Faire jouer un programme
Algorithme min-max*
Optimisation α-β+

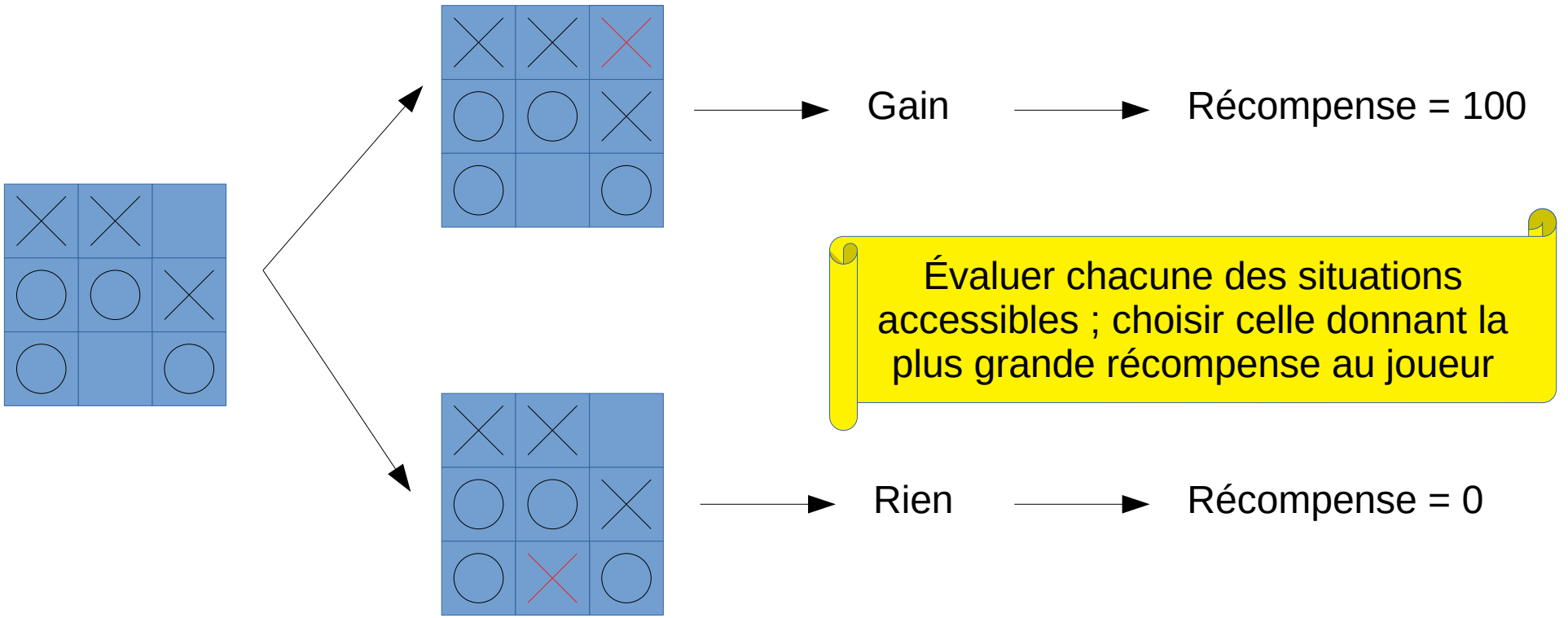
Bruno Mermet

2021

*Algorithme dû à Von Neumann, 1928
+Travail de McCarthy, 1956

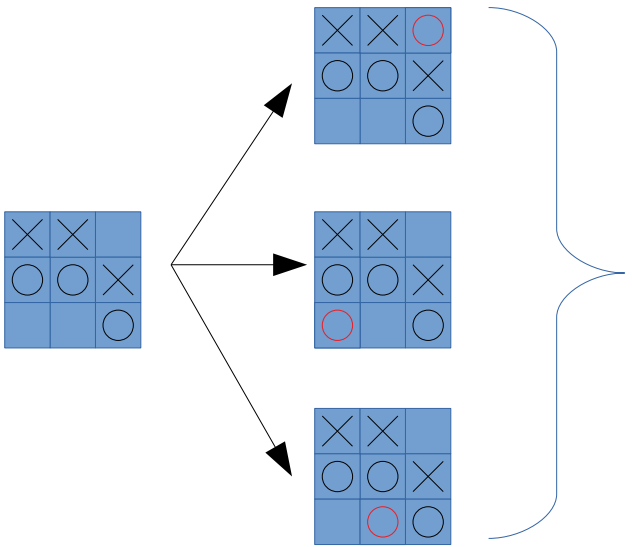
Exemple au Tic-Tac-Toe (1)

- c’est à  de jouer :



Exemple au Tic-Tac-Toe (2)

- C’est à ◯ de jouer

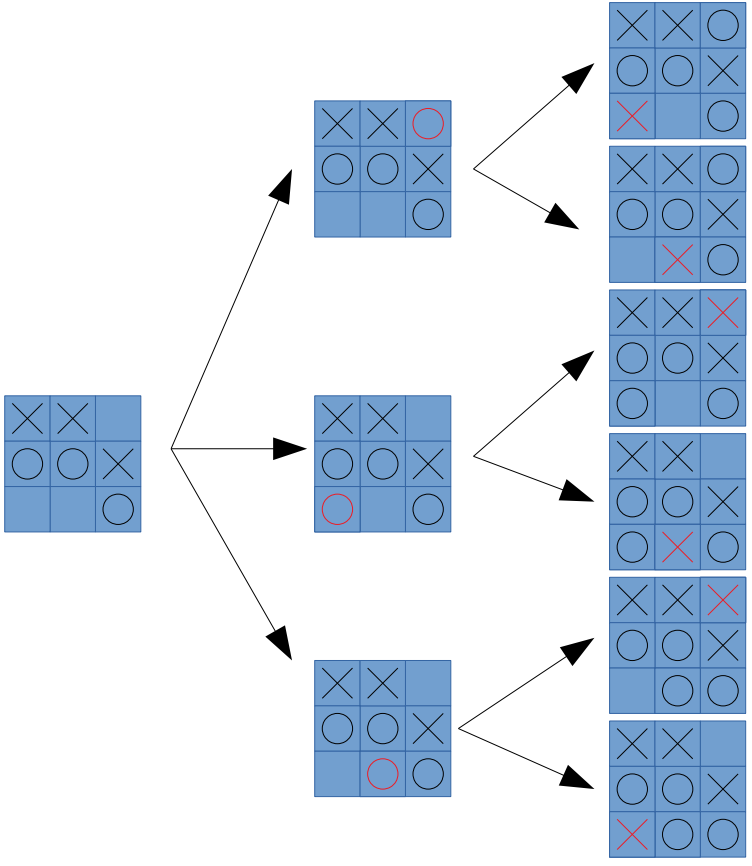


Pas de gain ? Quelle solution choisir ?

Et si on regardait plus loin ?

Exemple au Tic-Tac-Toe (3)

- C’est à  de jouer



R = ?

R = ?

R = -100

R = ?

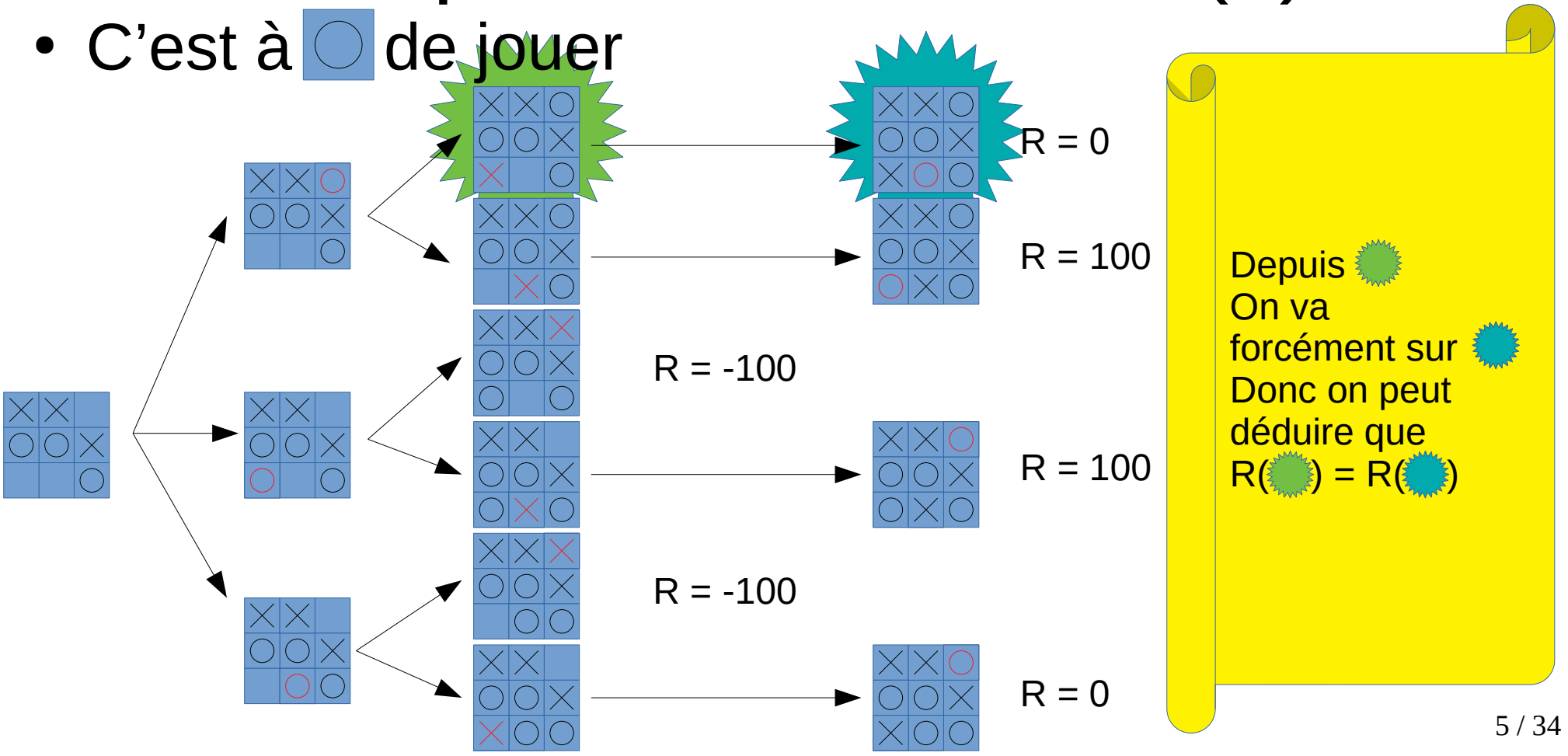
R = -100

R = ?

La récompense est évaluée par rapport à 

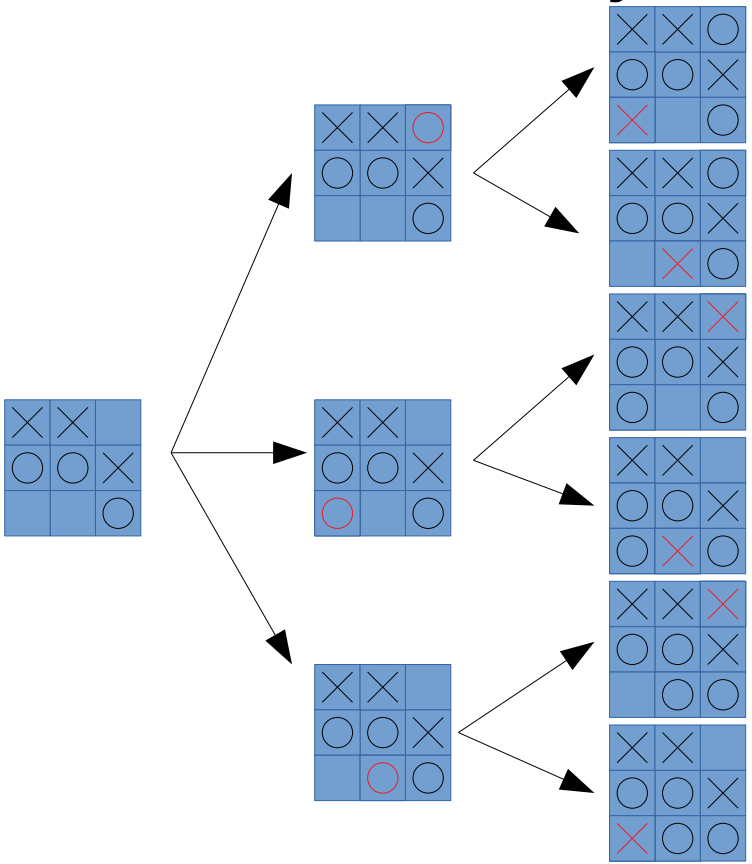
Exemple au Tic-Tac-Toe (4)

- C’est à ◯ de jouer



Exemple au Tic-Tac-Toe (5)

- C’est à ✕ de jouer



R = 0

R = 100

R = -100

R = 100

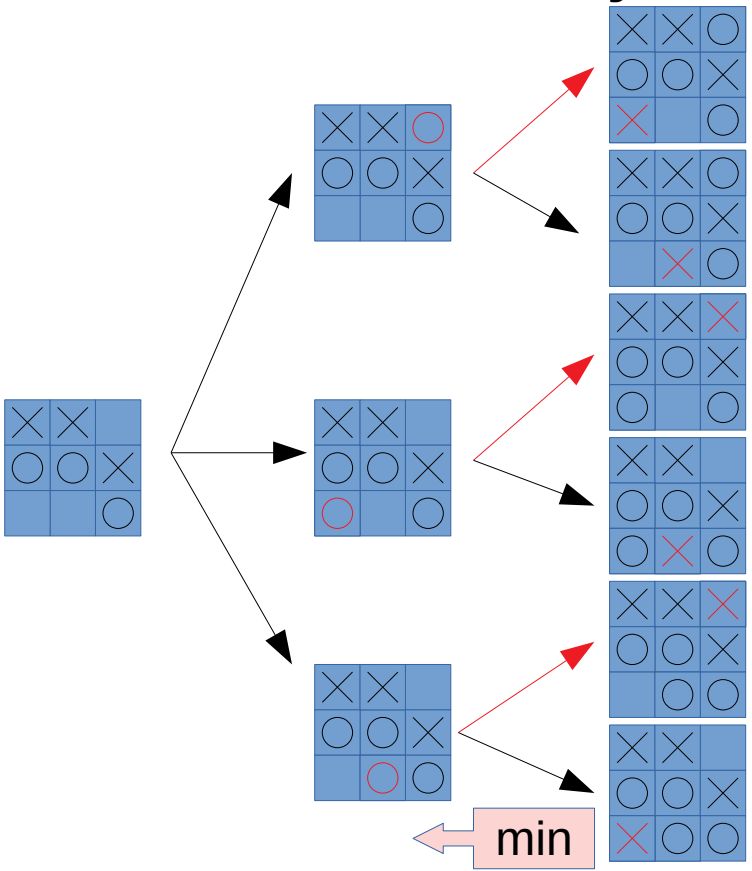
R = -100

R = 0

Dans chaque situation, ✕ va choisit le coup le meilleur pour lui, donc le moins bon pour ○ .

Exemple au Tic-Tac-Toe (5)

- C'est à ✕ de jouer



R = 0

R = 100

R = -100

R = 100

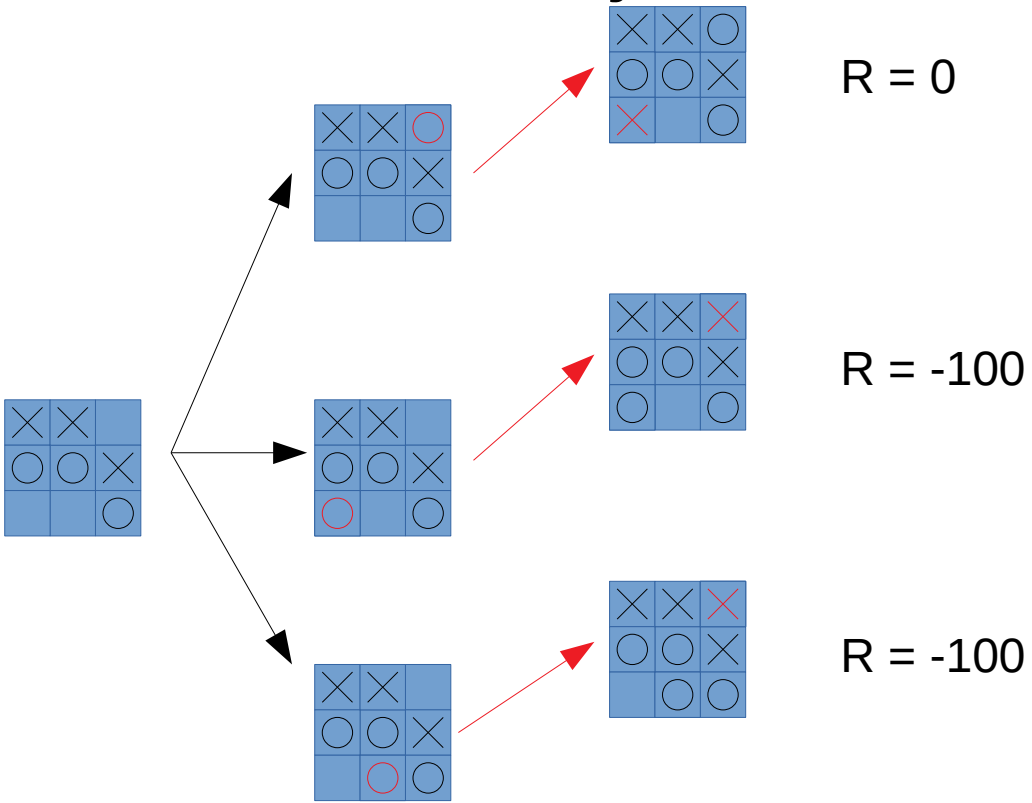
R = -100

R = 0

Dans chaque situation, ✕ va choisit le coup le meilleur pour lui, donc le moins bon pour ○ .

