briefly introduces quadtree-based picture partitioning and coding of motion parameters as employed in HEVC. The presented block merging algorithm is described in detail in Section IV. Section V addresses similarities between block merging and the direct mode. Conducted experiments and their results are presented in Section VI. Section VII concludes this paper.

II. Background and Problem Formulation

The main motivation for the use of block-based partitioning in image or video compression is to code each block with one specific choice out of a set of simple models since, in general, a single model cannot be expected to capture the properties of a whole image efficiently. The quadtree data structure allows partitioning of an image into blocks of variable size, and is thus a suitable framework for optimizing the tradeoff between model accuracy (e.g., given by distortion D) and model coding cost, typically measured in bit rate R. Simple tree pruning algorithms optimizing a Lagrangian functional

$$D + \lambda * R$$

were developed in [1] and [2], motivating the design of compression algorithms, which are based on quadtrees and related representations. The HEVC video codec examined in this paper follows a quadtree-structured video coding approach with syntax elements carrying the subdivision information for blocks of different types. Such syntax elements may be related to subdivision of blocks for the purpose of prediction or transform coding.

However, the concept of quadtree-based image and video coding is intrinsically tied to certain drawbacks as analyzed in [3]. For instance, the systematic decomposition of every block into four child blocks does not allow to jointly represent child blocks that belong to two different parent blocks. Also, if a given block is divided into four child blocks, all child blocks are typically coded separately, even if two or three of them share the same coding parameters. Thus, the suboptimal RD properties of an initial quadtree-based subdivision can be substantially improved by joining (or merging) nodes belonging to potentially different parents [3].

The authors of [4] have noted that quadtree-based motion models are likewise affected by the aforementioned drawbacks. They introduced a leaf merging step to rectify the associated performance loss. In [15], the authors have given a comprehensive theoretical study on the advantage of leaf merging in quadtree-based motion models.

One example illustrating the potential of block merging is shown in Fig. 1. This particular scene exemplifies the previously discussed drawbacks of quadtree-based partitioning. Fig. 1(b) reveals that an RD-optimized tree pruning forces the quadtree segmentation to subdivide regions unnecessarily. We observed over a range of sequences and coding conditions for HEVC that those blocks, which are neighbored by a previously coded block with exactly the same motion parameters covered approximately 40% of all samples on average. For this experiment, we used an HM version without the encoder estimation for the block merging algorithm, thus removing any associated bias toward merging blocks. This result is already an indication of the oversegmentation due to quadtree-based block partitioning, and it also indicates the potential of jointly coding those neighboring blocks with identical motion parameters.

Fig. 1(c) shows the same partitioning as in (b), but with boundaries between blocks with identical motion parameters removed. This reveals regions of identical motion parameters,

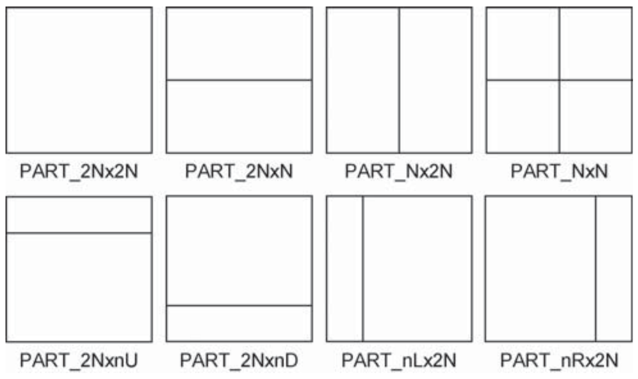


Fig. 2. Different partitions allowed for inter-picture prediction in a CB used by HEVC.

better capturing the distinct types of motion in the scene. Ideally, the boundary of each region should coincide with the motion discontinuities in the given video signal. Using quadtree-based block partitioning combined with merging, the picture can be subdivided into smaller and smaller blocks, thereby approximating the motion boundary and minimizing the size of partitions containing the boundary. Subsequently, each created quadtree leaf block can be merged into a region on either side of the boundary.

It should be noted that a scheme merging spatially neighboring blocks is conceptually similar to spatial prediction modes as, e.g., the *spatial direct mode* in H.264/AVC [16]. This mode also tries to reduce coding cost by using redundancies of motion parameters in neighboring blocks. However, the improvements over H.264/AVC shown in previous work [4], [6] suggest that the merging concept is superior in exploiting these redundancies. This is also confirmed by the experiments presented in this paper, where the proposed block merging algorithm is compared to a direct mode similar to that of H.264/AVC, which we have reintegrated into HEVC just for the purpose of analysis.

III. Quadtree-Based Partitioning in HEVC

The first part of this section gives an overview of the quadtree-based partitioning in HEVC and introduces terminology used throughout the rest of this paper. As we are particularly concerned about the motion model and its parameters, we also introduce the prediction employed for differential coding of motion vectors (MVs). The following description of HEVC is based on the *Main profile*, which is the only profile defined in the DIS.

A. Quadtree Structure

For HEVC, a quadtree-based coding approach was introduced such that each picture is divided into square coding tree blocks (CTBs). Each CTB is the root of a coding tree, which is used to further divide the CTB into coding blocks (CBs). Their size can be adaptively chosen by using a quadtree-based partitioning with the leaves of the quadtree representing the CBs [7]. Each CB is a root for a prediction and a transformation tree. The prediction tree has only one level and describes how a CB can be further split into so-called prediction blocks (PBs), for each of which prediction parameters are specified. Fig. 2 depicts all different ways allowed by the current *Main profile* to split a CB into inter-PBs. For transform coding of the prediction residual signal, each CB can also be split into smaller transform blocks (TBs) using another quadtree called the residual quadtree (RQT) [7], [17]. Fig. 3 illustrates this nested quadtree structure, i.e., the coding quadtree with the CTB as root (solid bold line) and the CBs as leaves (solid lines), each of which is the root of the nested RQT with the TBs as leaves (dashed lines).

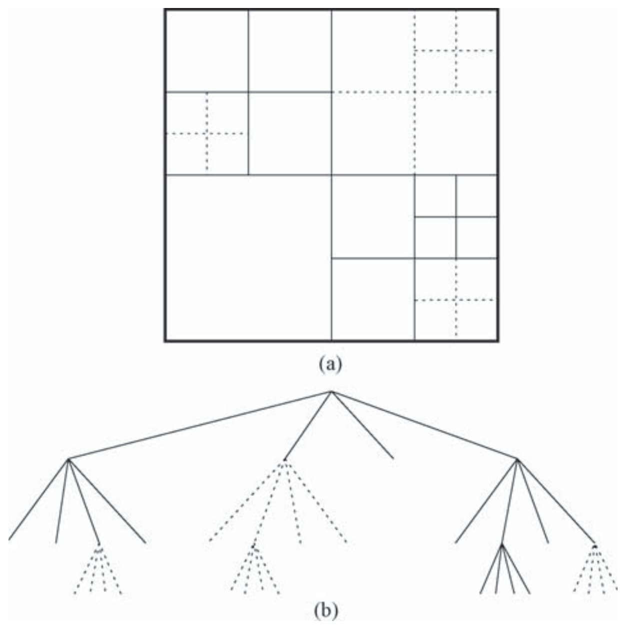


Fig. 3. HEVC quadtree structures. (a) CTB (solid block) partitioned into CB (solid) and transform blocks (dashed) of variable size. (b) Corresponding nested quadtree structure.