Exemple au Tic-Tac-Toe (6)

- C’est à  de jouer



R = 0

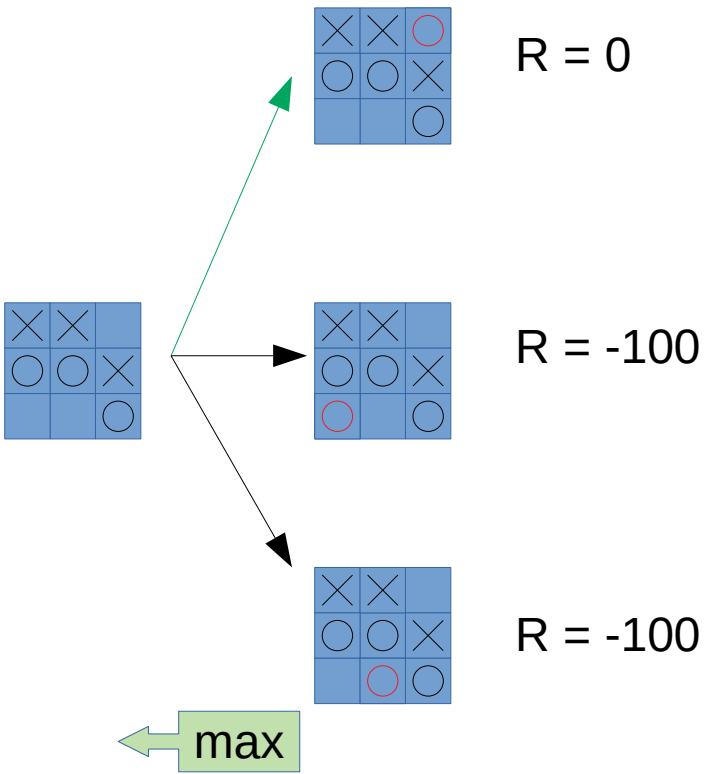
R = -100

R = -100

La récompense peut donc être « remontée »

Exemple au Tic-Tac-Toe (7)

- C’est à ◯ de jouer



Dans chaque situation, ◯ va choisit le coup le meilleur pour lui.

Généralisation

- Résumé des épisodes précédents
  - Au coup 8, c'était à X de jouer → min
  - Au coup 7, c'était à O de jouer → max
- Comment O doit choisir son coup précédent ?
  - Au coup 6, c'est à X de jouer → min
  - Au coup 5, c'est à O de jouer → max
- Et encore avant ?
  - Au coup 4, c'est à X de jouer → min
  - Au coup 3, c'est à O de jouer → max

  => On alterne des choix de min et de max → Algorithme min-max

Faisabilité ?

- Au coup 1, 9 positions possibles
- Au coup 2, 8 positions possibles
- Au coup 3, 7 positions possibles
- …
- Au coup 9, 1 position possible

  => 9 x 8 x 7 x … x 1 = 9 ! = 362 880 situations à calculer (en fait moins car victoires prématurées possibles) !

  => Faisable

Et sur un jeu comme le go ?

- 1er coup : 19x19 = 361 possibilités
- 2ème coup : 19x19-1 = 360 possibilités
- 3ème coup : 19x19 – 2 = 359 possibilités
- …

  => Si une partie se joue en 100 coups* :
  14383819305777268052878251034648751039590510672813907888736170455749271770994499123063703457167659441466656710337210671651206976646929682630275619993843121061414049239937812340873013242059839256653028079862407122536902783139840000000000000000000000000 situation à calculer !

* En supposant qu'il n'y a pas eu de prise

Et sur un jeu comme le go ?

- 1er coup : 19x19 = 361 possibilités

Impossible !

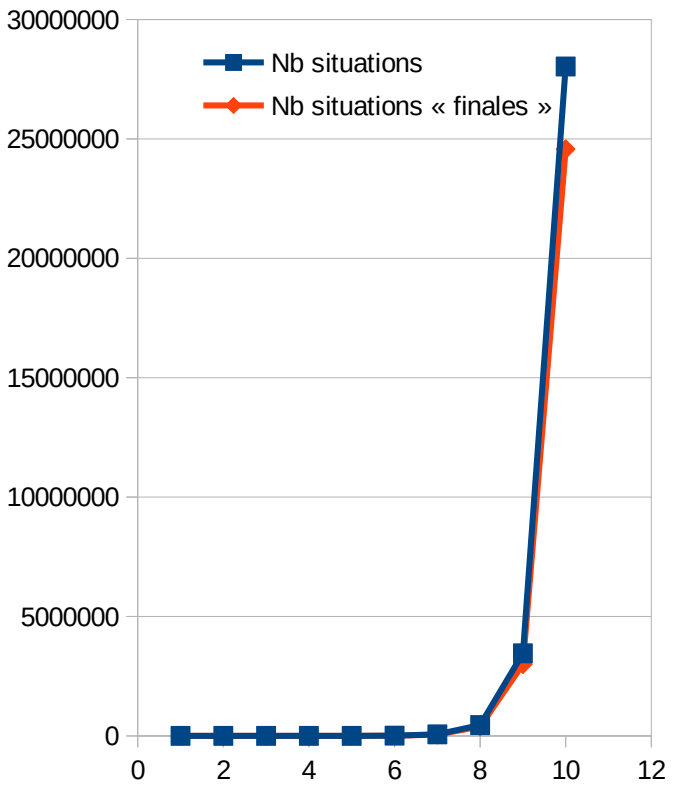
Ceci explique la nullité des programmes jouant au go avant l'arrivée d'alpha-go reposant sur une autre technique : le deep-learning

- 27831398400000000000000000000 situation à calculer !

* En supposant qu'il n'y a pas eu de prise

Et pour Othello ? Exemple au départ

Nb demi-coups	Nb situations	Nb situations « finales »
1	5	4
2	17	12
3	73	56
4	317	244
5	1 713	1 396
6	9 913	8 200
7	65 005	55 092
8	455 221	390 216
9	3 460 541	3 005 320
10	28 031 961	24 571 420



Évaluer une situation

- Idée : à partir de la position courante, calculer toutes les situation possibles sur *n* coups et donner une valeur de récompense à chacune
- Cette valeur de récompense doit être être déterminée en fonction du but du jeu, mais elle n'est qu'une approximation de la valeur de la situation : *fonction heuristique*.
- Le choix de la fonction heuristique est également déterminant dans le niveau de programme
- Un des meilleurs programmes d'échecs actuels (stockfish) utilise une fonction heuristique reposant sur plus de 30 critères

Algorithme min-max Informellement…

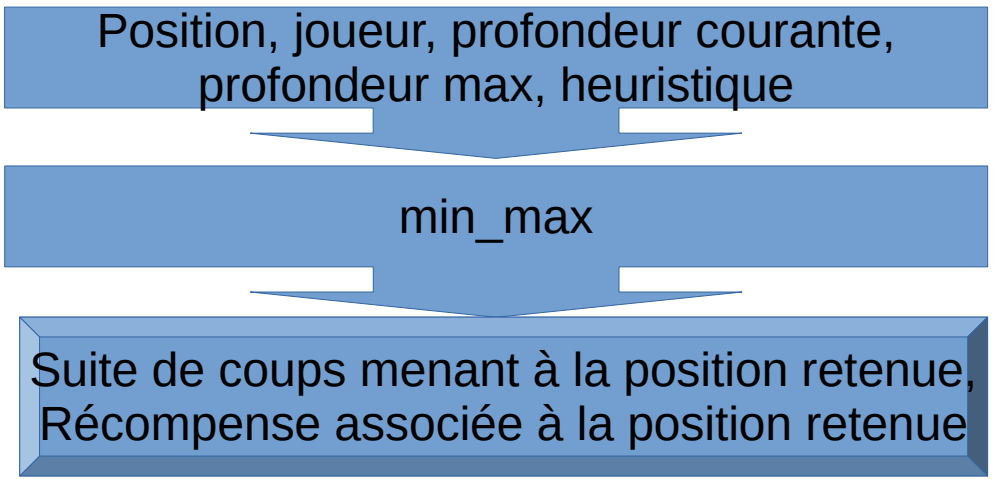
- Pour évaluer quel coup choisir, on utilise le principe min-max non pas à partir des récompenses sur les positions finales, mais à partir de la valeur de la fonction heuristique sur chacune des situations accessibles après n coups.
- On appelle *n* la *profondeur de raisonnement*. Plus *n* est grand, meilleure est la décision. Mais plus *n* est grand, plus le temps de calcul est long.

MinMax en Python (1)

- `jeu` est une variable contenant une donnée de type `Jeu`, où `Jeu` est une classe proposant :
  - Un constructeur de copie
  - Une méthode `tenter_coup` qui effectue un coup sur une copie du jeu sur lequel elle est appelée (et qui renvoie cette copie) ; cette méthode doit changer le joueur courant
  - Une méthode `liste_coups` qui renvoie la liste des coups possibles dans l'état courant
- `heuristique` est une fonction qui prend un jeu et un joueur en paramètre, et qui renvoie la valeur du jeu pour le joueur donné
- Au premier appel, la valeur à passer pour `prof_courante` est 1

N.B. : dans la version proposée, il manque notamment :
- le traitement des victoires et défaites prématurées
- la gestion des situations bloquées

MinMax en Python (1)



MinMax en Python (1)

MinMax en Python (2)

```
Def min_max(jeu, joueur, prof_courante, prof_max, heuristique):
```

- Si profondeur max atteinte :
  - Recompense = heuristique(jeu)
    Coups à réaliser = []
    Valeur de retour = (coups à réaliser, récompense)
- Sinon :
  - Déterminer la liste des coups
  - Si aucun coup possible :
    - Augmenter la profondeur
      Changer le joueur
      Valeur de retour = résultat de min_max sur la nouvelle situation
  - Sinon :
    - Déterminer le coup à retenir
      Définir la valeur de retour adéquate
- Renvoyer la valeur de retour retenue

MinMax en Python (2)

```
Def min_max(jeu, joueur, prof_courante, prof_max, heuristique):
    if prof_courante > prof_max:
        Recompense = heuristique(jeu)
        Coups à réaliser = []
        Valeur de retour =  (coups à réaliser, récompense)
    Sinon :
        Déterminer la liste des coups
        Si aucun coup possible :
            Augmenter la profondeur
            Changer le joueur
            Valeur de retour = résultat de min_max sur la nouvelle situation
        Sinon :
            Déterminer le coup à retenir
            Définir la valeur de retour adéquate
    Renvoyer la valeur de retour retenue
```

MinMax en Python (2)

```
Def min_max(jeu, joueur, prof_courante, prof_max, heuristique):
    if prof_courante > prof_max:

        retour = ([], heuristique(jeu, joueur))
```

Sinon :

- Déterminer la liste des coups
- Si aucun coup possible :
  - Augmenter la profondeur
    Changer le joueur
    Valeur de retour = résultat de min_max sur la nouvelle situation
- Sinon :
  - Déterminer le coup à retenir
    Définir la valeur de retour adéquate

Renvoyer la valeur de retour retenue

MinMax en Python (2)

```
Def min_max(jeu, joueur, prof_courante, prof_max, heuristique):
    if prof_courante > prof_max:

        retour = ([], heuristique(jeu, joueur))

    else:
```

Déterminer la liste des coups

Si aucun coup possible :

Augmenter la profondeur
Changer le joueur
Valeur de retour = résultat de min_max sur la nouvelle situation

Sinon :

Déterminer le coup à retenir
Définir la valeur de retour adéquate

Renvoyer la valeur de retour retenue

MinMax en Python (2)

```
Def min_max(jeu, joueur, prof_courante, prof_max, heuristique):
    if prof_courante > prof_max:

        retour = ([], heuristique(jeu, joueur))

    else:
        possibilites = jeu.liste_coups()
```

Si aucun coup possible :

Augmenter la profondeur
Changer le joueur
Valeur de retour = résultat de min_max sur la nouvelle situation

Sinon :

Déterminer le coup à retenir
Définir la valeur de retour adéquate

Renvoyer la valeur de retour retenue

MinMax en Python (2)

```
Def min_max(jeu, joueur, prof_courante, prof_max, heuristique):
    if prof_courante > prof_max:
        retour = ([], heuristique(jeu, joueur))
    else:
        possibilites = jeu.liste_coups()
        if len(possibilites) == 0:
```

Augmenter la profondeur
Changer le joueur
Valeur de retour = résultat de min_max sur la nouvelle situation

Sinon :

Déterminer le coup à retenir
Définir la valeur de retour adéquate

Renvoyer la valeur de retour retenue

MinMax en Python (2)

```
Def min_max(jeu, joueur, prof_courante, prof_max, heuristique):
    if prof_courante > prof_max:
        retour = ([], heuristique(jeu, joueur))
    else:
        possibilites = jeu.liste_coups()
        if len(possibilites) == 0:
            situation = Jeu(jeu); situation.changer_joueur()
            enchainement, recompense = min_max(situation, joueur, prof_courante+1,
                                               prof_max, heuristique)
            retour = (([None] + enchainement), recompense)
        Sinon :
            Déterminer le coup à retenir
            Définir la valeur de retour adéquate
    Renvoyer la valeur de retour retenue
```

MinMax en Python (2)

```
Def min_max(jeu, joueur, prof_courante, prof_max, heuristique):
    if prof_courante > prof_max:
        retour = ([], heuristique(jeu, joueur))
    else:
        possibilites = jeu.liste_coups()
        if len(possibilites) == 0:
            situation = Jeu(jeu); situation.changer_joueur()
            enchainement, recompense = min_max(situation, joueur, prof_courante+1,
                                               prof_max, heuristique)
            retour = (([None] + enchainement), recompense)
        else:
            Déterminer le coup à retenir
            Définir la valeur de retour adéquate
    Renvoyer la valeur de retour retenue
```

MinMax en Python (2)

```
Def min_max(jeu, joueur, prof_courante, prof_max, heuristique):
    if prof_courante > prof_max:

        retour = ([], heuristique(jeu, joueur))

    else:
        possibilites = jeu.liste_coups()
        if len(possibilites) == 0:
            situation = Jeu(jeu); situation.changer_joueur()
            enchainement, recompense = min_max(situation, joueur, prof_courante+1,
                                               prof_max, heuristique)
            retour = (([None] + enchainement), recompense)
        else:
```

Partie détaillée sur le transparent suivant

```
    Renvoyer la valeur de retour retenue
```

MinMax en Python (2)

```
Def min_max(jeu, joueur, prof_courante, prof_max, heuristique):
    if prof_courante > prof_max:
        retour = ([], heuristique(jeu, joueur))
    else:
        possibilites = jeu.liste_coups()
        if len(possibilites) == 0:
            situation = Jeu(jeu); situation.changer_joueur()
            enchainement, recompense = min_max(situation, joueur, prof_courante+1,
                                               prof_max, heuristique)
            retour = (([None] + enchainement), recompense)
        else:
            Partie détaillée sur le transparent suivant
    return retour
```

MinMax en Python (3)

Déterminer le coup à retenir
Définir la valeur de retour adéquate

MinMax en Python (3)

Pour chaque coup possible, lui associer la situation qu'il permet d'atteindre

Si c'est le tour du joueur courant :

    Retenir l'enchaînement correspondant à la meilleure récompense

Sinon:

    Retenir l'enchaînement correspondant à la pire récompense

MinMax en Python (3)

```
situations_possibles = map(lambda coup: (coup, jeu.tenter_coup(coup), possibilites)
```

Si c’est le tour du joueur courant :

Retenir l’enchaînement correspondant à la meilleure récompense

Sinon:

Retenir l’enchaînement correspondant à la pire récompense

MinMax en Python (3)

```
situations_possibles = map(lambda coup: (coup, jeu.tenter_coup(coup), possibilites)
if prof_courante % 2 == 1:
```

Retenir l’enchaînement correspondant à la meilleure récompense

Sinon:

Retenir l’enchaînement correspondant à la pire récompense

MinMax en Python (3)

```
situations_possibles = map(lambda coup: (coup, jeu.tenter_coup(coup), possibilites)
if prof_courante % 2 == 1:
```