All the blocks in different trees (coding, prediction, or transform tree) correspond to specific sample arrays with different sizes. Depending on which tree they are related to, these blocks are associated with a specific syntax structure and form together the so-called units. The TB luma and chroma sample arrays and associated syntax elements, e.g., coded block flags or transform coefficient levels, are grouped together in a transform unit (TU). A prediction unit (PU) encapsulates everything that is related to prediction, i.e., the PB sample arrays and associated syntax elements, e.g., MVs or intra-picture prediction modes. The CB sample arrays, the associated syntax elements like the mode information whether intra- or inter-picture prediction are used and the associated PUs and TUs are grouped together in a coding unit (CU). Consequently, the CTB sample arrays, associated coding tree syntax and associated CUs are considered as coding tree unit (CTU). Thus, it can be said that the CTU generalizes the concept of a macroblock as the basic processing unit in standardized video coding.

B. Prediction of MVs

The H.264/AVC standard has only one single MV predictor to differentially code the MVs, computed as the median of

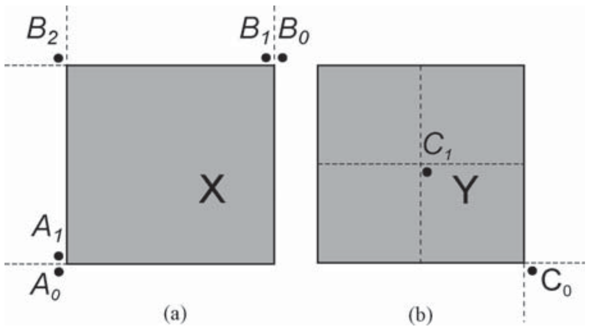


Fig. 4. Black dots indicate sample positions directly adjacent to block *X*, defining positions of possible MVPs. (a) Spatial MVPs positions. (b) Temporal MVPs positions, where *Y* is the collocated block of *X* in a reference picture (does not necessarily match with a PB in this reference picture). Positions C_0 and C_1 are candidates for the TMVP.

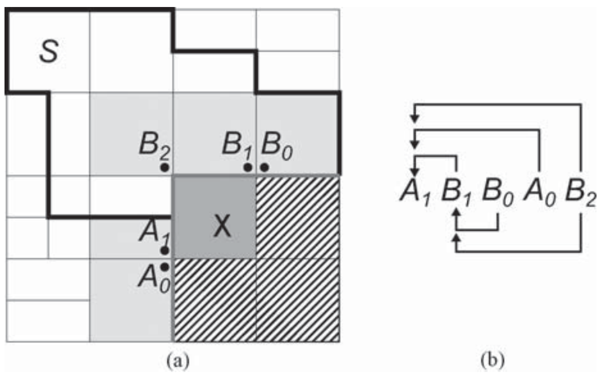


Fig. 5. (a) Illustration of possible merge candidates for block *X* in an example PB partitioning in HEVC. Multiple PBs are merged into a region (bold line). (b) Redundancy checks prior to adding a candidate to the list. For each candidate to check, outgoing arrows point to a candidate for comparison.

three spatially neighboring MVs. HEVC improves the MV prediction by applying an MV prediction competition as initially proposed in [18]. In HEVC, this competition was further adapted to large block sizes with so-called *advanced motion vector prediction* (AMVP) in [19]. In the DIS *Main profile*, AMVP has two predictor candidates competing for the prediction. Two spatial motion vector predictor (MVP) candidates are considered and, when at least one of them is not available or they are redundant, a temporal motion vector prediction (TMVP) candidate is considered. The candidates are selected among the positions shown in Fig. 4: three spatial MVP candidate positions located above the current PB (B_0, B_1, B_2) and two on the left (A_0, A_1) as well as two TMVP candidate positions (C_0, C_1). The position selected to derive the TMVP is C_0 in Fig. 4(b). Additionally, to tackle unavailability of the MV derived from position C_0 and to enable CTU-aligned motion data compression, storage, and fetch, the position used to derive the TMVP is kept inside the row of processed CTUs. The following design is applied: use position C_0 if available and inside the processed CTU row, use centered position C_1 otherwise. When the AMVP candidate list contains less than two candidates, the list is filled up with zero MVs.

HEVC further defines a picture parameter set flag that specifies whether the TMVP can be used or not. This allows the encoder to perform an adaptation of the set of predictors (spatio-temporal or spatial only) at the picture level, to increase coding efficiency purpose for specific contents, or improve error resilience for some applications.

Moreover, the TMVP requires the storage of the motion data (including MVs, reference indices and coding modes) in reference pictures. Considering the granularity of motion representation, the memory size needed for storing motion data could be significant. HEVC includes *motion data storage reduction* (MDSR) to reduce the size of the motion data buffer and the associated memory access bandwidth by subsampling motion data in the reference pictures [20]. While H.264/AVC is storing these information on a 4×4 block basis, HEVC uses a $16{\times}16$ block where in case of subsampling a 4×4 grid, the information of the top-left 4×4 block is stored.

IV. HEVC Block Merging Algorithm

This section presents the details of the block merging algorithm in the HEVC design. Description of the core algorithm, candidate list construction, parsing, and parallel processing/throughput considerations are described in the following.

A. Block Merging Core Algorithm

In order to illustrate the core operation of the HEVC block merging algorithm, Fig. 5 shows a possible PB partitioning. The purpose of block merging, as already explained in Section II, is to compensate for the disadvantages of an initial quadtree-based subdivision by reducing the redundant sets of coding parameters to be transmitted. This is achieved by creating regions composed of neighboring PBs with associated PUs sharing identical motion information, which is only signaled once for each region. Thus, each region must contain at least one PB of a PU, which is not merged with a neighbor and seeds new motion information. In Fig. 5, PB *S* corresponds to such a particular block, whereas PB *X* corresponds to the current PB to be coded.

All the PBs along the block scanning pattern coded before the current PB form the set of causal PBs. The block scanning pattern for the CTUs (the tree roots) is raster scan and within a CTU, the CUs (leaves) and the associated PBs within are processed in depth-first order along a *z*-scan. Hence, the PBs in the striped area of Fig. 5 are successors (anticausal blocks) to PB *X* and do not have associated prediction data yet. In any case, the current PB *X* can be merged only with one of the causal PBs. When *X* is merged with one of these causal PBs, the PU of *X* copies and uses the same motion information as the PU of this particular block. Out of the set of causal PBs, only a subset is used to build a list of merge candidates. In order to identify a candidate to merge with, an index to the list is signaled to the decoder for block *X*.

We should point out that the merge candidate seeding the prediction data is not necessarily the PU of a spatially neighboring PB like *S* in Fig. 5. It can also be a candidate with prediction data from a temporally collocated PU or an additional candidate with prediction data derived otherwise. As a consequence, the merged region represented by the bold line in Fig. 5 possibly extends to the temporal dimension as a 3-D region with similar motion data.

B. Candidate List Construction

An overview of the merge candidate list construction process is given in Fig. 6. The output of this process is a list of merge candidates, each of them being a tuple of motion parameters, which can be used for motion-compensated prediction (MCP) of a block. That is, the information on whether one or two motion compensated hypothesis are used for prediction, and for each hypothesis an MV and a reference picture index. The number of candidates within this list is controlled by *NumMergeCands*, which is signaled in the slice header. The whole process of adding candidates will stop as soon as the number of candidates reaches *NumMergeCands* during the following ordered steps: the process starts with the derivation of the initial candidates from spatially neighboring PBs, called spatial candidates. Then, a candidate from a PB in a temporally collocated picture can be included, which is called the temporal candidate. Due to unavailability of certain candidates or some of them being redundant, the number of initial candidates could be less than *NumMergeCands*. In this case, additional "virtual" candidates are inserted to the list so that the number of candidates in the list is always equal to *NumMergeCands*. Having a fixed number of candidates is part of the means to achieve robustness against transmission errors as is explained in Section IV-C.

1) *Initial Candidates:* The initial candidates are obtained by first analyzing the spatial neighbors of the current PB, i.e., the PBs covering the positions A_1, B_1, B_0, A_0, and B_2, as in Fig. 5. Note that these are the same positions used for AMVP described in Fig. 4. A candidate is only available to be added to the list when the corresponding PU is inter-picture predicted and does not lie outside of the current slice. Moreover, a position-specific redundancy check is performed so that redundant candidates are excluded from the list to improve the coding efficiency. First, the candidate at position A_1 is added to the list if available. Then, the candidates at the positions B_1, B_0, A_0, and B_2 are processed in that order. Each of these candidates is compared to one or two candidates at a previously tested position and is added to the list only in case these comparisons show a difference. These redundancy checks are depicted in Fig. 5(b) by an arrow each. Note that instead of performing a comparison for every candidate pair, this process is constrained to a subset of comparisons to reduce the complexity while maintaining the coding efficiency [21].

Moreover, the candidate at position A_1 can only be added into the list if the current PB, corresponding to X in Fig. 5(a), is not the left block of a CB vertically split into two partitions (PART_N×2N or PART_nL×2N or PART_nR×2N partitions in Fig. 2). In the same manner, the candidate at position B_1 can only be added into the list if the current PB is not the bottom block of a CB horizontally split into two partitions (PART_2N×N or PART_2N×nU or PART_2N×nD partitions in Fig. 2). Under these conditions, the respective candidate is not added, as merging with this candidate would lead to two PUs in a CU having the same motion information, which is redundant to just have one PU in a CU. Finally, no more than four out of these five potential spatial candidates are added into the merge candidate list.

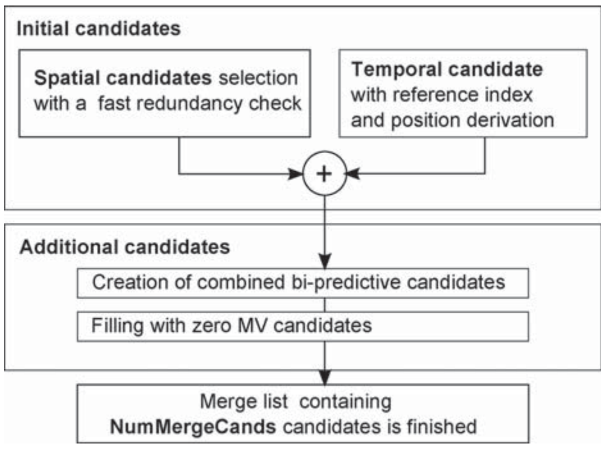

Initial candidates
Spatial candidates selection with a fast redundancy check
Temporal candidate with reference index and position derivation
+
Additional candidates
Creation of combined bi-predictive candidates
Filling with zero MV candidates
Merge list containing NumMergeCands candidates is finished

Fig. 6. Major steps for the construction of the block merging candidate list in HEVC.

After the spatial candidates, a temporal merge candidate is added to the list, if available. The temporal merge candidate includes one or two MVs depending on their availability in the collocated PU. The position of the temporal PB associated with the collocated PU is derived the same way as for AMVP, as described in Section III-B. Each reference index of the temporal merge candidate is set to zero. This avoids decoding dependency between PUs in a CU. No redundancy check is performed after adding the temporal merge candidate to the list. This improves the error resilience, as the candidate list construction is less dependent on the motion information of the temporal candidate, which belongs to a different slice and may be prone to loss. In addition, the complexity is reduced by eliminating four motion data comparisons in the worst case.

2) *Additional Candidates:* The candidate list is completed to contain *NumMergeCands* candidates by combining motion information already in the list, to form new bi-predictive candidates and by adding zero candidates [22]. The combined bi-predictive candidates are generated by combining the first reference picture list motion parameters of an initial candidate with the second reference picture list motion parameters of another. Provided these two parameter tuples reference different motion hypotheses, they will form a new bi-predictive candidate. The zero candidates have zero spatial displacement and an increasing reference picture index, starting at zero for the first added zero candidate. They use two prediction hypotheses in B-slices and one in P-slices. If the number of candidates in the list is still less than *NumMergeCands* after adding zero candidates for all valid reference picture indices, the list is completed by adding zero candidates with reference picture index set to zero. Even if some redundancy still exists between additional and initial candidates in the list, no redundancy check is done for the additional candidates to save complexity, while maintaining coding efficiency [23].

C. Signaling and Parsing Robustness

Block merging in HEVC features a mode, which signals at the CU level that the prediction residual is zero and that the CU contains a single inter-picture predicted PU (PART_2N×2N partition in Fig. 2) [24], thus being conceptually very similar to the H.264/AVC *skip mode*. Prior to any other information associated with a particular CB, this mode is signaled by a flag, called *skip flag*. Instead of the median-based approach in H.264/AVC, the motion information of the PU associated with a *skipped* CU is inferred by block merging. If the CU is not *skipped*, a *merge flag* is sent for each inter-picture predicted PU. If *skip flag* or *merge flag* is set, i.e., block merging is to be used, a *merge index* is sent to identify one of the candidates.

As mentioned in the previous section, the set of merge candidates includes a temporal candidate. Therefore, the signaling has to be carefully designed to provide satisfying parsing throughput and in case of a frame loss, to ensure that the bitstream remains parsable, avoiding in particular any conditional coding of the merge flag and the merge index. These issues have been studied in [20] and [25] and a method to explicitly signal the length of the candidate list beforehand, instead of deriving it from the list construction process, has proven to be the best among several alternative methods. First, the length of the candidate list is controlled at slice level by *NumMergeCands*, which can be between 1 and 5. Second, the code used to signal merge index for each block in the slice solely depends on the length of the list. This way, parsing remains independent of the list construction process, and is still possible, e.g., after the temporal candidate was lost due to transmission errors. Also, this is friendly to architectures on which bitstream needs to be preparsed before full decoding.

D. Parallel Processing and Throughput

The usage of directly neighboring motion information by block merging introduces tight data dependencies, so care must be taken to allow for parallelizable and high-throughput implementations. Therefore, block merging includes the following features.

1) *Motion Estimation Regions:* Many of the embedded encoders perform motion estimation in parallel for the neighboring blocks or pipeline the motion estimation steps to improve throughput. Typically, parallel motion estimation is performed such that all the MVs of PUs inside a region are estimated in parallel. Let us call these regions used for parallel motion estimation *motion estimation regions* (MER). For the purpose of estimating motion information of a PU inside an MER, motion information of other PUs within the same MER is not used. This implies that the estimation cannot take into account data dependencies as in the candidate list construction process for block merging. In turn, this parallel motion estimation must estimate the coding cost less accurately, sacrificing coding efficiency. In order to limit this coding efficiency loss associated with a parallel motion estimation implementation, HEVC includes a signaling mechanism to inform the decoder what size of MER was utilized at the encoder [26]. When this mechanism is used, the merge candidate list construction process is modified such that the candidates associated with the PUs/CUs inside the same MER are marked as unavailable.

2) *One Candidate List Per CU:* In the block merging design, each PU within a CU can have a different block merging candidate list. Encoder and decoder need to access different memory locations depending on the partitioning (see Fig. 2) to construct the candidate list for each PU, which can reduce the throughput. In order to improve the throughput while maintaining the coding efficiency, HEVC can be configured that all PUs associated with a CU with an 8×8 luma samples CB share the same block merging candidate list [27].

3) *Candidate Specific Redundancy Removal:* As described in Section IV-B, the candidate list construction process performs only a subset of all comparisons necessary to avoid any duplicate candidate. This restriction has virtually no effect on coding efficiency but reduces the number of gates or cycles in a hardware implementation [21].