- Initialiser les valeurs à retenir
- Pour chaque couple (coup, situation):
  - Déterminer (suite_de_coups, récompense)
  - Si la récompense est meilleure
    - Mettre à jour suite_de_coups et récompense
  - Mettre à jour la valeur de retour
- Sinon:
  - Retenir l'enchaînement correspondant à la pire récompense

MinMax en Python (3)

```
situations_possibles = map(lambda coup: (coup, jeu.tenter_coup(coup), possibilites)
if prof_courante % 2 == 1:
    Initialiser les valeurs à retenir
    Pour chaque couple (coup, situation):
        Déterminer (suite_de_coups, récompense)
        Si la récompense est meilleure
            Mettre à jour suite_de_coups et récompense
    Mettre à jour la valeur de retour
Sinon:
    Initialiser les valeurs à retenir
    Pour chaque couple (coup, situation):
        Déterminer (suite_de_coups, récompense)
        Si la récompense est pire
            Mettre à jour suite_de_coups et récompense
    Mettre à jour la valeur de retour
```

MinMax en Python (3)

```
situations_possibles = map(lambda coup: (coup, jeu.tenter_coup(coup), possibilites)
if prof_courante % 2 == 1:
    meilleure_recompense = None
    Pour chaque couple (coup, situation):
        Déterminer (suite_de_coups, récompense)
        Si la récompense est meilleure
            Mettre à jour suite_de_coups et récompense
    Mettre à jour la valeur de retour
Sinon:
    Initialiser les valeurs à retenir
    Pour chaque couple (coup, situation):
        Déterminer (suite_de_coups, récompense)
        Si la récompense est pire
            Mettre à jour suite_de_coups et récompense
    Mettre à jour la valeur de retour
```

MinMax en Python (3)

```
situations_possibles = map(lambda coup: (coup, jeu.tenter_coup(coup), possibilites)
if prof_courante % 2 == 1:
    meilleure_recompense = None
    for (coup, situation) in situations_possibles:
        Déterminer (suite_de_coups, récompense)
        Si la récompense est meilleure
            Mettre à jour suite_de_coups et récompense
    Mettre à jour la valeur de retour
Sinon:
    Initialiser les valeurs à retenir
    Pour chaque couple (coup, situation):
        Déterminer (suite_de_coups, récompense)
        Si la récompense est pire
            Mettre à jour suite_de_coups et récompense
    Mettre à jour la valeur de retour
```

MinMax en Python (3)

```
situations_possibles = map(lambda coup: (coup, jeu.tenter_coup(coup)), possibilites)
if prof_courante % 2 == 1:
    meilleure_recompense = None
    for (coup, situation) in situations_possibles:
        liste_coups, recompense = min_max(situation, joueur, prof_courante+1, prof_max, heuristique)
        Si la récompense est meilleure
            Mettre à jour suite_de_coups et récompense
    Mettre à jour la valeur de retour
Sinon:
    Initialiser les valeurs à retenir
    Pour chaque couple (coup, situation):
        Déterminer (suite_de_coups, récompense)
        Si la récompense est pire
            Mettre à jour suite_de_coups et récompense
    Mettre à jour la valeur de retour
```

MinMax en Python (3)

```
situations_possibles = map(lambda coup: (coup, jeu.tenter_coup(coup)), possibilites)
if prof_courante % 2 == 1:
    meilleure_recompense = None
    for (coup, situation) in situations_possibles:
        liste_coups, recompense = min_max(situation, joueur, prof_courante+1, prof_max, heuristique)
        if meilleure_recompense is None or recompense > meilleure_recompense:
            Mettre à jour suite_de_coups et récompense
    Mettre à jour la valeur de retour
Sinon:
    Initialiser les valeurs à retenir
    Pour chaque couple (coup, situation):
        Déterminer (suite_de_coups, récompense)
        Si la récompense est pire
            Mettre à jour suite_de_coups et récompense
    Mettre à jour la valeur de retour
```

MinMax en Python (3)

```
situations_possibles = map(lambda coup: (coup, jeu.tenter_coup(coup)), possibilites)
if prof_courante % 2 == 1:
    meilleure_recompense = None
    for (coup, situation) in situations_possibles:
        liste_coups, recompense = min_max(situation, joueur, prof_courante+1, prof_max, heuristique)
        if meilleure_recompense is None or recompense > meilleure_recompense:
            meilleure_recompense = recompense
            enchainement_retenu = [coup] + liste_coups
    Mettre à jour la valeur de retour
Sinon:
    Initialiser les valeurs à retenir
    Pour chaque couple (coup, situation):
        Déterminer (suite_de_coups, récompense)
        Si la récompense est pire
            Mettre à jour suite_de_coups et récompense
    Mettre à jour la valeur de retour
```

MinMax en Python (3)

```
situations_possibles = map(lambda coup: (coup, jeu.tenter_coup(coup)), possibilites)
if prof_courante % 2 == 1:
    meilleure_recompense = None
    for (coup, situation) in situations_possibles:
        liste_coups, recompense = min_max(situation, joueur, prof_courante+1, prof_max, heuristique)
        if meilleure_recompense is None or recompense > meilleure_recompense:
            meilleure_recompense = recompense
            enchainement_retenu = [coup] + liste_coups
    retour = enchainement_retenu, meilleure_recompense
```

Sinon:

- Initialiser les valeurs à retenir
- Pour chaque couple (coup, situation):
  - Déterminer (suite_de_coups, récompense)
  - Si la récompense est pire
    - Mettre à jour suite_de_coups et récompense
- Mettre à jour la valeur de retour

MinMax en Python (3)

```
situations_possibles = map(lambda coup: (coup, jeu.tenter_coup(coup)), possibilites)
if prof_courante % 2 == 1:
    meilleure_recompense = None
    for (coup, situation) in situations_possibles:
        liste_coups, recompense = min_max(situation, joueur, prof_courante+1, prof_max, heuristique)
        if meilleure_recompense is None or recompense > meilleure_recompense:
            meilleure_recompense = recompense
            enchainement_retenu = [coup] + liste_coups
    retour = enchainement_retenu, meilleure_recompense
else:
```

- Initialiser les valeurs à retenir
- Pour chaque couple (coup, situation):
  - Déterminer (suite_de_coups, récompense)
  - Si la récompense est pire
    - Mettre à jour suite_de_coups et récompense
  - Mettre à jour la valeur de retour

MinMax en Python (3)

```
situations_possibles = map(lambda coup: (coup, jeu.tenter_coup(coup)), possibilites)
if prof_courante % 2 == 1:
    meilleure_recompense = None
    for (coup, situation) in situations_possibles:
        liste_coups, recompense = min_max(situation, joueur, prof_courante+1, prof_max, heuristique)
        if meilleure_recompense is None or recompense > meilleure_recompense:
            meilleure_recompense = recompense
            enchainement_retenu = [coup] + liste_coups
    retour = enchainement_retenu, meilleure_recompense
else:
    pire_recompense = None
```

Pour chaque couple (coup, situation):

- Déterminer (suite_de_coups, récompense)
- Si la récompense est pire
  - Mettre à jour suite_de_coups et récompense
- Mettre à jour la valeur de retour

MinMax en Python (3)

```
situations_possibles = map(lambda coup: (coup, jeu.tenter_coup(coup)), possibilites)
if prof_courante % 2 == 1:
    meilleure_recompense = None
    for (coup, situation) in situations_possibles:
        liste_coups, recompense = min_max(situation, joueur, prof_courante+1, prof_max, heuristique)
        if meilleure_recompense is None or recompense > meilleure_recompense:
            meilleure_recompense = recompense
            enchainement_retenu = [coup] + liste_coups
    retour = enchainement_retenu, meilleure_recompense
else:
    pire_recompense = None
    for (coup, situation) in situations_possibles:
        Déterminer (suite_de_coups, récompense)
        Si la récompense est pire
            Mettre à jour suite_de_coups et récompense
    Mettre à jour la valeur de retour
```

MinMax en Python (3)

```
situations_possibles = map(lambda coup: (coup, jeu.tenter_coup(coup)), possibilites)
if prof_courante % 2 == 1:
    meilleure_recompense = None
    for (coup, situation) in situations_possibles:
        liste_coups, recompense = min_max(situation, joueur, prof_courante+1, prof_max, heuristique)
        if meilleure_recompense is None or recompense > meilleure_recompense:
            meilleure_recompense = recompense
            enchainement_retenu = [coup] + liste_coups
    retour = enchainement_retenu, meilleure_recompense
else:
    pire_recompense = None
    for (coup, situation) in situations_possibles:
        liste_coups, recompense = min_max(situation, joueur, prof_courante+1, prof_max, heuristique)
        Si la récompense est pire
            Mettre à jour suite_de_coups et récompense
    Mettre à jour la valeur de retour
```

MinMax en Python (3)

```
situations_possibles = map(lambda coup: (coup, jeu.tenter_coup(coup)), possibilites)
if prof_courante % 2 == 1:
    meilleure_recompense = None
    for (coup, situation) in situations_possibles:
        liste_coups, recompense = min_max(situation, joueur, prof_courante+1, prof_max, heuristique)
        if meilleure_recompense is None or recompense > meilleure_recompense:
            meilleure_recompense = recompense
            enchainement_retenu = [coup] + liste_coups
    retour = enchainement_retenu, meilleure_recompense
else:
    pire_recompense = None
    for (coup, situation) in situations_possibles:
        liste_coups, recompense = min_max(situation, joueur, prof_courante+1, prof_max, heuristique)
        if pire_recompense is None or recompense < pire_recompense:
            Mettre à jour suite_de_coups et récompense
    Mettre à jour la valeur de retour
```

MinMax en Python (3)

```
situations_possibles = map(lambda coup: (coup, jeu.tenter_coup(coup)), possibilites)
if prof_courante % 2 == 1:
    meilleure_recompense = None
    for (coup, situation) in situations_possibles:
        liste_coups, recompense = min_max(situation, joueur, prof_courante+1, prof_max, heuristique)
        if meilleure_recompense is None or recompense > meilleure_recompense:
            meilleure_recompense = recompense
            enchainement_retenu = [coup] + liste_coups
    retour = enchainement_retenu, meilleure_recompense
else:
    pire_recompense = None
    for (coup, situation) in situations_possibles:
        liste_coups, recompense = min_max(situation, joueur, prof_courante+1, prof_max, heuristique)
        if pire_recompense is None or recompense < pire_recompense:
            pire_recompense = recompense
            enchainement_retenu = [coup] + liste_coups
    Mettre à jour la valeur de retour
```

MinMax en Python (3)

```
situations_possibles = map(lambda coup: (coup, jeu.tenter_coup(coup)), possibilites)
if prof_courante % 2 == 1:
    meilleure_recompense = None
    for (coup, situation) in situations_possibles:
        liste_coups, recompense = min_max(situation, joueur, prof_courante+1, prof_max, heuristique)
        if meilleure_recompense is None or recompense > meilleure_recompense:
            meilleure_recompense = recompense
            enchainement_retenu = [coup] + liste_coups
    retour = enchainement_retenu, meilleure_recompense
else:
    pire_recompense = None
    for (coup, situation) in situations_possibles:
        liste_coups, recompense = min_max(situation, joueur, prof_courante+1, prof_max, heuristique)
        if pire_recompense is None or recompense < pire_recompense:
            pire_recompense = recompense
            enchainement_retenu = [coup] + liste_coups
    retour = enchainement_retenu, pire_recompense
```

Déterminer une fonction heuristique Exemple pour le jeu Othello (1)

- Résumé des règles :
  - Les joueurs disposent de pions avec une face à leur couleur, une face à la couleur adverse
  - À chaque coup, un joueur pose un pion (avec la face à sa couleur dessus) sur une case libre et retourne sur sa couleur tous les pions adverses encadrés par le pion qu'il vient de jouer et au autre pion à lui
  - Le gagnant est celui qui, à la fin, à le plus de pions avec la face à sa couleur visible

Déterminer une fonction heuristique Exemple pour le jeu Othello (2)

- Fonction heuristique de base :
  - Le gain est directement lié à la différence entre les nombres de faces de chaque couleur visibles

    => f(position, joueur) = nb_faces_joueur – nb_faces_adversaire
- Conséquence :
  - Un comportement simpliste : pas beaucoup d’anticipation dans un jeu où les retournements de situation peuvent être brutaux

    => niveau d’un joueur débutant / d’un enfant

Déterminer une fonction heuristique Exemple pour le jeu Othello (3)

- Fonction heuristique améliorée :
  - Une pièce dans un coin est imprenable
  - Une pièce d'un bord adjacente à une pièce imprenable est imprenable

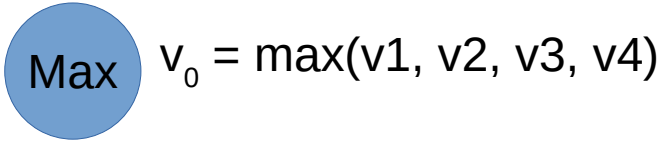
    => Surévaluer le fait d'avoir une pièce vérifiant ces conditions (10 au lieu de 1 par exemple) ; éventuellement, faire décroître la surévaluation au fur et à mesure que la partie avance
- Conséquence

  Amélioration nette du programme : avec une analyse sur une dizaine de coups, programme imbattable par un joueur moyen

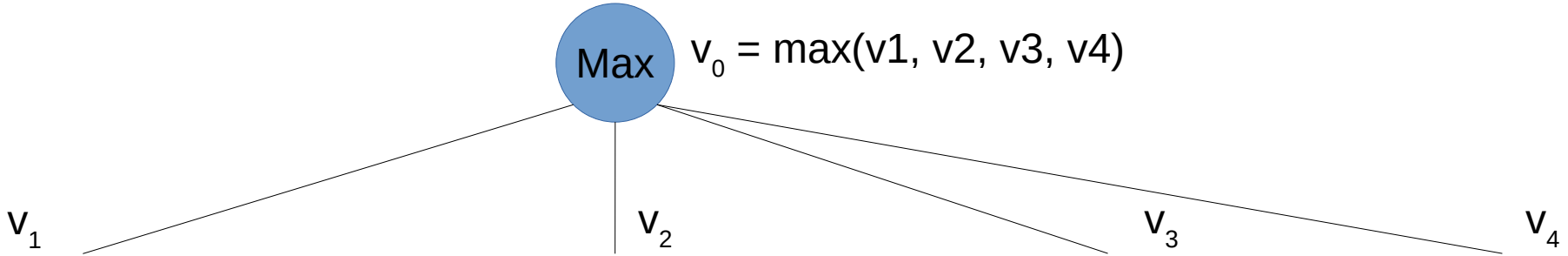
Applicabilité en Python ?

Nb demi-coup	Tps calcul Python	Tps calcul Java
1	0,0025	0,03
2	0,01	0,033
3	0,046	0,045
4	0,26	0,076
5	1,17	0,254
6	6,1	0,3
7	42	0,84
8	276	3,4
9	2 079	24
10		180

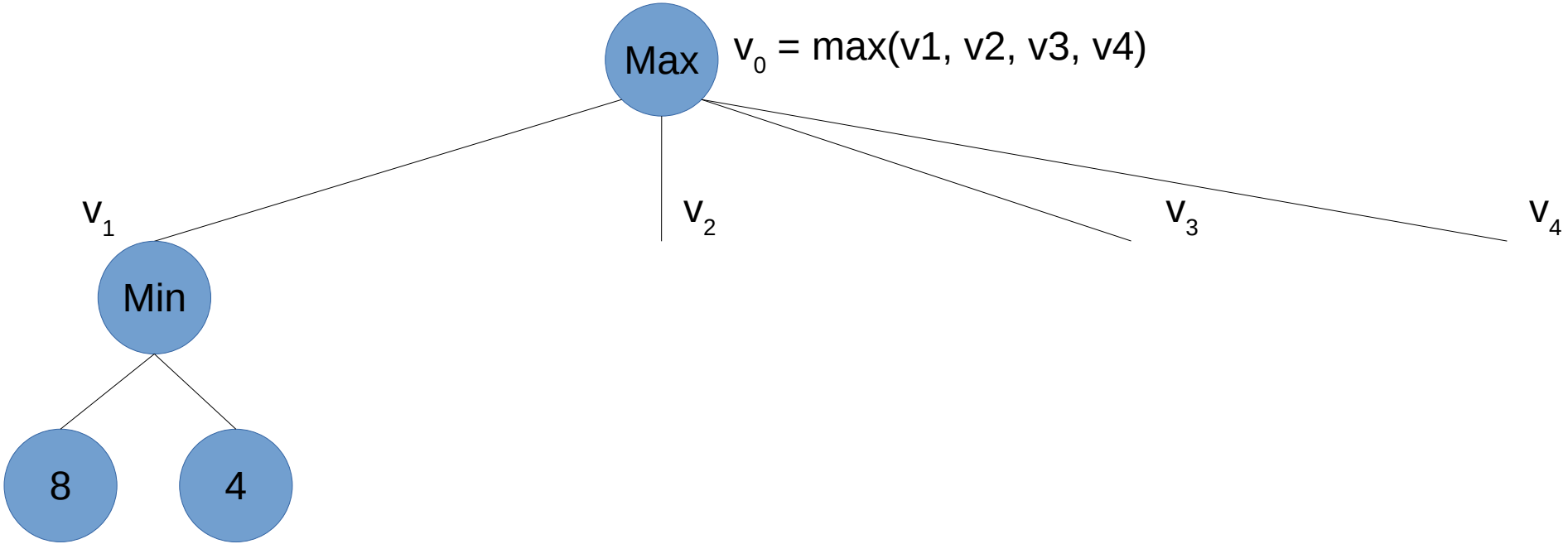
Améliorer l’efficacité : l’élagage α



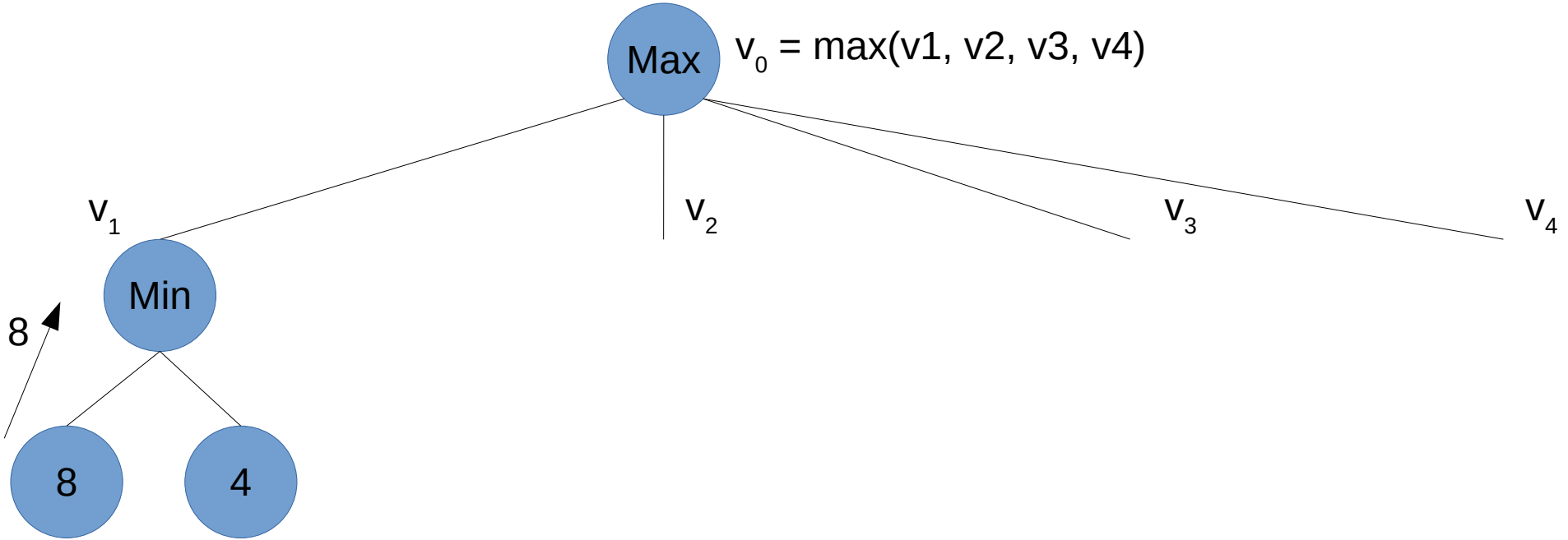
Améliorer l’efficacité : l’élagage α



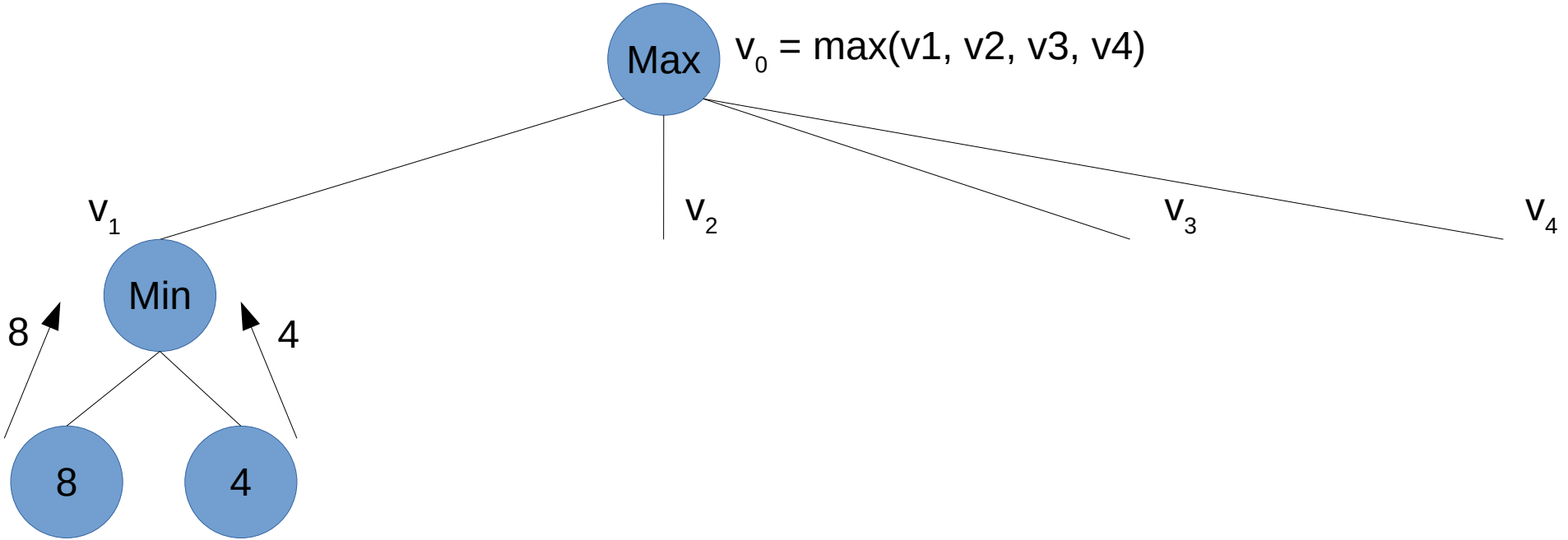
Améliorer l’efficacité : l’élagage α



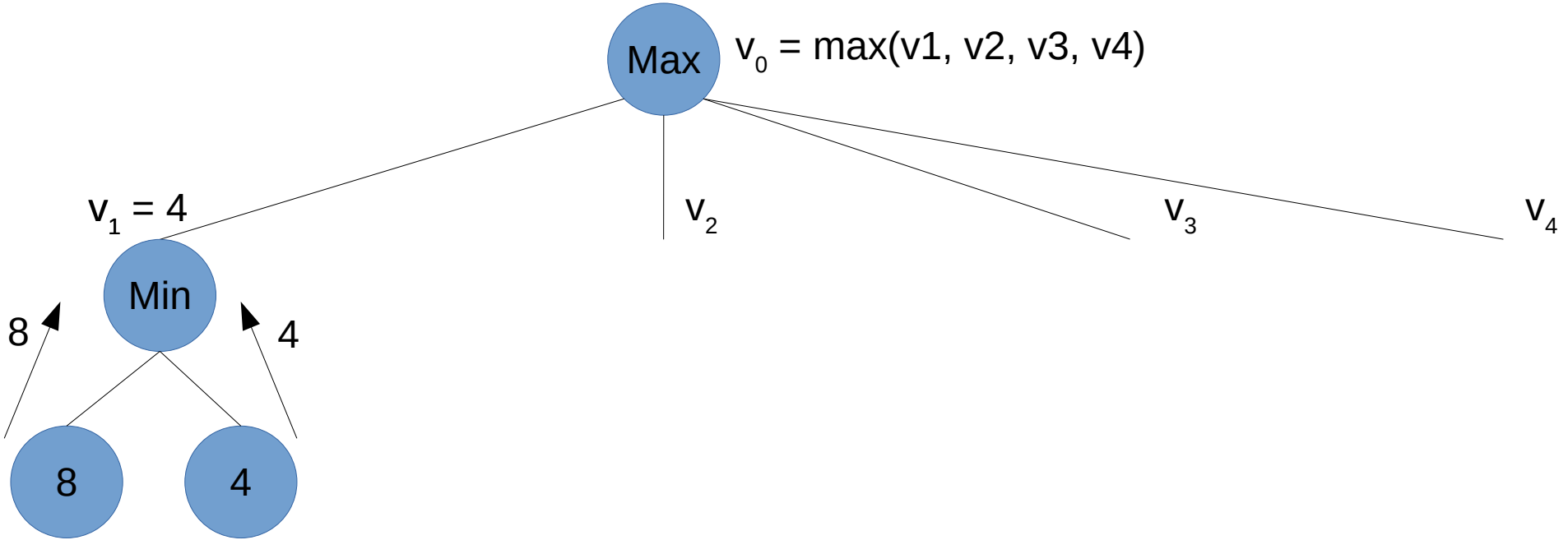
Améliorer l’efficacité : l’élagage α



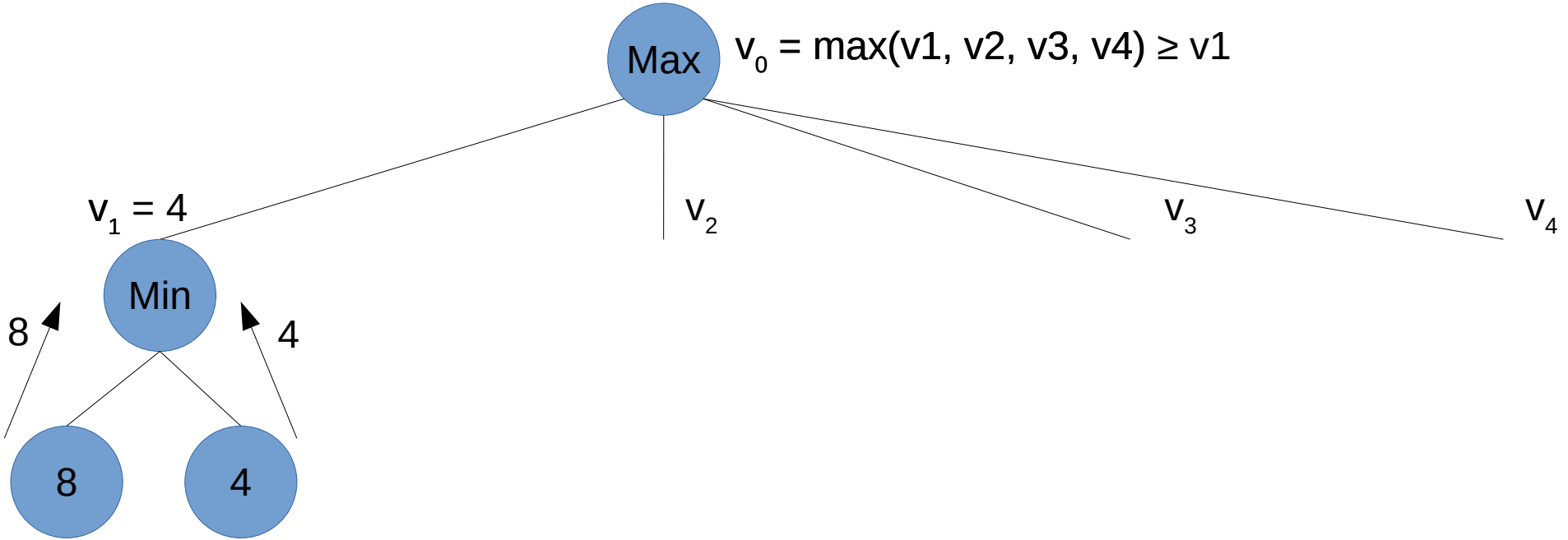
Améliorer l’efficacité : l’élagage α