V. Relation of Block Merging to Direct Modes

There is a conceptual similarity between the block merging approach and what is commonly named *direct prediction* or *direct mode* [16]. Direct mode also infers the motion information for MCP from previously decoded blocks. Thus, both algorithms are using motion information from neighboring blocks in order to reduce the bit rate required for transmitting motion parameters. Despite this similarity, the two algorithms differ in the way they handle neighboring motion information. While direct mode infers motion parameters from adjacent blocks, the block merging core algorithm creates regions within which all blocks share the same set of motion parameters.

To be more precise, the version of direct mode well known and referenced here is called the *spatial direct mode* of H.264/AVC [5], [16]. It can be applied to macroblocks and sub-macroblocks of 8×8 luma samples in a bi-predictively coded slice. For each block, the motion information is derived from up to four spatial neighboring blocks, namely the above, left, above-right and when above-right is not available, the above-left neighbor. The reference indices for each of the two reference picture lists are determined by selecting the minimum available reference index from the respective list among the three neighboring blocks. If the reference indices for both lists are negative, they are inferred to be zero and both MVs are set to zero as well. Otherwise, only the list associated with an available reference index is used for prediction and the corresponding MV is derived by using the same procedure as for the H.264/AVC MV predictor, i.e., median of the neighboring MV components.

The *spatial direct mode* of H.264/AVC inspired also the development of a similar mode in HEVC. This direct mode was originally a part of HEVC and is further referred to as *WD1 direct mode*, because WD1 is the last version that incorporates this mode [28]. Unlike block merging, it is only used for PBs of the same size as the containing CB (PART_2N×2N partition in Fig. 2). When the direct mode is signaled for a CU in an uni-prediction slice, only one reference picture list is used whereas in a bi-prediction slice, it is additionally signaled whether two hypotheses are used and in case of one hypothesis, which list is used (inter_pred_idc). While the reference index for each hypothesis is set equal to zero, the MV is set equal to the MVP chosen from a list created by AMVP as presented in Section III-B. Consequently, an MVP index has to be signaled for each hypothesis. Based on this direct mode, a *skip mode* similar to the H.264/AVC skip mode is also present in WD1. The differences to the direct mode are that it is signaled before the prediction mode using a separate flag, the associated residual signal is zero and for bi-predictive slices, always two hypotheses are used instead of signaling the reference picture lists explicitly (inter_pred_idc).

For WD1, a core experiment on MV coding evaluated the impact of replacing the WD1 direct mode with block merging. An improved coding efficiency together with an encoding time reduction for merge lead to the replacement of WD1 direct mode in WD2. Note that the related skip mode that uses AMVP predictors has not been changed. In order to investigate whether a skip mode that uses merge to derive motion information is more efficient, another core experiment on MV coding was established. It has been reported that with comparable encoder and decoder runtimes, the merge skip mode improved the average coding performance. Consequently, the skip mode in WD3 was replaced by the merge skip mode described in Section IV-C. Detailed results and further references for all the conducted tests can be found in the break out group reports on these core experiments [24], [29]. A performance comparison between the presented HEVC merge and merge skip mode with the WD1 direct and skip mode can be found in Section VI-D.

VI. Experimental Results

In this section, we evaluate the impact of the block merging algorithm in HEVC and its behavior depending on different types of sequences and coding configurations. We also show how different components of the merge scheme interact and help for coding efficiency. Additionally, we compare the RD performance of block merging to that of WD1 direct mode. All presented bit-rate savings are measured in terms of the Bjøntegaard Delta (BD) rate [30], [31]. The BD rate values are calculated from a total of four rate points as defined in the JCT-VC common test conditions [32]. Note that negative BD rate values indicate bit-rate savings in comparison to the reference.

A. Simulation Environment and Settings

1) *Software:* The environment we use to perform our experiments is the HEVC software test model version HM8.0 [12], which we refer to as HM8.0 *Default* in the sequel. We obtain our results by comparing this version and two modified versions.

To evaluate the overall performance of the block merging algorithm, we use a version referred to as HM8.0 without block merging. This version is identical to the HM8.0 *Default* but with the block merging algorithm and the associated syntax (merge flag, merge index, and skip flag) removed.

To compare block merging with *WD1 direct mode* described in Section V, we reintegrated this mode into HM8.0. In summary, we modified HM8.0 by replacing the merge and merge skip mode with the WD1 direct and skip mode.

TABLE I

BD Rates [%] of HM8.0 *Default* With Block Merging and HM8.0 With *WD1 Direct* Mode, Both With Reference to HM8.0 Without Block Merging or Direct Mode

Sequence	**DIS merge**			**WD1 direct**		
	RA	**LB**	**LP**	**RA**	**LB**	**LP**
Class A						
Traffic	−8.8	n/a	n/a	−5.8	n/a	n/a
PeopleOnStreet	−6.5	n/a	n/a	−4.1	n/a	n/a
Nebuta	−1.5	n/a	n/a	−0.5	n/a	n/a
SteamLocomotive	−11.1	n/a	n/a	−8.7	n/a	n/a
Class B						
Kimono	−9.6	−7.8	−5.4	−7.0	−4.5	−4.0
ParkScene	−7.3	−7.0	−5.6	−4.8	−2.7	−3.2
Cactus	−11.7	−8.8	−6.9	−7.7	−3.9	−4.1
Basketball Drive	−9.1	−8.1	−6.1	−6.9	−5.1	−4.2
BQTerrace	−10.2	−10.3	−5.9	−7.0	−4.8	−2.9
Class C						
Basketball Drill	−7.5	−7.6	−6.1	−4.9	−3.0	−3.0
BQMall	−9.2	−7.5	−5.7	−5.5	−2.8	−3.0
PartyScene	−4.8	−3.4	−2.5	−2.8	−0.8	−0.7
RaceHorses	−3.9	−4.3	−3.3	−2.0	−1.8	−1.6
Class D						
Basketball Pass	−6.1	−5.1	−4.0	−3.6	−2.0	−1.9
BQSquare	−6.9	−3.3	−2.5	−4.2	−0.2	0.1
BlowingBubbles	−6.1	−3.7	−3.1	−3.9	−0.7	−1.0
RaceHorses	−4.4	−3.9	−3.6	−2.2	−1.1	−1.5
Class E						
FourPeople	n/a	−11.4	−8.5	n/a	−6.1	−5.8
Johnny	n/a	−20.0	−16.4	n/a	−12.8	−12.6
KristenAndSara	n/a	−15.0	−11.2	n/a	−9.3	−8.3
Average	**−7.3**	**−8.0**	**−6.0**	**−4.8**	**−3.8**	**−3.6**

2) *Configuration:* All used HM versions were configured following the JCT-VC common test conditions embodying the DIS *Main profile*. In these common test conditions, the quadtree structure, presented in Section III-A has CTUs with a fixed CTB size of 64 $\times$ 64 luma samples. The maximum coding quadtree depth is set to 3, resulting in CUs with a minimum CB size of 8×8 luma samples. For such a minimum CU, only the symmetric rectangular partitions depicted in Fig. 2 are allowed, resulting in a minimum inter-PB size of 4 $\times$ 8 or 8 $\times$ 4 luma samples.

Corresponding to different applications of a video codec, three coding configurations for the DIS *Main profile*, restricting the temporal coding structure, are defined in [32]. The random access (*RA*) configuration restricts the structural temporal delay to at most 8 pictures and the RA intervals to not exceed 1.1 s. In the low delay B (*LB*) and low delay P (*LP*) configurations, no picture reordering is allowed in the decoder between the decoding and output process. In the *LP* configuration, inter-picture prediction is restricted to one hypothesis whereas, in *RA* and *LB* configurations, up to two hypotheses are allowed for inter-picture prediction.

B. Overall Performance of Block Merging

When comparing HM8.0 *Default* to HM8.0 without block merging, we observe that block merging yields RD-performance gains from −6% to −8% BD rate. All results of this comparison are listed in the left half of Table I.

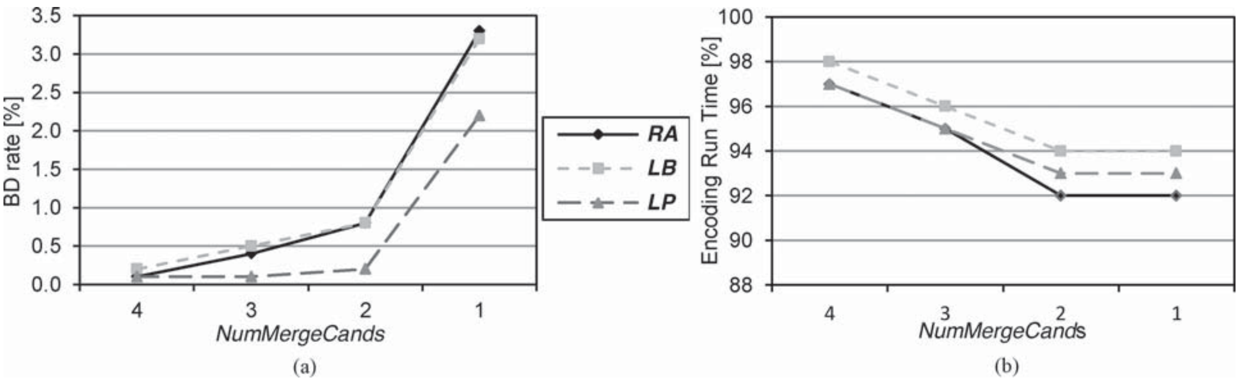


Fig. 7. Impact of the number of block merging candidates: (a) average bit rate losses and (b) encoding time with different numbers of block merging candidates (*NumMergeCands*) compared to HM8.0 *Default* (where *NumMergeCands* = 5).

The average bit rate savings of block merging are almost the same for *RA* and *LB* configurations. For the particular case of *LP*, these gains are more limited because inter-picture prediction is restricted to one hypothesis in this configuration and consequently the motion information savings through the block merging algorithm are reduced. For the common classes (B, C, and D) between *RA* and *LB*, the average bit rate savings of block merging are slightly better for *RA* (1.2% improvement in average in comparison with *LB*). Even though for both configurations up to two hypotheses are allowed in inter-picture prediction, the hierarchical B picture coding structure of *RA* allows more combinations of these hypotheses, which are more efficiently signalized through block merging.

There are several sequences where coding efficiency lies significantly above the average such as, e.g., all class E sequences. These particularly favorable results for class E are mostly due to the static background of these sequences which can be easily represented by using a single merged region using a zero MV.

C. Performance Analysis of Candidate List Construction

As an essential part of block merging, we now focus on the candidate selection strategy. We therefore measure the impact of changes to the block merging candidate list construction process in terms of RD performance and complexity. The tested version for each of these experiments is HM8.0 with a reduced set of block merging features to be compared against HM8.0 *Default*. It should be noted that, in this section, the loss observed in performance indicates the efficiency of each element in the current block merging algorithm. As an estimate of the encoder complexity, encoding run times are reported as a fraction of a reference encoding time. Decoding run times for all tested versions are very similar and are not reported.

1) *Number of Merge Candidates:* Fig. 7 displays the tradeoff between coding efficiency and complexity caused by changing the number of merge candidates (*NumMergeCands*). Compared to a value of five candidates in HM8.0 *Default*, Fig. 7(a) presents the performance loss when the number of block merging candidates is decreased. It is observed that the bit rate loss increases only slightly when *NumMergeCands* decreases to 2. A significant loss is noticed when the list is limited to one candidate, reflecting the efficiency of the MV

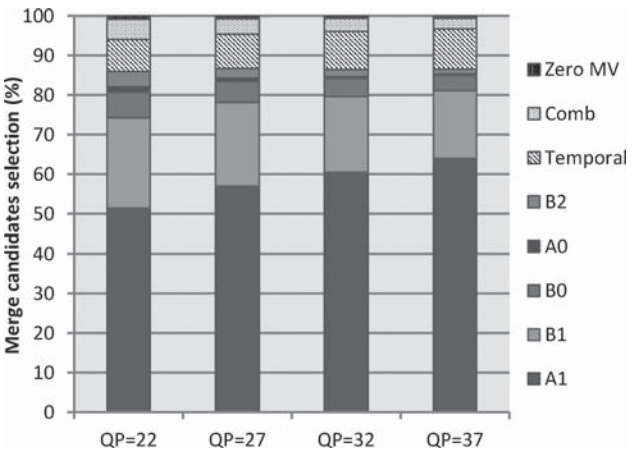


Fig. 8. Frequency of selection for different types of block merging candidates. The candidates are represented in the order they are added to the candidate list, with the first at the bottom of each bar. These average values are obtained from HM8.0 *Default* using the *RA* configuration and different QP values.

predictor competition. Furthermore, Fig. 7(b) gives a rough measure of the encoder complexity for HM8.0 with different values of *NumMergeCands* in comparison with HM8.0 *Default*. We observe an overall decrease of encoder run time approximately linear with the number of merge candidates until the run times saturate when the number of merge candidates equals 2. The encoding run time significantly depends on the number of merge candidates, because the HM encoder performs a motion compensation operation for each candidate to evaluate its RD cost. The possibility of configuring the total number of merge candidates can serve as a tool to trade off complexity versus RD performance.

2) *Spatial Candidates:* How frequently the different merge candidates are selected is shown in Fig. 8. The spatial candidates' portions are represented by the bars shaded in plain gray at the bottom of each column. The selection of the spatial candidates is on average very high, around 86%. Among the spatial candidates, the frequent selection of position A_1 can be explained by the binarization scheme used for the merge index. The candidate from position A_1 is at the first position in the candidate list and thus only a single bin is used to binarize its index, giving this candidate an advantage in the

TABLE II

Influence on Coding Efficiency of Disabling Temporal MV Prediction and Additional Combined Bi-Predictive Candidates in BD Rate (%) Compared to HM8.0 *Default*

	Temporal MV prediction		**Additional combined candidates disabled**
	Completely disabled	**Disabled only for Merge**	
RA	2.6	1.8	0.3
LB	2.6	1.5	0.4
LP	2.3	0.9	0.0

RD optimization. When the QP value increases, this effect becomes even more evident because of the lower dispersion of MV values connected to higher QP values.

3) *Temporal MV Prediction:* As described in Sections III-B and IV-B, a temporal MV prediction is included for AMVP, and a temporal merge candidate can be included in the merge list. Table II shows BD rate losses of 2.6%, 2.6%, 2.3%, for the *RA*, *LB*, and *LP* configurations, respectively, when the temporal MV prediction is disabled for both block merging and AMVP. The loss reaches 3.9% for the BQSquare and BQMall sequences. Similarly, significant losses are obtained when the temporal MV prediction is disabled for block merging only, confirming the usefulness of this prediction. Fig. 8 shows that for the *RA* configuration, 9% of all merged blocks select the temporal candidate, which also indicates its usefulness.

4) *Additional Candidates:* Table II presents also the performance losses associated with deactivating the additional combined merging candidates, i.e., the list is completed using only zero candidates. For for the *RA* and *LB* configurations the loss associated to this modification is around 0.3% and 0.4%. No bit rate loss is observed for the *LP* configuration. For this configuration, where only one inter-picture prediction hypothesis is used, combined candidates are not applicable as described in Section IV-B. The impact of the additional combined candidates is also visible in Fig. 8 with a relative frequency of 4% on average for the *RA* configuration.

D. Comparison to Direct Mode

In Table I, the last three columns show the results for the WD1 direct mode implemented in HM8.0. The reference is again HM8.0 without block merging or direct mode, which is the same as for the comparison with DIS block merging in the previous columns.