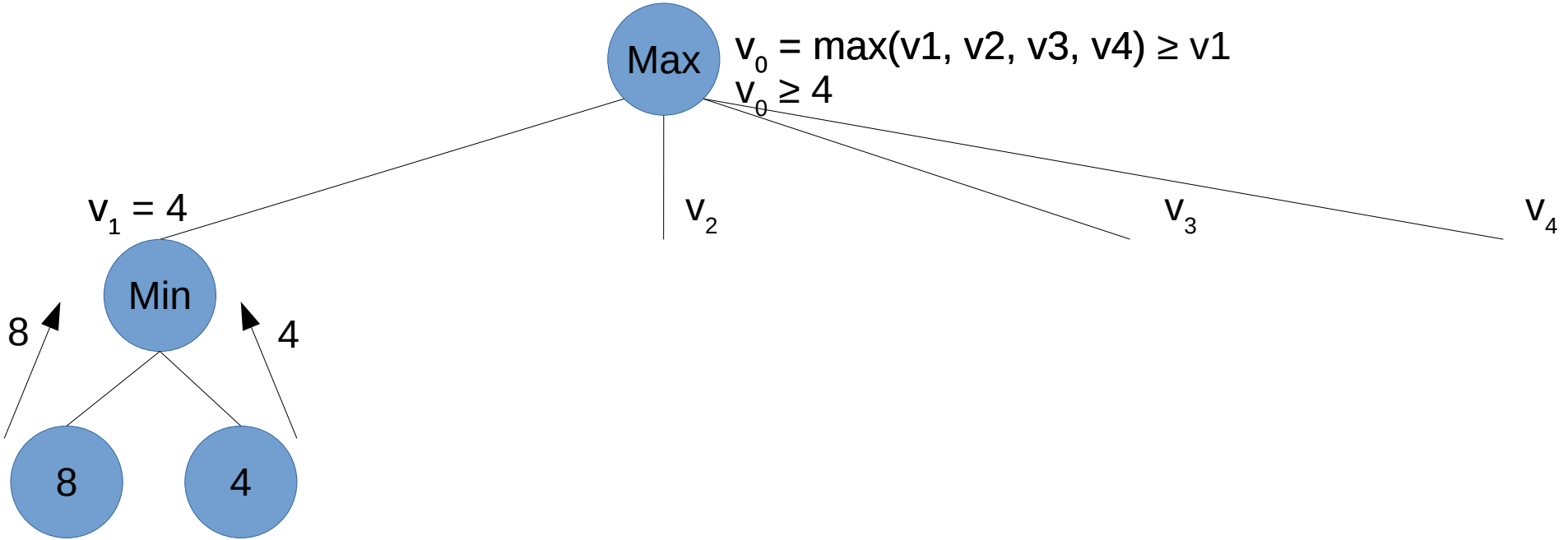
Améliorer l’efficacité : l’élagage α



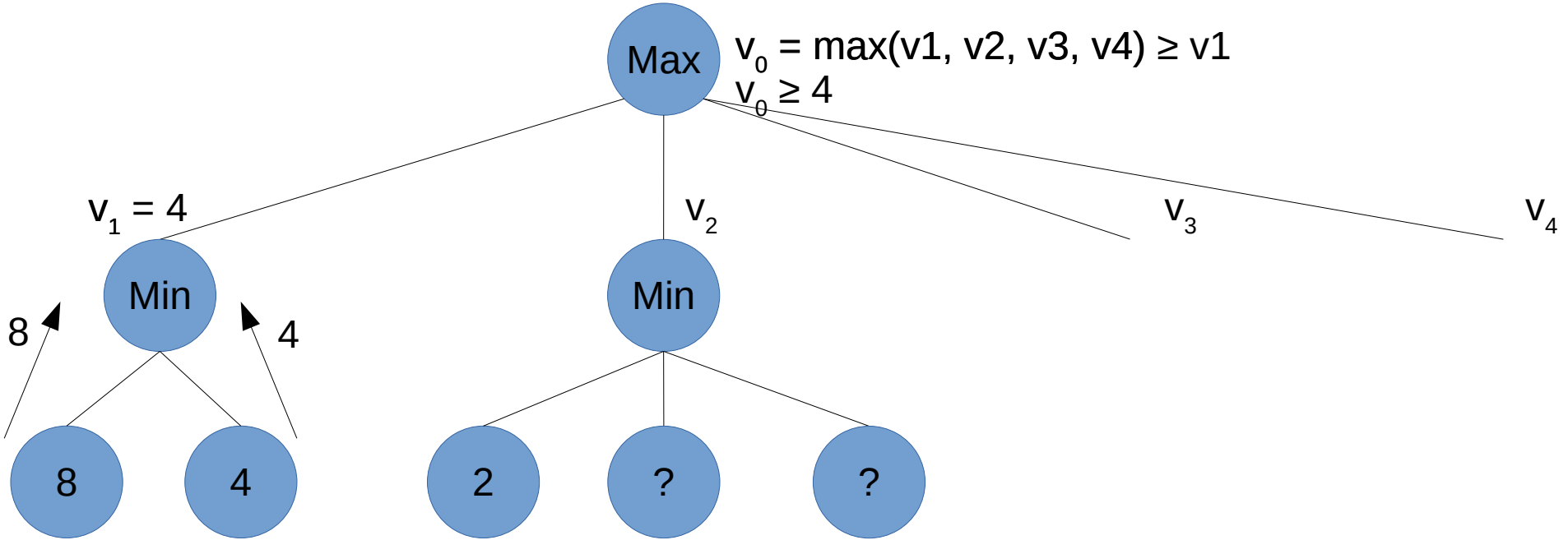
Améliorer l’efficacité : l’élagage α



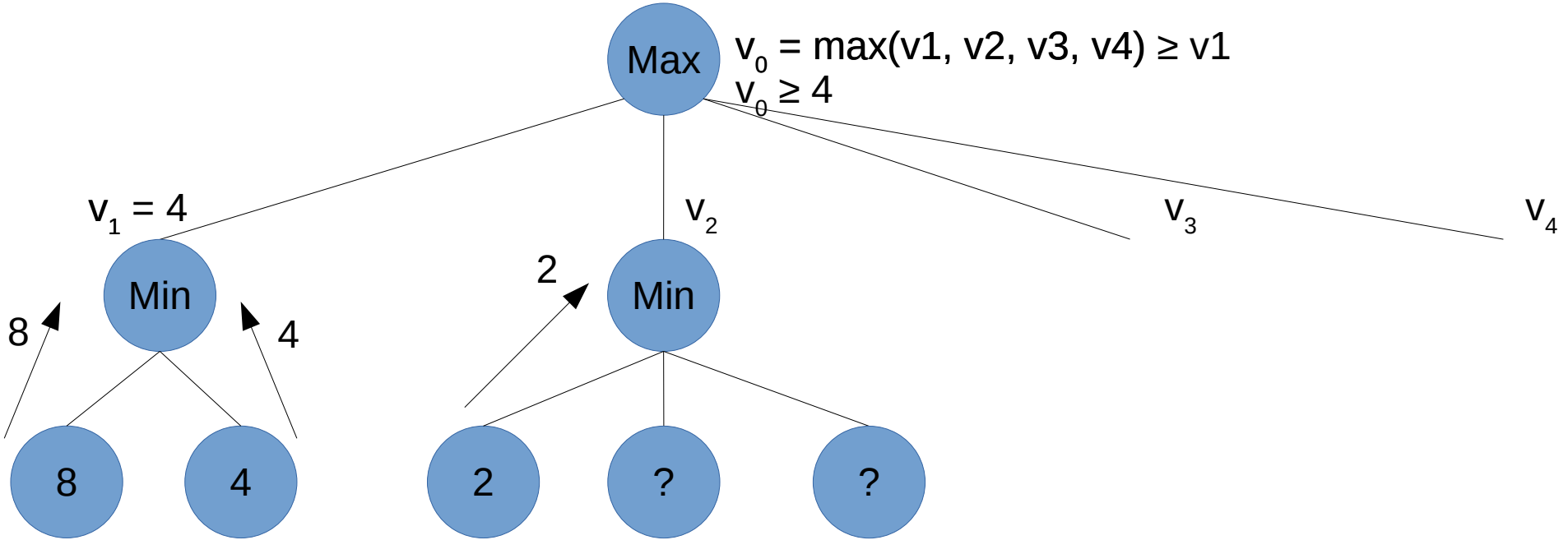
Améliorer l’efficacité : l’élagage α



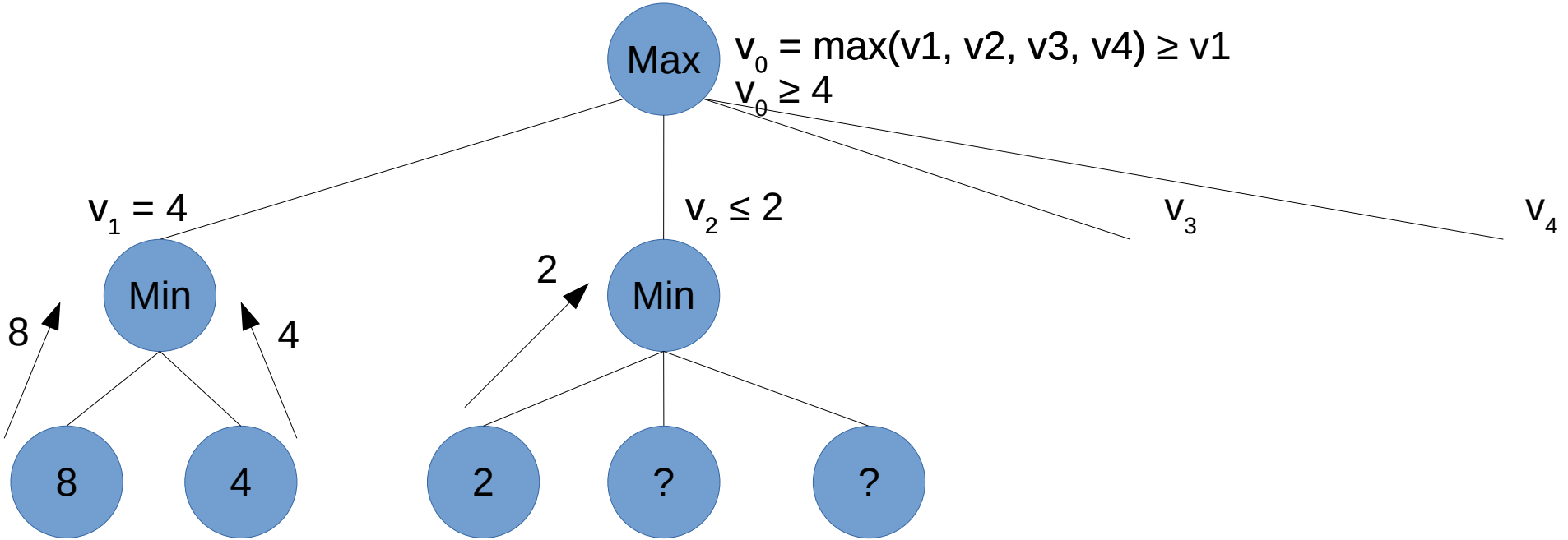
Améliorer l’efficacité : l’élagage α



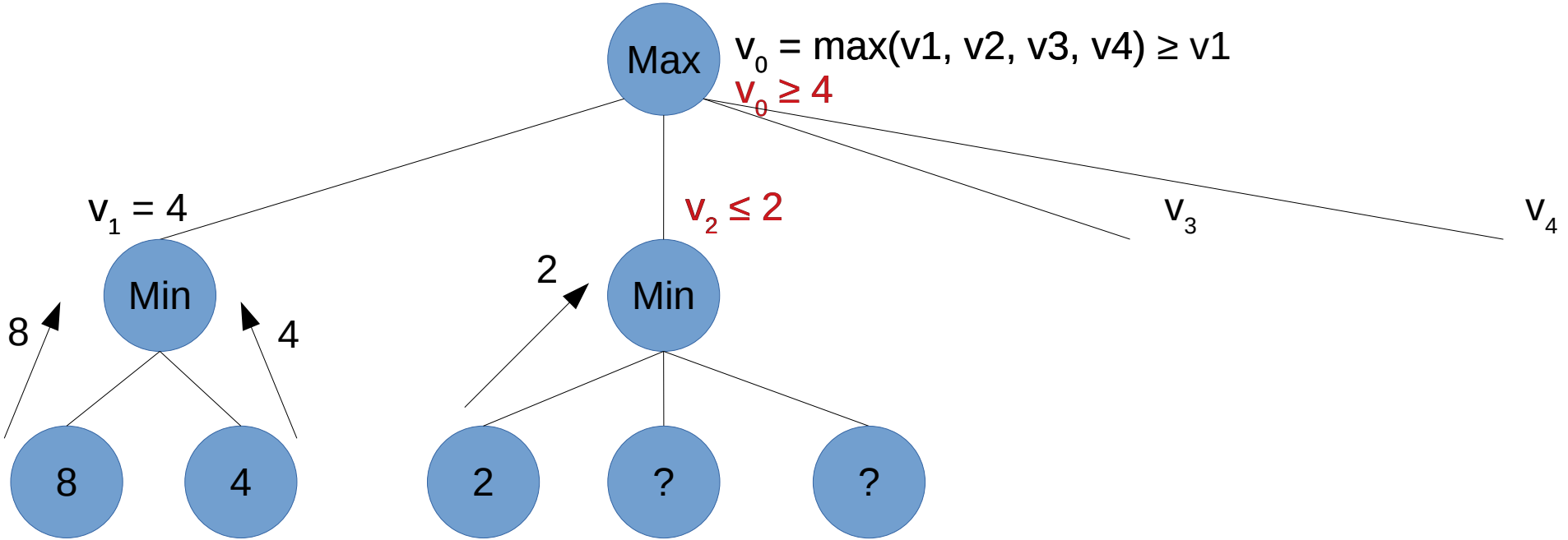
Améliorer l’efficacité : l’élagage α



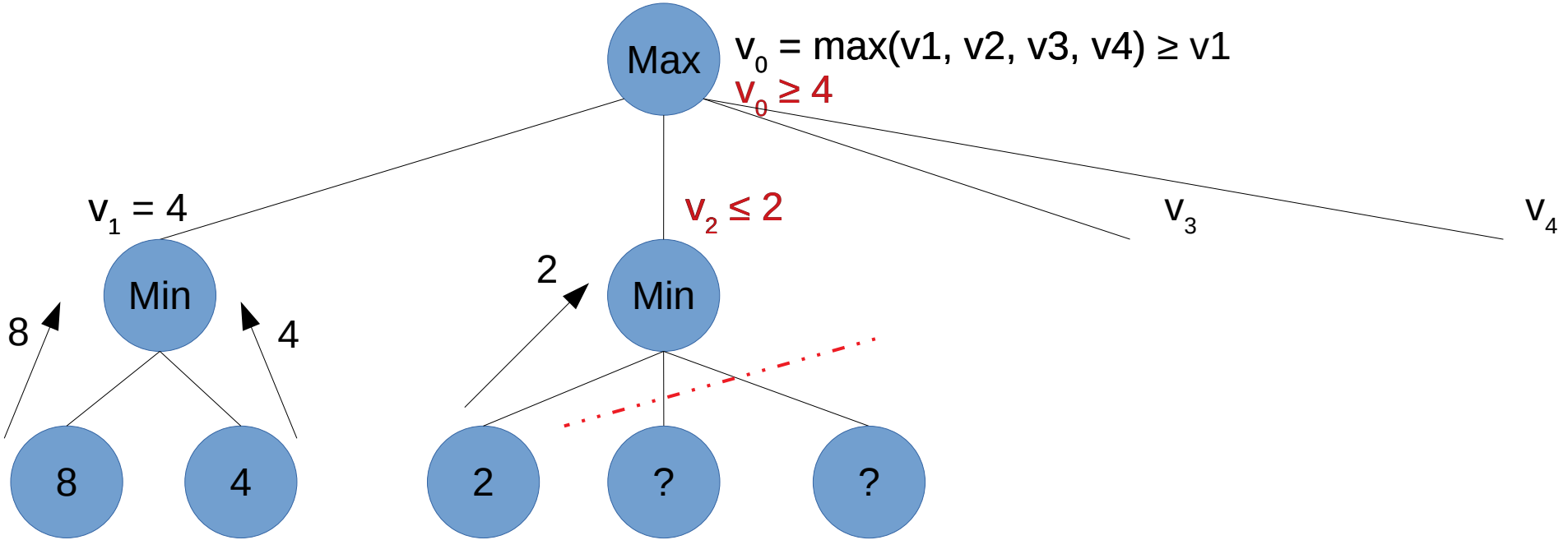
Améliorer l’efficacité : l’élagage α



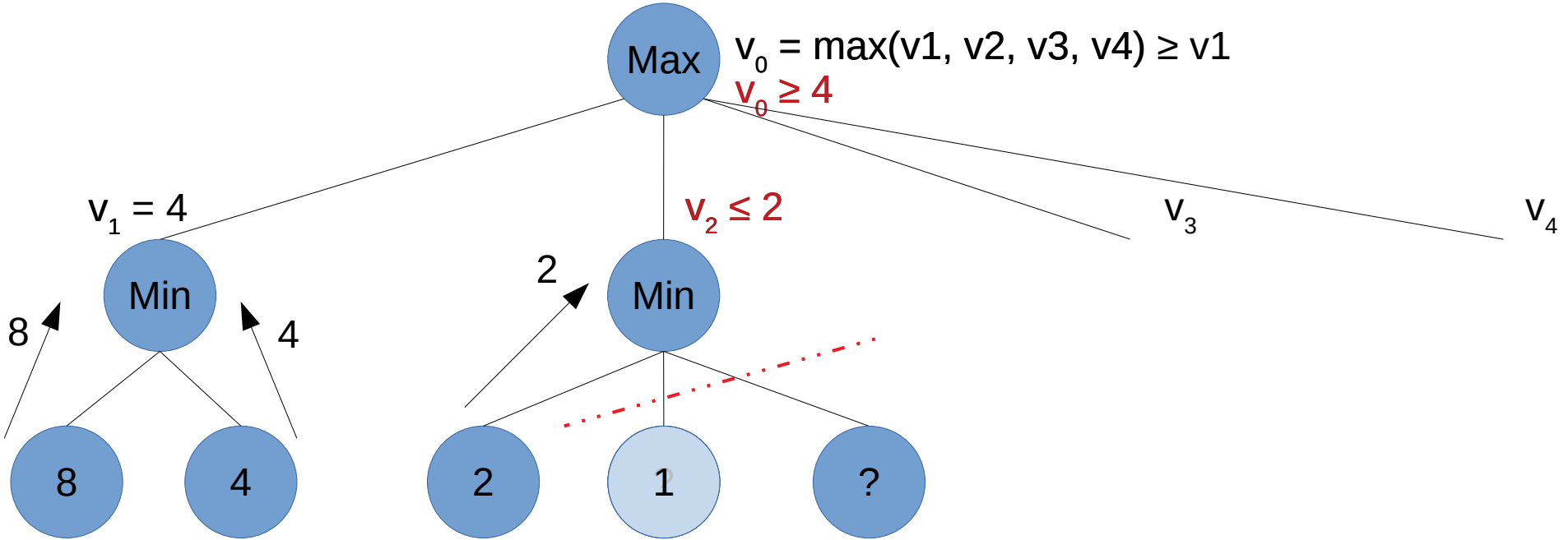
Améliorer l’efficacité : l’élagage α