Regarding the overall performance, two things can be observed. First, the WD1 direct mode shows significantly less bit rate reduction on average compared to DIS block merging. When taking the DIS block merging results from Table I and calculating the BD rate with WD1 direct results as reference, the average BD rate gains for block merging are −2.7%, −4.2%, and −2.6% for the *RA*, *LB*, and *LD* configurations, respectively. Second, the same tendencies as reported for block merging, e.g., more bit rate reduction when bi-predictive slices are used, are showing up for the WD1 direct mode. This shows that both modes operate in a similar way whereas block merging is better suited to exploit motion parameter redundancies in the quadtree-based HEVC.

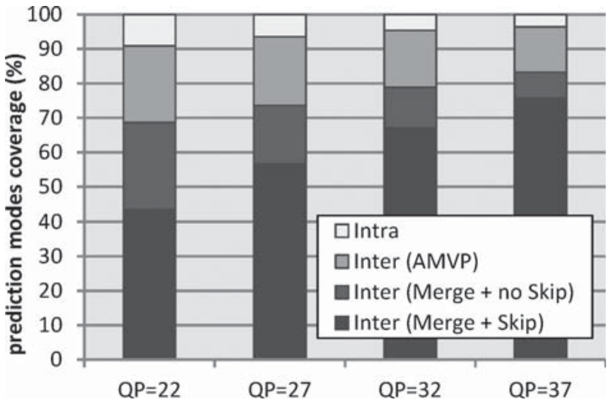


Fig. 9. Sample coverage fractions for intra- and inter-picture prediction modes. In case of inter-picture prediction, statistics are separated for PBs coded with AMVP or with block merging. Different fractions are shown for merged blocks with the skip flag set (Merge + Skip) or cleared (Merge + no Skip). The average values are obtained from HM8.0 *Default* using the *RA* configuration for all test sequences.

When looking at the detailed results, it can be noticed that the gains for DIS block merging over WD1 direct mode are showing up over all test sequences.

E. Sample Coverage of Prediction Modes

To give an insight on how block merging competes with the other coding modes, we analyzed the number of samples predicted by the different modes. Fig. 9 compares the fraction of all samples covered by intra- and inter-picture prediction modes for the *RA* configuration. Furthermore, for the case of inter-picture prediction, two distinct categories are shown, which are the samples coded with AMVP and the samples coded with block merging. Different fractions are shown for merged blocks with the skip flag set (*skip* CUs) or cleared. The fraction of samples associated with *skip* CUs represent an average of 55%. It can also be observed that an average of 76% of the samples are coded with block merging for the *RA* configuration, which is another indication of the efficiency of this algorithm. Another observation is that the fraction of samples coded with block merging increases when the value of the quantization parameter QP increases. This can be explained by two reasons. When the QP increases, the RD decision process will tend to choose larger blocks with a limited dispersion of the MVs, leading to an increased quantity of samples coded with block merging. Another consequence of an increased QP value is an increased amount of blocks with zero quantized residual. These are efficiently coded using *skip* CUs.

F. Complexity

As a rough measure of computational complexity of block merging, encoding, and decoding run times have been measured on the same hardware platform. When we compare HM8.0 *Default* (which uses *NumMergeCands* equal to 5) to HM8.0 without block merging, we measure an increase of 20% in encoding time and an increase of 3% in decoding time on average. The comparatively small increase in decoder runtime is mostly due to the additional operations needed to compile the candidate list. As we already analyzed in VI-C1, the encoder's run time is significantly influenced by the number of merge candidates. This is mostly due to the accurate, yet computationally expensive RD cost calculation performed by the HM encoder for each merge candidate. Of course, what value of *NumMergeCands* is the most favorable operating point for a given application depends on the importance attributed to the two criteria of RD performance and complexity. Also, different encoders might use faster, approximate cost estimation techniques.

Furthermore, we give results on the techniques of MERs and the candidate list restriction presented in IV-C that aim to reduce complexity of block merging. The coding efficiency impact of the signalling mechanism for MERs is as follows: When the mechanism for MERs of size 16×16 is utilized, which is a typical size in hardware implementations, the average loss is reported to be 0.6% on the average for all common test conditions [26]. On the other hand, if the mechanism is not used and if an encoder performs parallel motion estimation within regions of size 16×16, then the loss increases to 3.2% on the average. This shows, that the losses caused by parallel motion estimation at the encoder can be dramatically reduced. For the *one candidate list per CU* technique, enabling the tool is reported to result in a 0.1% loss on the average [27]. Therefore, it can be seen as a beneficial tradeoff between throughput and coding efficiency.

A general overview of HEVC complexity can be found in [33].

VII. Conclusion

Block merging together with a quadtree-based block-partitioning concept has been presented in this paper as an alternative to the conceptually similar direct mode for MCP. As the proposed block merging algorithm better exploits spatio-temporal correlations in motion model parameters, it has been chosen to replace the direct mode in HEVC. Simulation results show the overall performance improvement of block merging with average BD rate savings from 6% to 8%. This gain is due to various components of the block-merging algorithm, which were adapted and optimized for the HEVC environment in the standardization process.

Appendix A
Downloadable Resources Related to this Paper

All the JCT-VC documents can be found in the JCT-VC document management system at http://phenix.int-evry.fr/jct/. All cited VCEG and JVT documents are also publicly available and can be downloaded at http://wftp3.itu.int/av-arch in the "video-site" and "jvt-site" folder, respectively.

Acknowledgment

The authors would like to thank all experts in the ISO/IEC and ITU-T Joint Collaborative Team on Video Coding (JCT-VC) that have contributed to study and improve block merging within the HEVC project.

References

[1] P. Chou, T. Lookabaugh, and R. Gray, "Optimal pruning with applications to tree-structured source coding and modeling," *IEEE Trans. Informat. Theory*, vol. 35, no. 2, pp. 299–315, Mar. 1989.

[2] G. J. Sullivan and R. L. Baker, "Efficient quadtree coding of images and video," *IEEE Trans. Image Process.,* vol. 3, no. 3, pp. 327–331, Jan. 1994.

[3] R. Shukla, P. L. Dragotti, M. N. Do, and M. Vetterli, "Rate-distortion optimized tree-structured compression algorithms for piecewise polynomial images," *IEEE Trans. Image Process.,* vol. 14, no. 3, pp. 343–359, Mar. 2005.

[4] R. De Forni and D. Taubman, "On the benefits of leaf merging in quad-tree motion models," in *Proc. IEEE Int. Conf. Image Process.*, Sep. 2005, pp. II–858.

[5] T. Wiegand, G. J. Sullivan, G. Bjøntegaard, and A. Luthra, "Overview of the H.264/AVC video coding standard," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 13, no. 7, pp. 560–576, Jul. 2003.

[6] M. Tagliasacchi, M. Sarchi, and S. Tubaro, "Motion estimation by quadtree pruning and merging," in *Proc. IEEE Int. Conf. Multimedia Expo.*, Jul. 2006, pp. 1861–1864.

[7] D. Marpe, H. Schwarz, S. Bosse, B. Bross, P. Helle, T. Hinz, H. Kirchhoffer, H. Lakshman, T. Nguyen, S. Oudin, M. Siekmann, K. Sühring, M. Winken, and T. Wiegand, "Video compression using nested quadtree structures, leaf merging, and improved techniques for motion representation and entropy coding," *IEEE Trans. Circuits Syst.*, vol. 20, no. 12, pp. 1676–1687, Dec. 2010.

[8] S. Oudin, P. Helle, J. Stegemann, C. Bartnik, B. Bross, D. Marpe, H. Schwarz, and T. Wiegand, "Block merging for quadtree-based video coding," in *Proc. IEEE Int. Conf. Multimedia Expo.*, Jul. 2011, pp. 1–6.