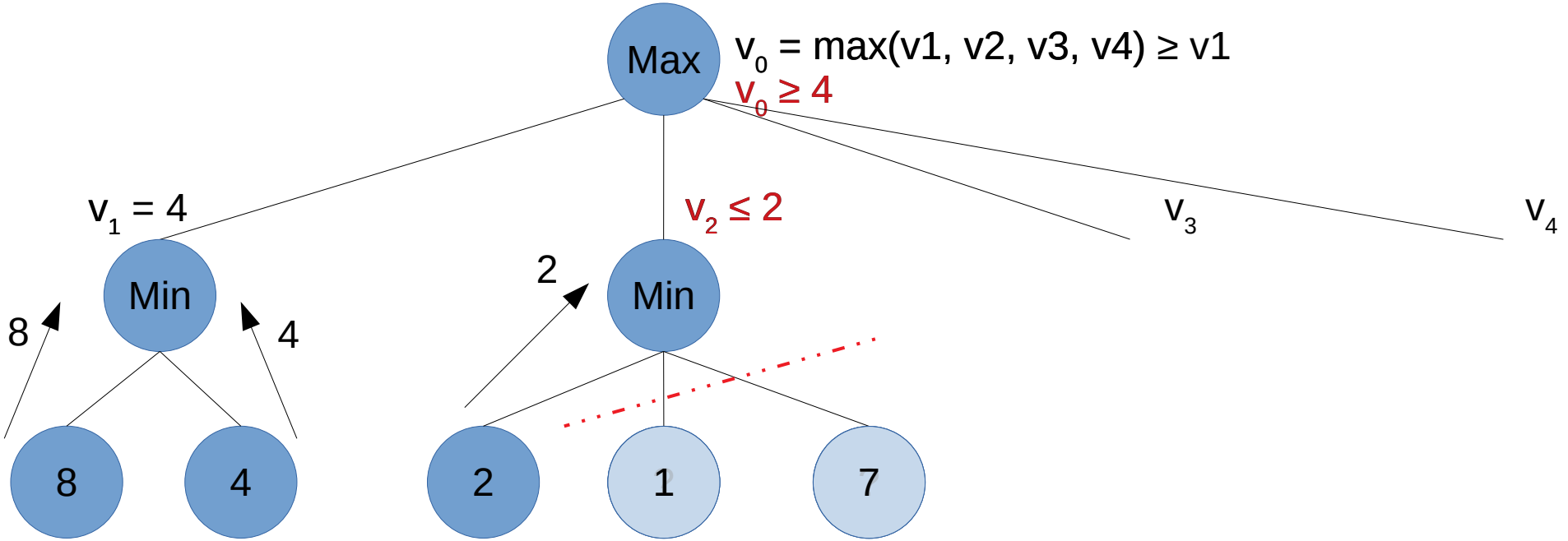
Améliorer l'efficacité : l'élagage α



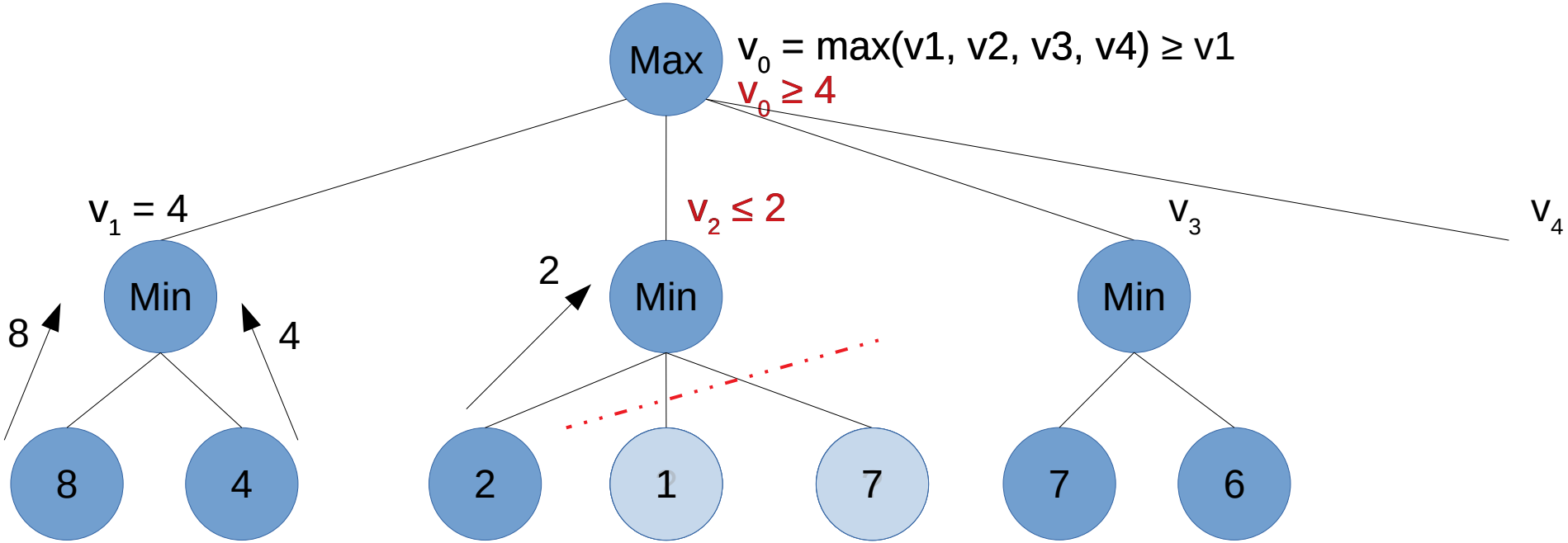
Améliorer l'efficacité : l'élagage α



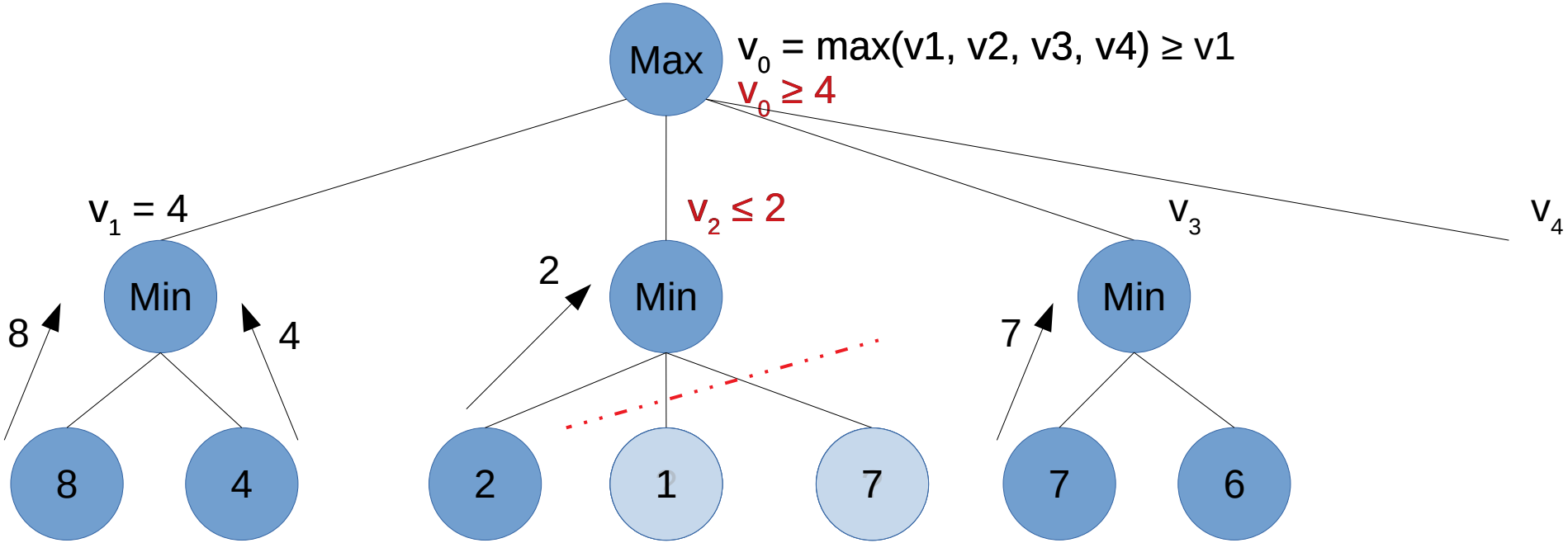
Améliorer l'efficacité : l'élagage α



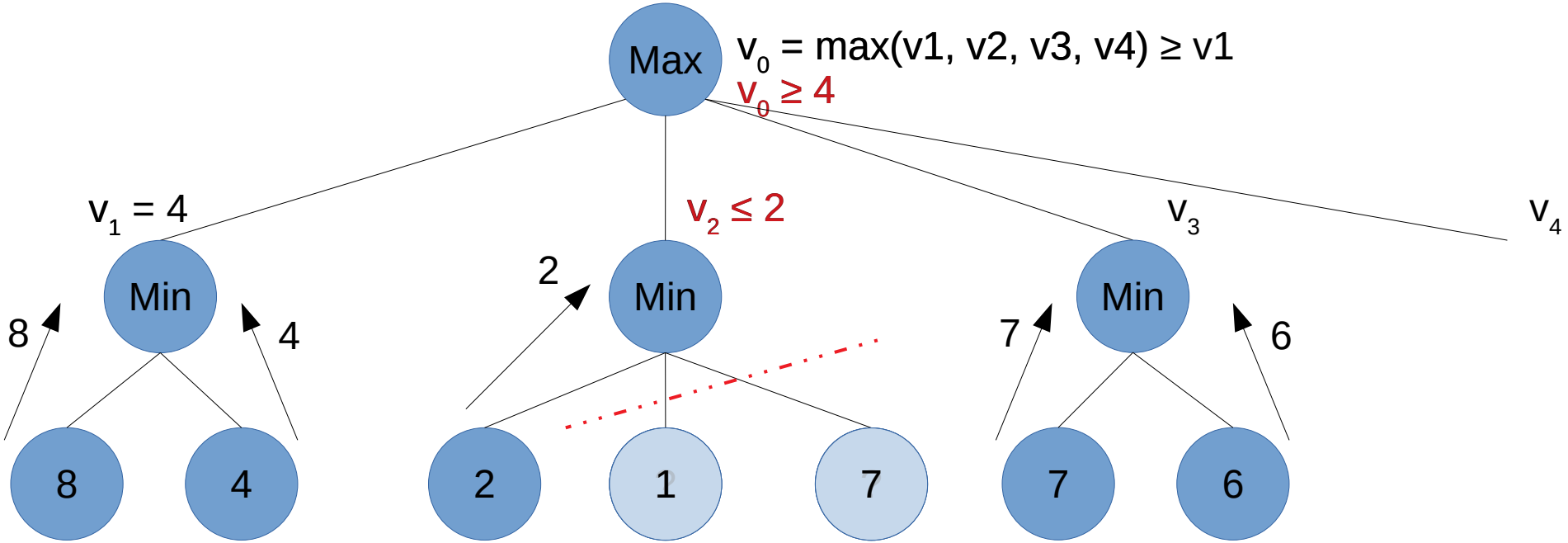
Améliorer l'efficacité : l'élagage α



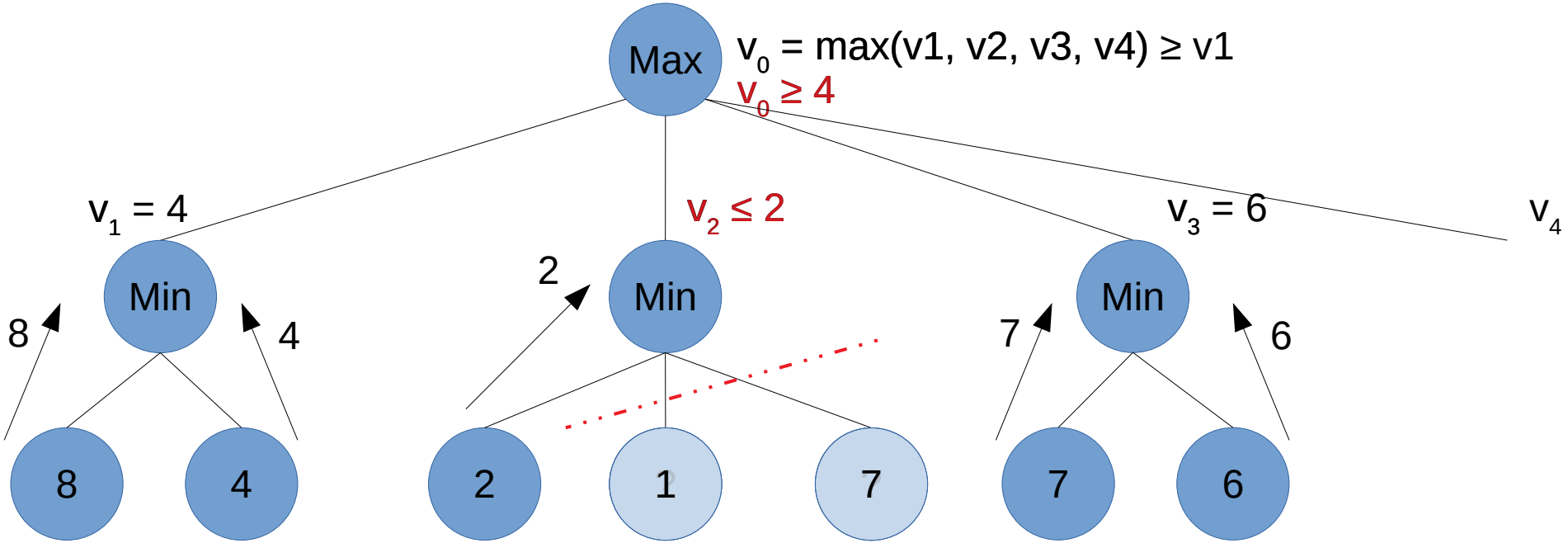
Améliorer l'efficacité : l'élagage α



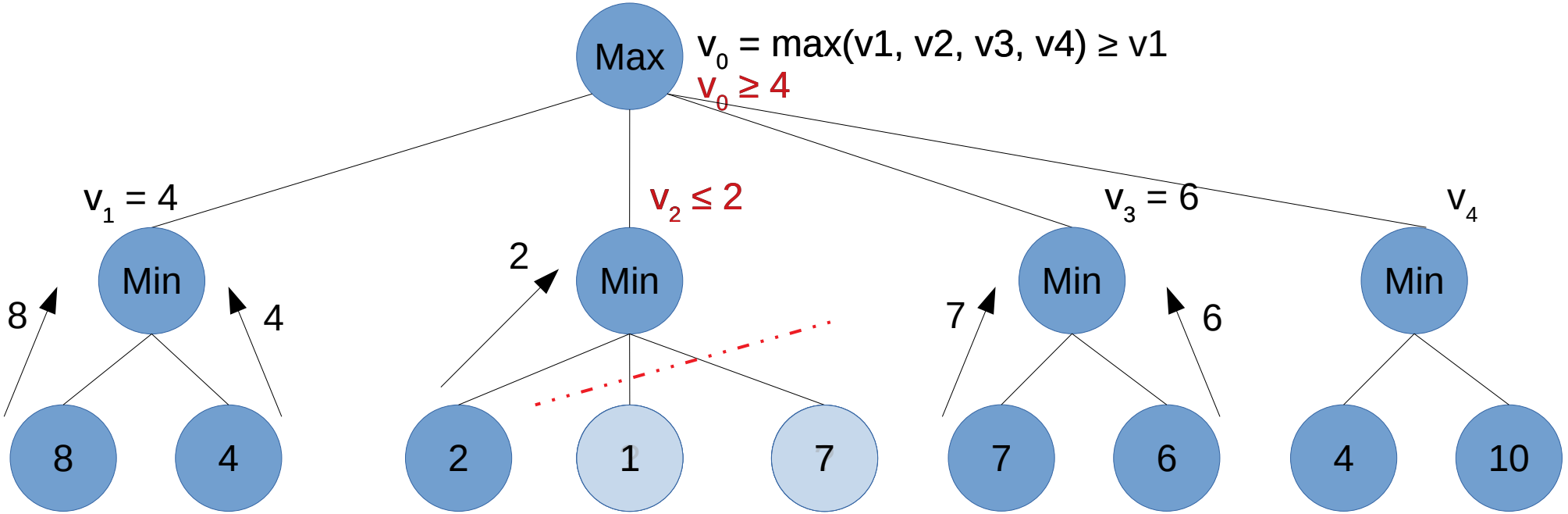
Améliorer l'efficacité : l'élagage α



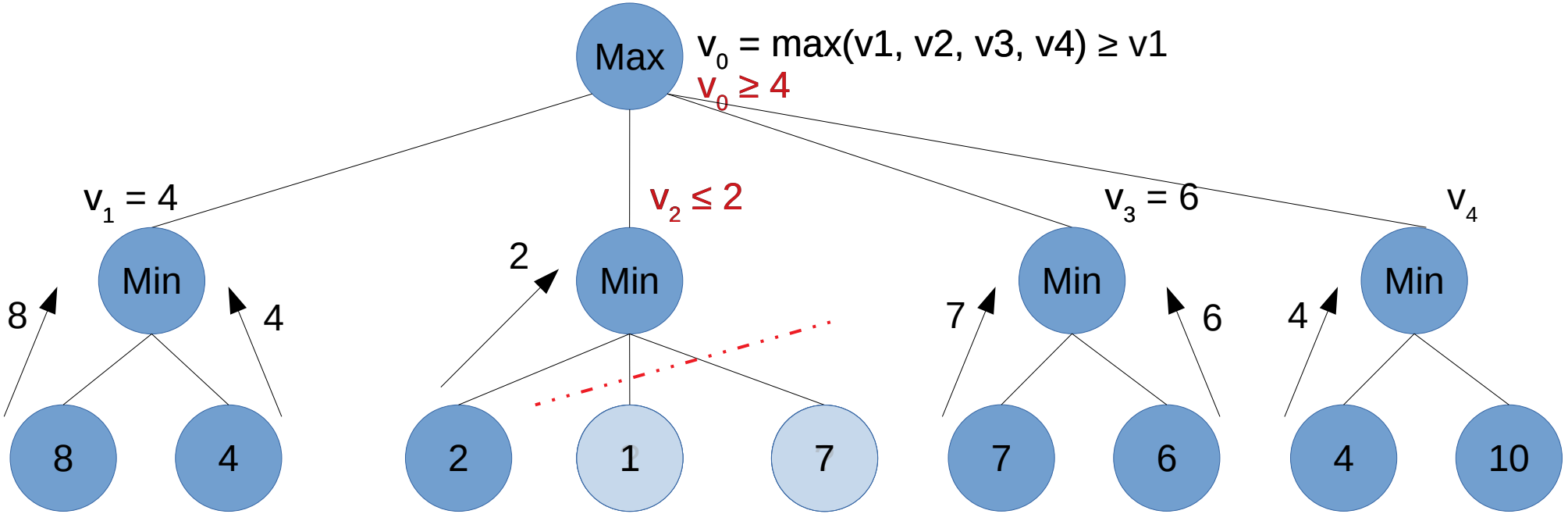
Améliorer l'efficacité : l'élagage α



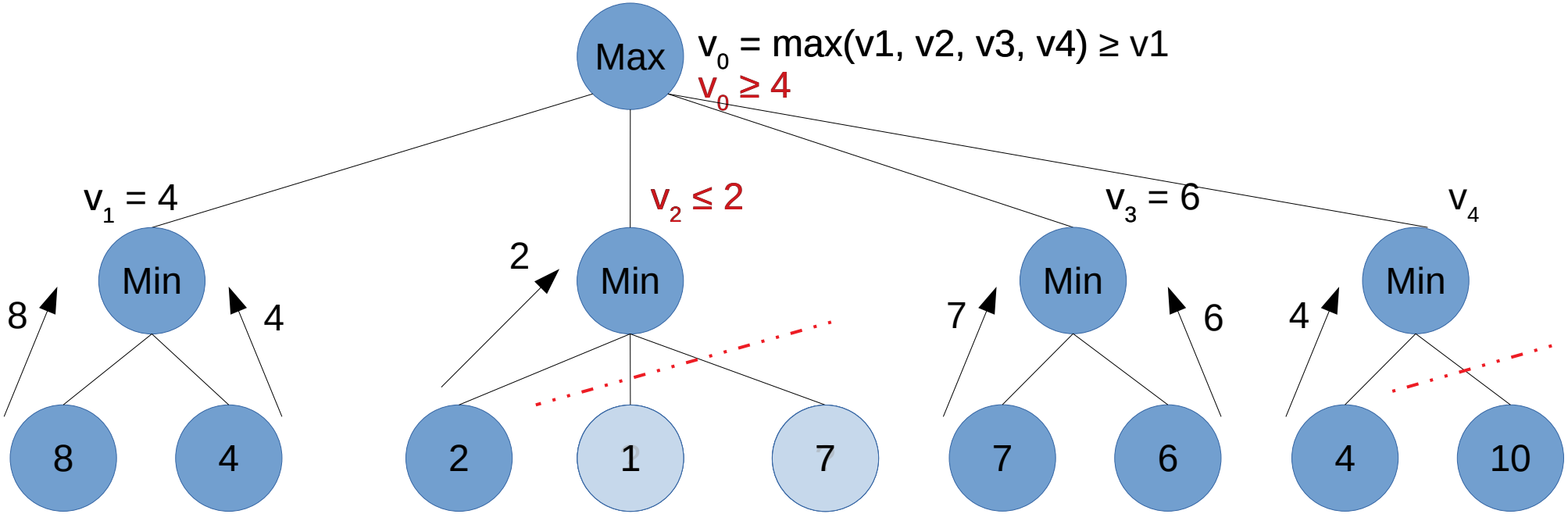
Améliorer l'efficacité : l'élagage α



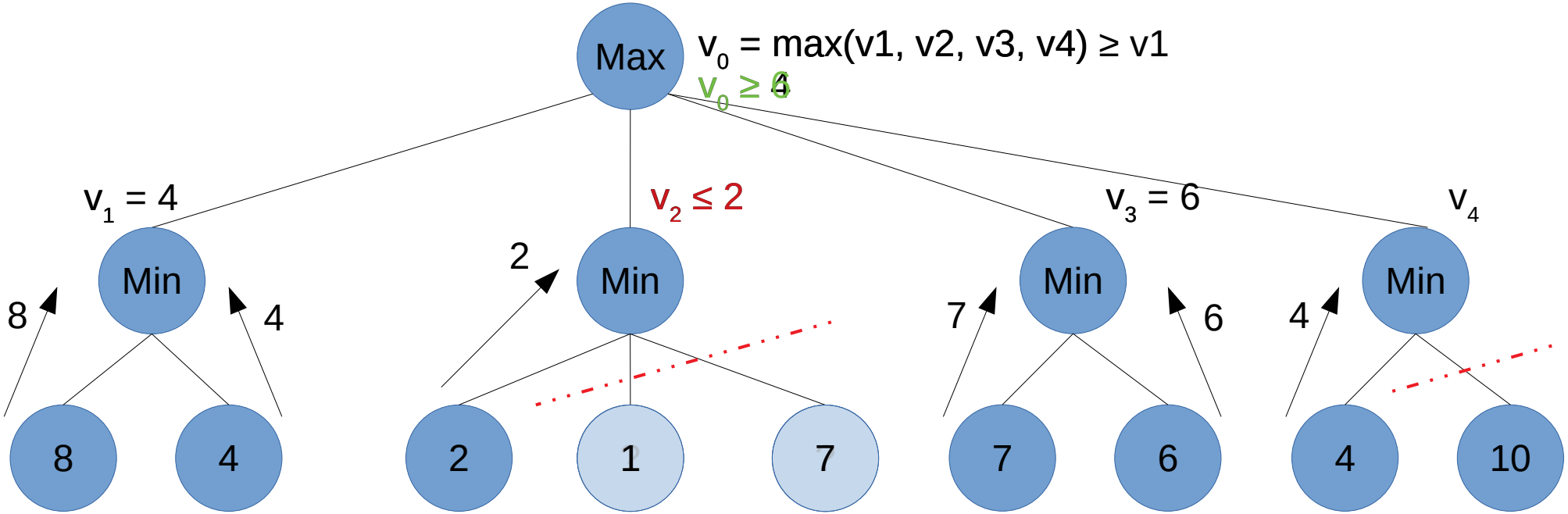
Améliorer l'efficacité : l'élagage α



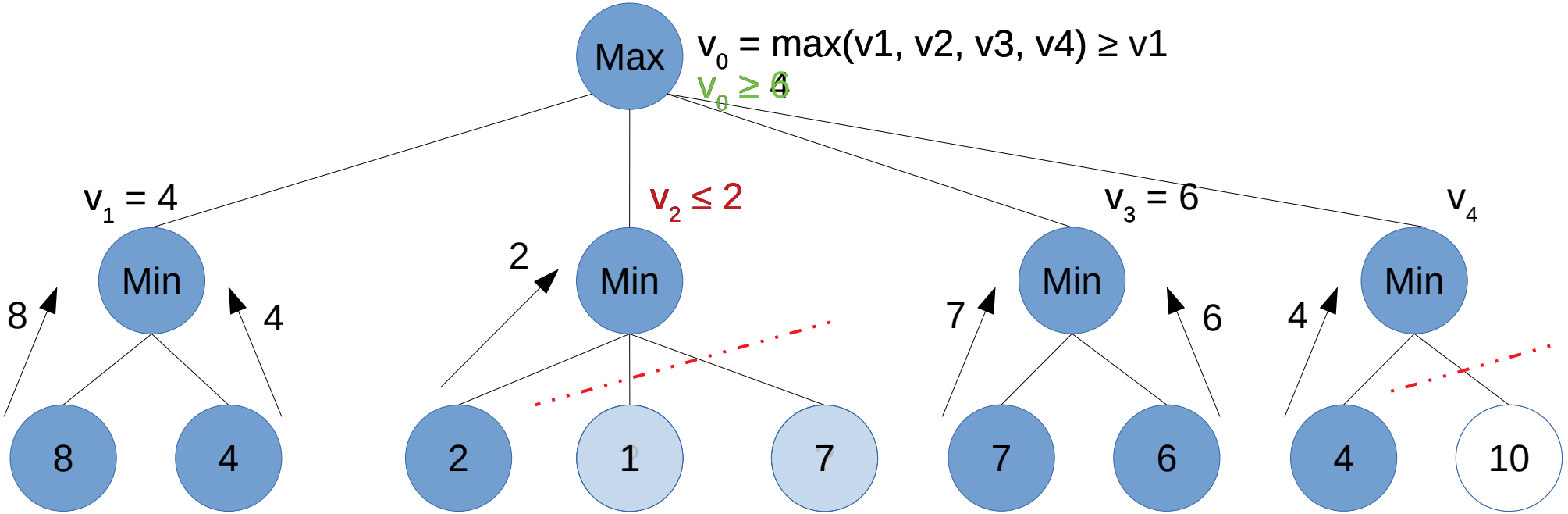
Améliorer l'efficacité : l'élagage α



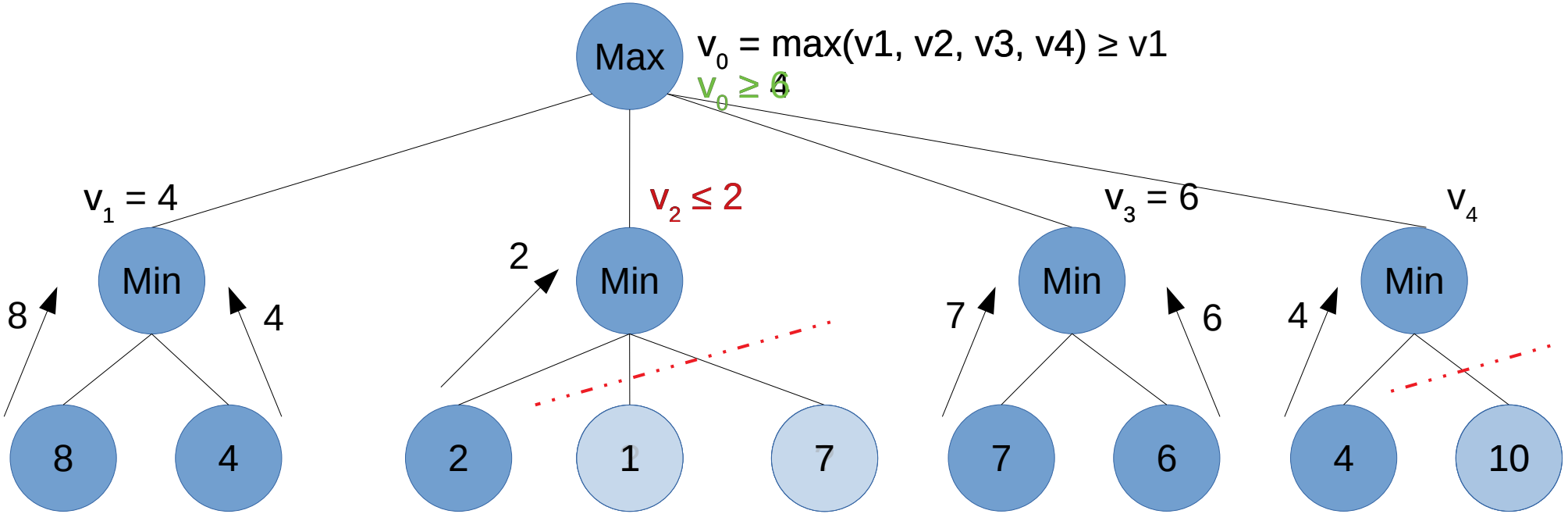
Améliorer l'efficacité : l'élagage α



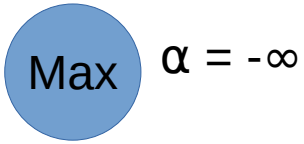
Améliorer l'efficacité : l'élagage α



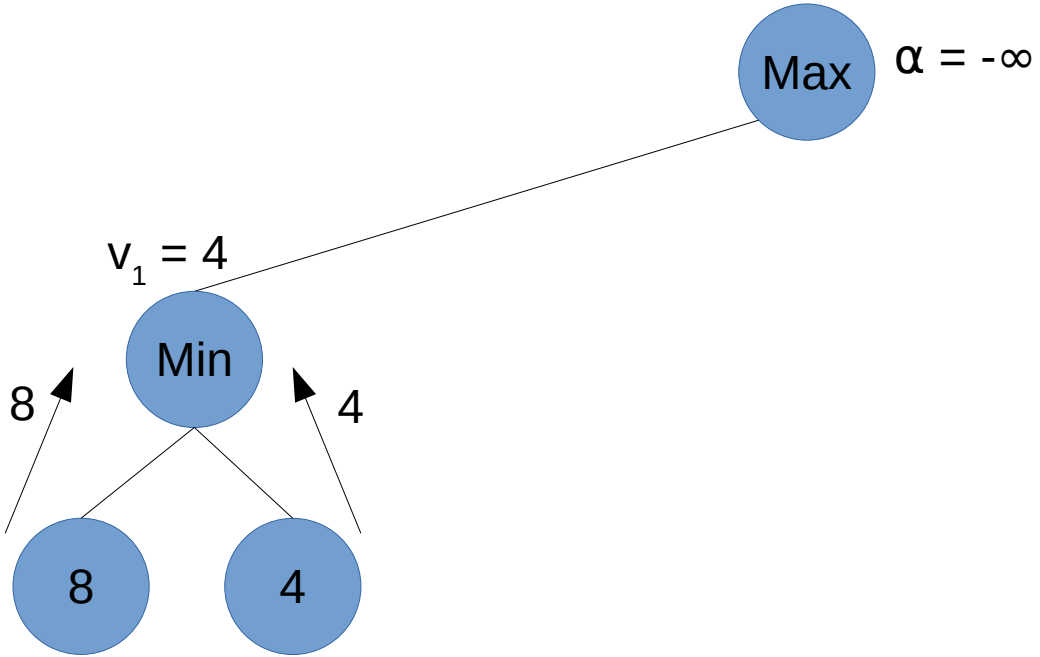
Améliorer l'efficacité : l'élagage α



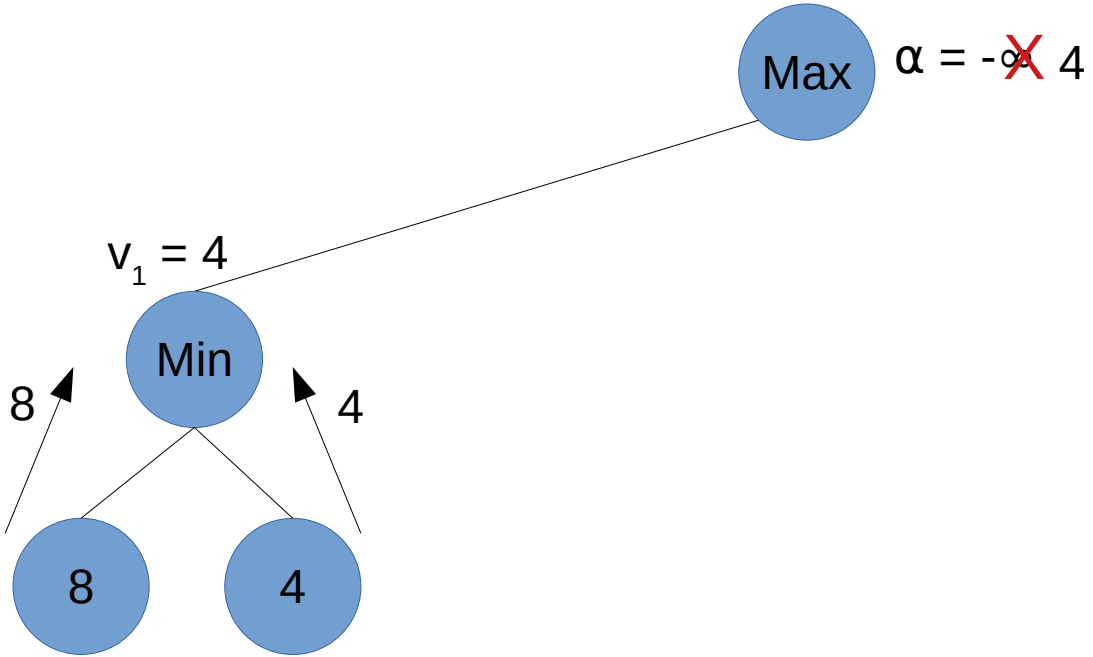
Améliorer l’efficacité : l’élagage α