[9] M. Winken, S. Bosse, B. Bross, P. Helle, T. Hinz, H. Kirchhoffer, H. Lakshman, D. Marpe, S. Oudin, M. Preiß, H. Schwarz, M. Siekmann, K. Sühring, and T. Wiegand, *Description of Video Coding Technology Proposal by Fraunhofer HHI*, document JCTVC-A116, Apr. 2010.

[10] Video Coding Experts Group of ITU (VCEG), *Joint Call for Proposals on Video Compression Technology*, document VCEG-AM91 of ITU-T Q6/16 and N1113 of JTC1/SC29/WG11, Jan. 2010.

[11] B. Bross, W.-J. Han, J.-R. Ohm, G. J. Sullivan, and T. Wiegand, *High Efficiency Video Coding (HEVC) Text Specification Draft 8*, document JCTVC-J1003, Jul. 2012.

[12] JCT-VC. (2012). *Subversion Repository for the HEVC Test Model Version HM8.0* [Online]. Available: https://hevc.hhi.fraunhofer.de/svn/svn_HEVCSoftware/tags/HM-8.0/

[13] G. J. Sullivan, J.-R. Ohm, W.-J. Han, and T. Wiegand, "Overview of the High Efficiency Video Coding (HEVC) standard," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 22, no. 12, pp. 1648–1667, Dec. 2012.

[14] J.-R. Ohm, G. J. Sullivan, H. Schwarz, T. K. Tan, and T. Wiegand, "Comparison of the coding efficiency of video coding standards—Including High Efficiency Video Coding (HEVC)," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 22, no. 12, pp. 1668–1683, Dec. 2012.

[15] R. Mathew and D. S. Taubman, "Quad-tree motion modeling with leaf merging," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 20, no. 10, pp. 1331–1345, Oct. 2010.

[16] A. M. Tourapis, F. Wu, and S. Li, "Direct mode coding for bi-predictive slices in the H. 264 standard," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 15, no. 1, pp. 119–126, Jan. 2005.

[17] M. Winken, P. Helle, D. Marpe, H. Schwarz, and T. Wiegand, "Transform coding in the HEVC test model," in *Proc. IEEE Int. Conf. Image Process.*, Sep. 2011, pp. 3693–3696.

[18] G. Laroche, J. Jung, and B. Pesquet-Popescu, "RD optimized coding for motion vector predictor selection," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 18, no. 9, pp. 1247–1257, Sep. 2008.

[19] K. McCann, W.-J. Han, I.-K. Kim, J.-H. Min, E. Alshina, A. Alshin, T. Lee, J. Chen, V. Seregin, S. Lee, Y.-M. Hong, M.-S. Cheon, and N. Shlyakhov, *Samsungs Response to the Call for Proposals on Video Compression Technology*, document JCTVC-A124, Apr. 2010.

[20] B. Li and J. Xu, "Parsing robustness in High Efficiency Video Coding-analysis and improvement," in *Proc. IEEE Visual Commun. Image Process.*, Nov. 2011, pp. 1–4.

[21] O. Bici, J. Lainema, and K. Ugur, *CE9: Results of SP Experiments on Simplification of Merge Process*, document JCTVC-H0252, Feb. 2012.

[22] T. Sugio and T. Nishi, *Parsing Robustness for Merge/AMVP*, document JCTVC-F470, Jul. 2011.

[23] B. Li, J. Xu, and H. Li, *Non-CE9/Non-CE13: Simplification of Adding New Merge Candidates*, document JCTVC-G397, Nov. 2011.

[24] B. Bross, J. Jung, Y.-W. Huang, Y. H. Tan, I.-K. Kim, T. Sugio, M. Zhou, T. K. Tan, E. Francois, K. Kazui, W.-J. Chien, S. Sekiguchi, S. Park, and W. Wan, *BoG Report of CE9: MV Coding and Skip/Merge Operations*, document JCTVC-E481, Mar. 2011.

[25] J. Jung, B. Bross, J. Chen, P. Onno, and M. Zhou, *CE13: Summary Report of Core Experiment 13 on Motion Data Parsing Robustness and Throughput*, document JCTVC-G043, Nov. 2011.

[26] M. Zhou, *AHG10: Configurable and CU-Group Level Parallel Merge/Skip*, document JCTVC-H0082, Feb. 2012.

[27] K. Hui Yong, K. Kyung Yong, K. Sang Min, P. Gwang Hoon, C. Seunghyun, L. Sung-Chang, L. Jinho, and C. Jin Soo, *Non-CE9: Throughput Improvement for Merge/Skip Mode*, document JCTVC-H0240, Feb. 2012.

[28] T. Wiegand, W.-J. Han, B. Bross, J.-R. Ohm, and G. J. Sullivan, *WD1: Working Draft 1 of High-Efficiency Video Coding*, document JCTVC-C403, Oct. 2010.

[29] T. K. Tan, W.-J. Han, B. Bross, J. Jung, K. McCann, Y. Suzuki, G. Clare, H. Schwarz, and A. Fujibayashi, *BoG Report of CE9: Motion Vector Coding*, document JCTVC-D441, Jan. 2011.

[30] G. Bjøntegaard, *Calculation of Average PSNR Differences Between RD Curves*, document VCEG-M33, ITU-T Q6/16, Apr. 2001.

[31] G. Bjøntegaard, *Improvements of the BD-PSNR model*, document VCEG-AI11, ITU-T Q6/16, Jul. 2008.

[32] F. Bossen, *HM 8 Common Test Conditions and Software Reference Configurations*, document JCTVC-J1100, Jul. 2012.

[33] F. Bossen, B. Bross, K. Sühring, and D. Flynn, "HEVC complexity and implementation analysis," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 22, no. 12, pp. 1684–1695, Dec. 2012.

Philipp Helle received the Dipl.-Ing. degree in computer engineering from the Technical University of Berlin, Berlin, Germany, in 2009.

From 2004 to 2008, he was with MikroM-Mikroelektronik für Multimedia GmbH, Berlin, where he designed software and hardware components for video decoders dedicated to digital cinema. Since 2008, he has been with the Image and Video Coding Group, Fraunhofer Institute for Telecommunications-Heinrich Hertz Institute, Berlin. His current research interests include still image and video coding, signal processing, robotics, and computer vision.

Simon Oudin received the Dipl. Engineer degree from the Ecole Nationale Supérieure d'Electronique et de Radioélectricité de Grenoble, Grenoble, France, in 2006.

In 2005, during an exchange program with the Technical University of Berlin, Berlin, Germany, he took part on a H.264/AVC codec implementation project in the Communication Systems Group. From 2006 to 2009, he was with Allegro DVT, a startup in Grenoble, France, specialized in the H.264/AVC solutions. He has contributed to the creation and design of real-time video encoder/transcoder for the professional broadcast market. He is currently with the Image and Video Coding Group, Fraunhofer Institute for Telecommunications-Heinrich Hertz Institute, Berlin. His current research interests include video coding and more precisely motion prediction.

Benjamin Bross (S'11) received the Dipl.-Ing. degree in electrical engineering from Rheinisch-Westfaelische Technische Hochschule University Aachen, Aachen, Germany, in 2008.

During his studies, he focused on 3-D image registration in medical imaging and on decoder side motion vector derivation in H.264/AVC. He is currently with the Image Processing Group, Fraunhofer Institute for Telecommunications-Heinrich Hertz Institute, Berlin, Germany. His current research interests include motion estimation/prediction, residual coding, and contributions to the evolving HEVC standard.

Mr. Bross coordinates core experiments for the development of HEVC and co-chairs the editing *ad hoc* group since 2010. In July 2011, he was appointed as an editor of the HEVC video coding standard.

Detlev Marpe (M'00–SM'08) received the Dipl.-Math. degree (Highest Hons.) from the Technical University of Berlin (TUB), Berlin, Germany, in 1990, and the Dr.-Ing. degree from the University of Rostock, Rostock, Germany, in 2004.

Before joining the Fraunhofer Institute for Telecommunications–Heinrich Hertz Institute (HHI), Berlin, in 1999, he was a Research Assistant with TUB, University of Applied Sciences, Berlin, and University Hospital Charite, Berlin. He is currently the Head of the Image and Video Coding Group, Fraunhofer HHI. Since 1998, he has been an active contributor to the standardization activities of the ITU-T Visual Coding Experts Group, the ISO/IEC Joint Photographic Experts Group, and the ISO/IEC Moving Picture Experts Group for still image and video coding. In the development of the H.264/AVC standard, he was a Chief Architect of the CABAC entropy coding scheme, as well as one of the main technical and editorial contributors to the so-called fidelity range extensions with the addition of the High Profile in H.264/AVC. He was one of the key people in designing the basic architecture of Scalable Video Coding and Multiview Video Coding as algorithmic and syntactical extensions of H.264/AVC. He has authored or co-authored more than 200 publications in the areas of image coding and signal processing. He holds numerous internationally issued patents and patent applications in this field. His current research interests include still image and video coding, signal processing for communications as well as computer vision and information theory.

Dr. Marpe was a co-recipient of two Technical Emmy Awards as a Key Contributor and a co-editor of the H.264/AVC standard in 2008 and 2009, respectively. He was a recipient of the 2011 Karl Heinz Beckurts Award, the 2009 Best Paper Award of the IEEE Circuits and Systems Society, the Joseph-von-Fraunhofer Prize in 2004, and the Best Paper Award of the German Society for Information Technology in 2004. As a co-founder of the Berlin-based daviko GmbH, he was the winner of the Prime Prize of the 2001 Multimedia Start-Up Competition of the German Federal Ministry of Economics and Technology.

M. Oguz Bici received the B.S. and Ph.D. degrees in electrical and electronics engineering from Middle East Technical University (METU), Ankara, Turkey, in 2005 and 2010, respectively.

He was with the Multimedia Research Group, Department of Electrical and Electronics Engineering, METU, Turkey, from 2004 to 2010. Since 2010, he has been a Research Fellow with the Department of Signal Processing, Tampere University of Technology, Tampere, Finland, and an external Researcher with the Nokia Research Center, Tampere. His current research interests include coding and communication of multimedia content.

Kemal Ugur received the M.Sc. degree in electrical and computer engineering from the University of British Columbia, Vancouver, BC, Canada, in 2004, and the Doctorate degree from the Tampere University of Technology, Tampere, Finland, in 2010.

He is currently a Research Leader with Nokia, working on development of the next generation audiovisual technologies and standards. Since joining Nokia, he has been actively participating to several standardization forums, such as the Joint Video Team work on the standardization of the Multiview Video Coding extension of H.264/Moving Picture Experts Group-4 AVC, and the Video Coding Experts Group explorations toward a next generation video coding standard, the 3GPP activities for mobile broadcast and multicast standardization, and recently the Joint Collaborative Team on Video Coding for the development of High Efficiency Video Coding standard. He has authored more than 30 publications in academic conferences and journals and filed around 40 patent applications.

Dr. Ugur was a member of a research team that won the Nokia Quality Award in 2006.

Joël Jung received the Ph.D. degree from the University of Nice-Sophia Antipolis, Nice, France, in 2000.

From 1996 to 2000, he was with the I3S/CNRS Laboratory, University of Nice-Sophia Antipolis, involved in the improvement of video decoders based on the correction of compression and transmission artifacts. He joined Philips Research France in 2000 as a Research Scientist in video coding, postprocessing, perceptual models, objective quality metrics, and low-power codecs. He is currently with Orange Labs, Issy les Moulineaux, France, focusing on 2-D and 3-D video compression. He is an active participant in video compression standardization with several contributions to the Video Coding Expert Group ITU-T SG16-Q6, and to the JCT-VC. In particular, he was a Coordinator of several activities related to the motion prediction in HEVC. He holds more than 30 patents in the field of video coding.

Gordon Clare received the M.Sc. degree from the University of Auckland, Auckland, New Zealand, in 1984.

From 1985 to 1989, he was a Development Engineer with Rakon Computers, Sydney, Australia. From 1989 to 1991, he was with Software Development International, heading a team of developers creating a network management system for fault tolerant environments. From 1991 to 1997, he was with CISRA, a Canon Research Center, Sydney, focusing on real-time hardware and software image processing solutions. Moving to France in 1997, he was with several international companies developing document management systems and search engine. As a consultant from 2005 to 2010, he was developing H.264 and SVC solutions at Canon and France Telecom-Orange. Since 2010, he has been with Orange Laboratories, Rennes, France, continuing his work related to HEVC standardization and development.

Thomas Wiegand (M'05–SM'08–F'11) received the Dipl.-Ing. degree in electrical engineering from the Technical University of Hamburg-Harburg, Hamburg, Germany, in 1995, and the Dr.-Ing. degree from the University of Erlangen-Nuremberg, Erlangen, Germany, in 2000.

He is currently a Professor with the Department of Electrical Engineering and Computer Science, Berlin Institute of Technology, Berlin, Germany, chairing the Image Communication Laboratory, and is jointly heading the Image Processing Department of the Fraunhofer Institute for Telecommunications-Heinrich Hertz Institute, Berlin. From 1993 to 1994, he was a Visiting Researcher with Kobe University, Kobe, Japan. In 1995, he was a Visiting Scholar with the University of California, Santa Barbara. From 1997 to 1998, he was a Visiting Researcher with Stanford University, Stanford, CA, and served as a Consultant with 8x8, Inc., Santa Clara, CA. He joined the Heinrich Hertz Institute in 2000 as the Head of the Image Communication Group in the Image Processing Department. From 2006 to 2008, he was a Consultant with Stream Processors, Inc., Sunnyvale, CA. Since 2006, he has been a member of the Technical Advisory Board of Vidyo, Inc., Hackensack, NJ. From 2007 to 2009, he was a Consultant with Skyfire, Inc., Mountain View, CA. His current research interests include video processing and coding, multimedia transmission, and computer vision and graphics.

Dr. Wiegand has been an active participant in standardization for multimedia since 1995, with successful submissions to ITU-T VCEG, ISO/IEC MPEG, 3GPP, DVB, and IETF. In October 2000, he was appointed as the Associated Rapporteur of ITU-T VCEG. In December 2001, he was appointed as the Associated Rapporteur/Co-Chair of the JVT. In February 2002, he was appointed as the Editor of the H.264/MPEG-4 AVC video coding standard and its extensions (FRExt and SVC). From 2005 to 2009, he was a Co-Chair of MPEG Video. In 1998, he received the SPIE VCIP Best Student Paper Award. In 2004, he received the Fraunhofer Award and the ITG Award of the German Society for Information Technology. The projects that he co-chaired for development of the H.264/AVC standard have been recognized by the 2008 ATAS Primetime Emmy Engineering Award and a pair of NATAS Technology & Engineering Emmy Awards. In 2009, he received the Innovations Award of the Vodafone Foundation, the EURASIP Group Technical Achievement Award, and the Best Paper Award of IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS FOR VIDEO TECHNOLOGY. In 2010, he received the Eduard Rhein Technology Award. He was a Guest Editor for the IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS FOR VIDEO TECHNOLOGY for its Special Issue on the H.264/AVC Video Coding Standard in July 2003 and its Special Issue on Scalable Video Coding-Standardization and Beyond in September 2007. Since January 2006, he has been an Associate Editor of IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS FOR VIDEO TECHNOLOGY.