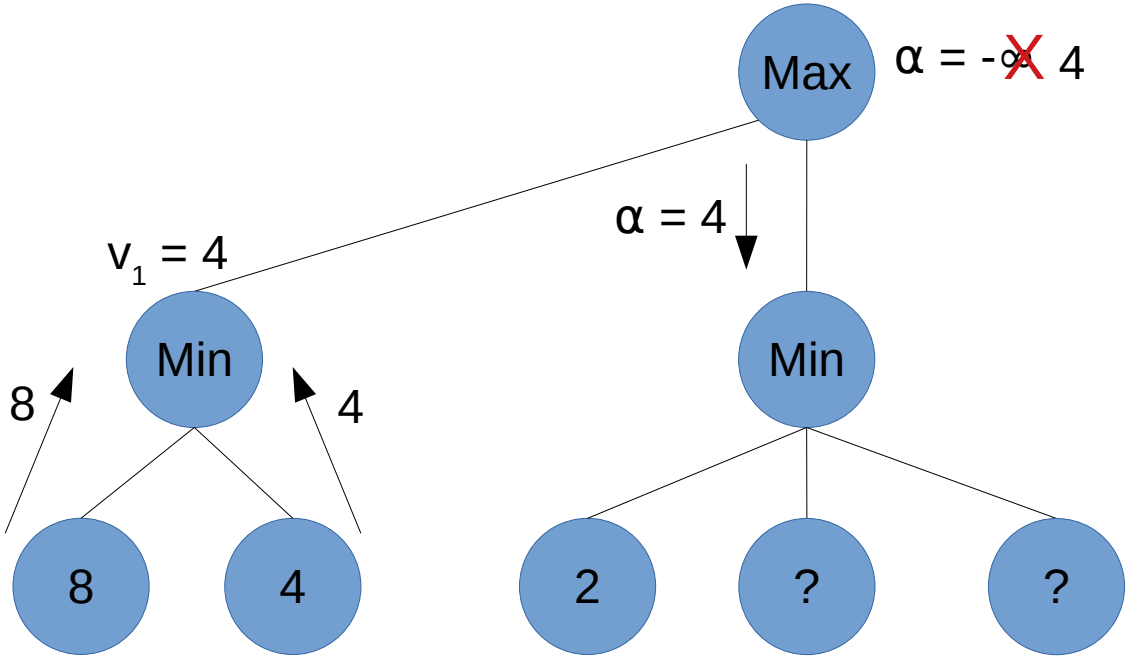
Améliorer l’efficacité : l’élagage α



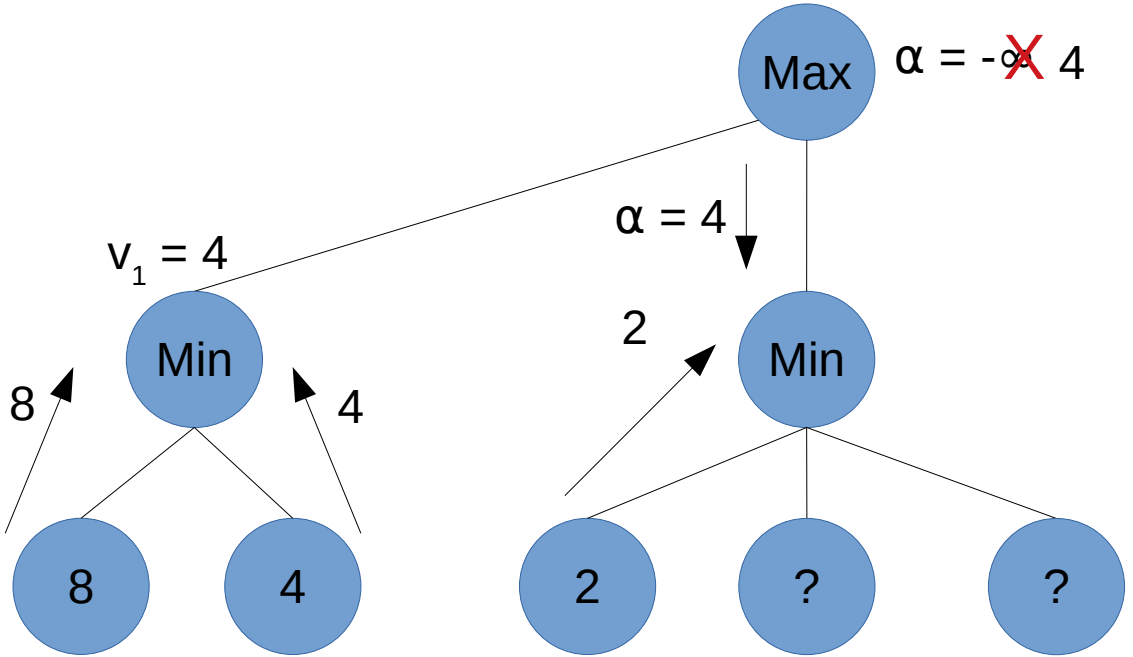
Améliorer l’efficacité : l’élagage α



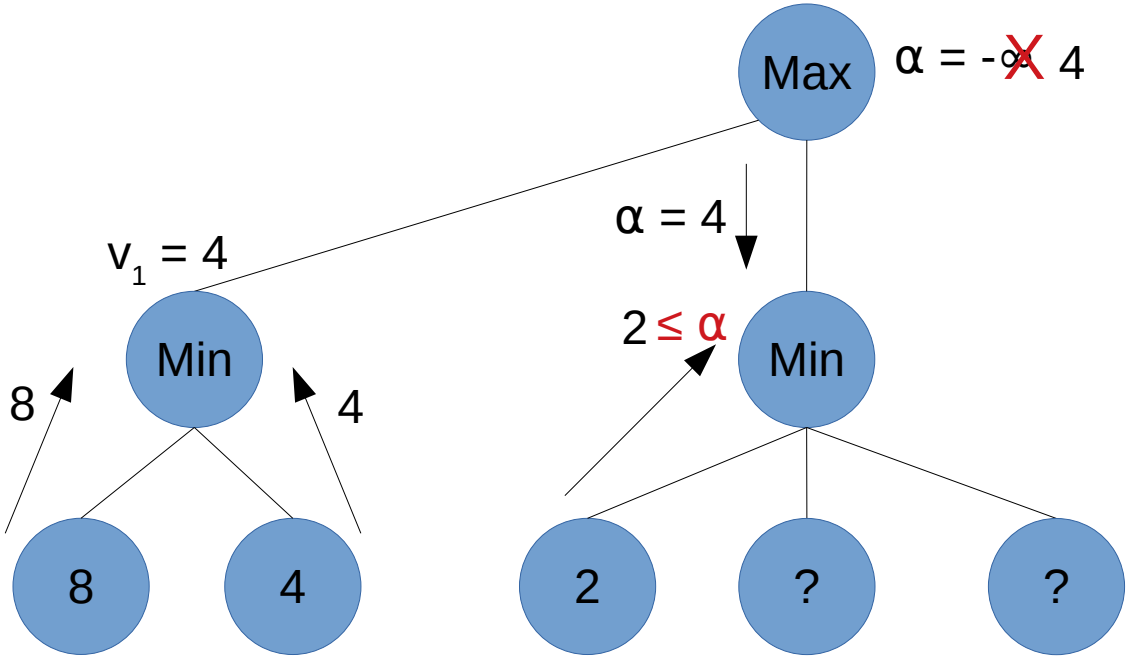
Améliorer l’efficacité : l’élagage α



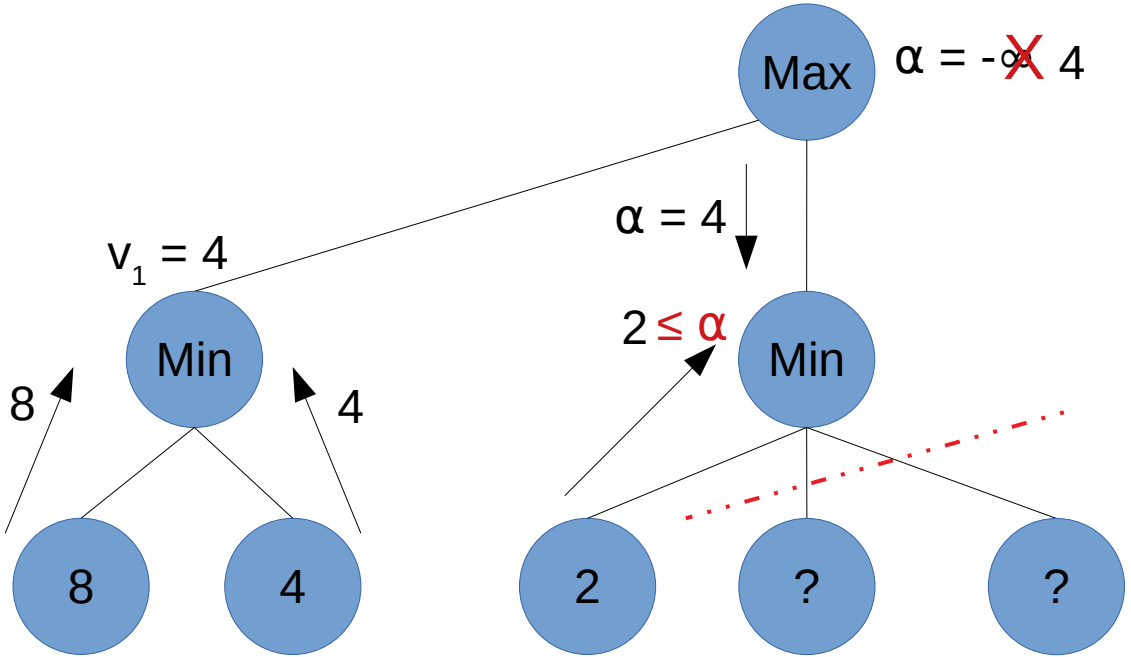
Améliorer l’efficacité : l’élagage α



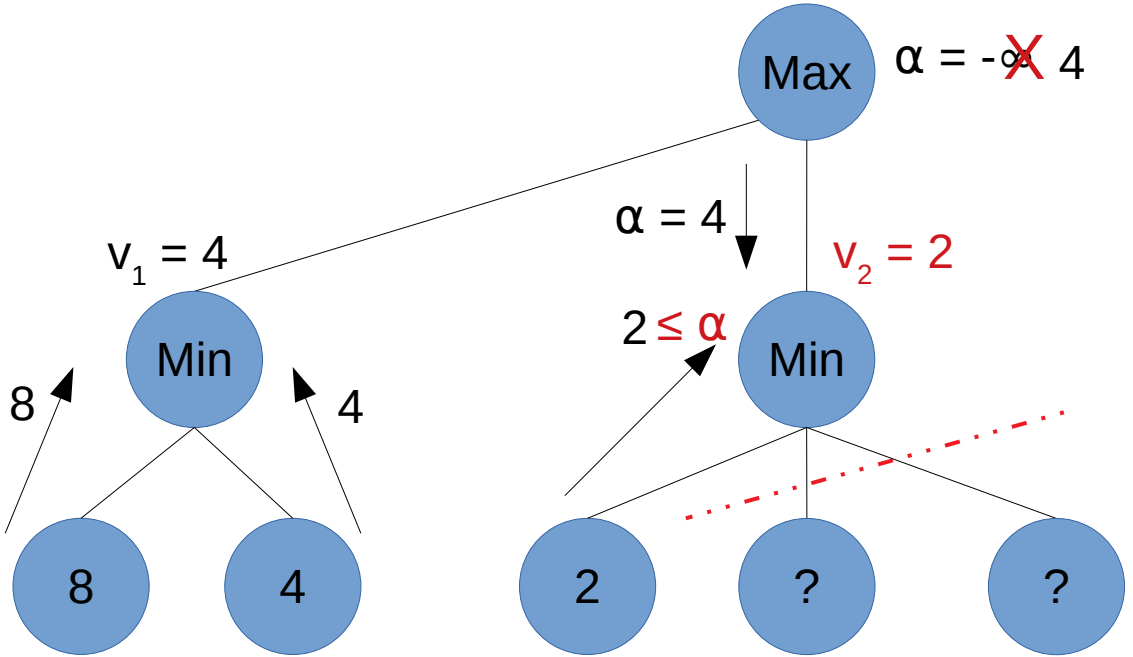
Améliorer l'efficacité : l'élagage α



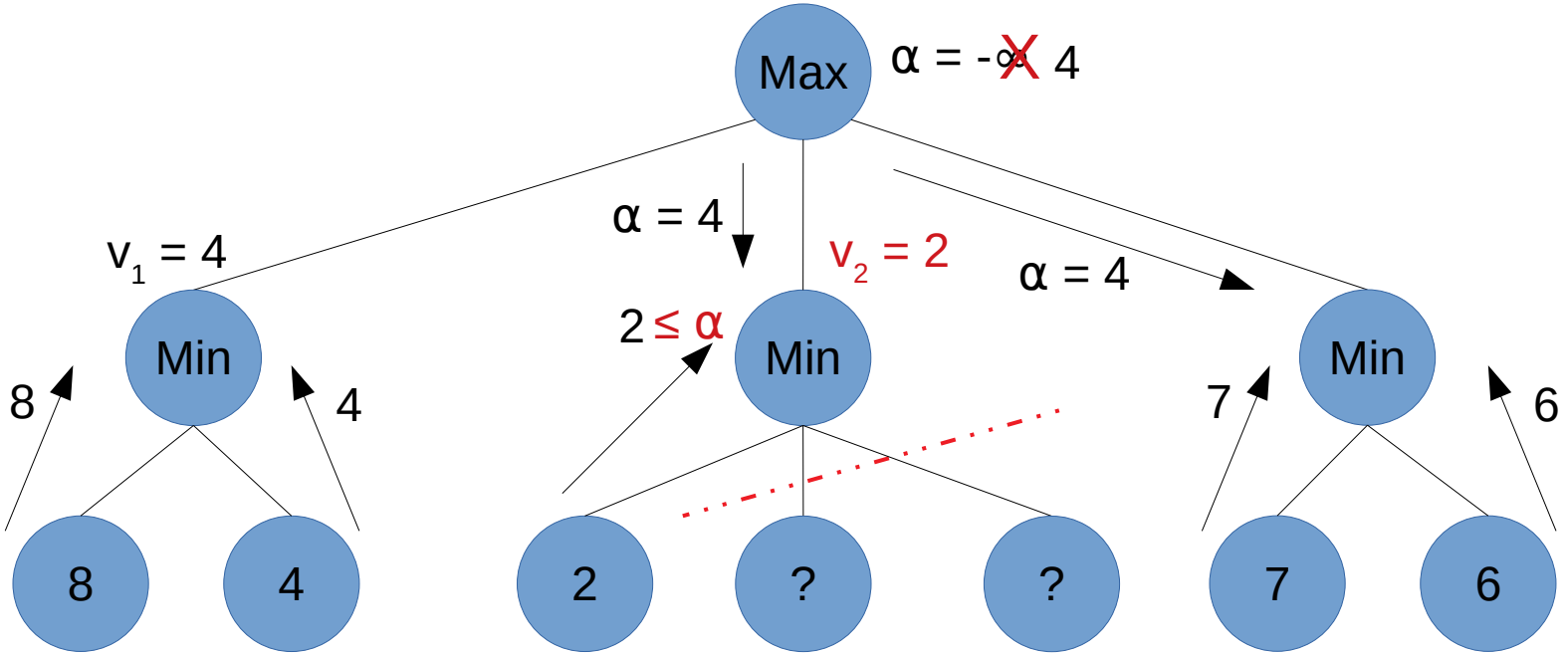
Améliorer l'efficacité : l'élagage α



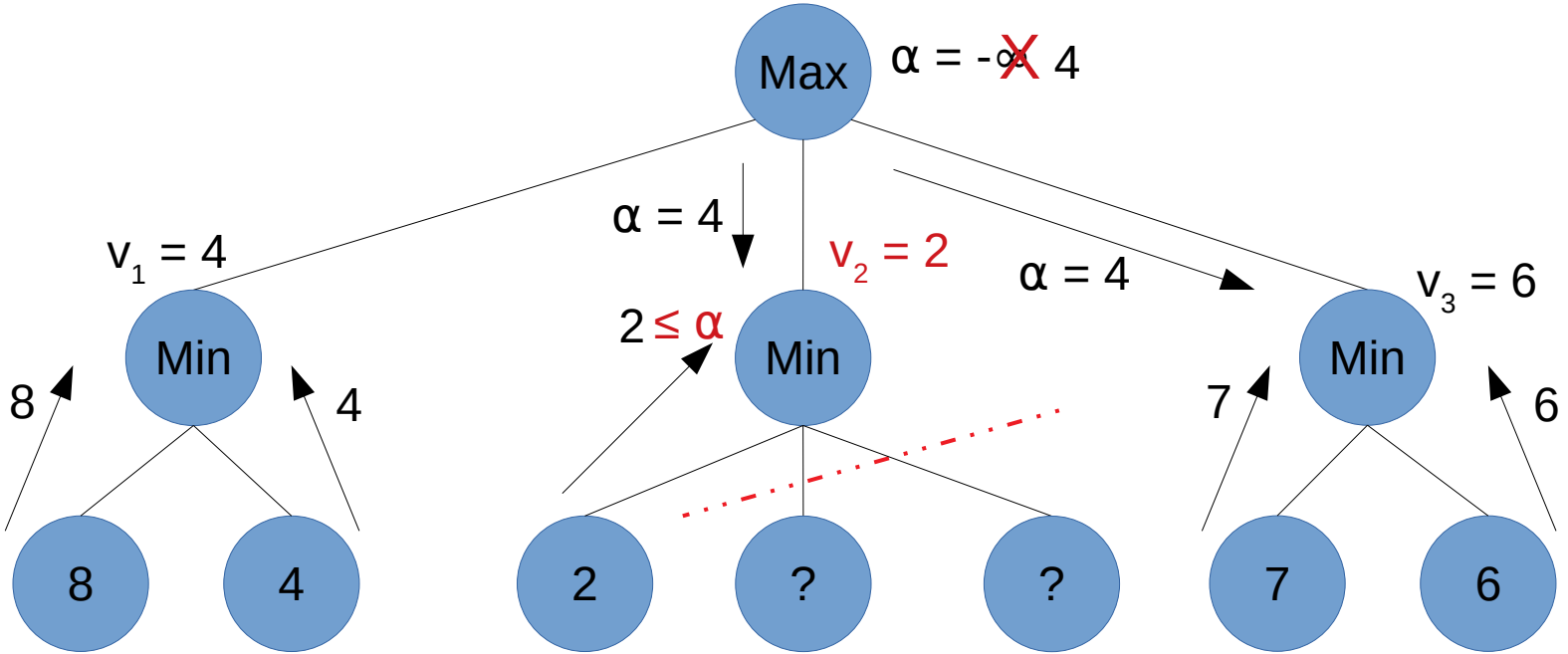
Améliorer l'efficacité : l'élagage α



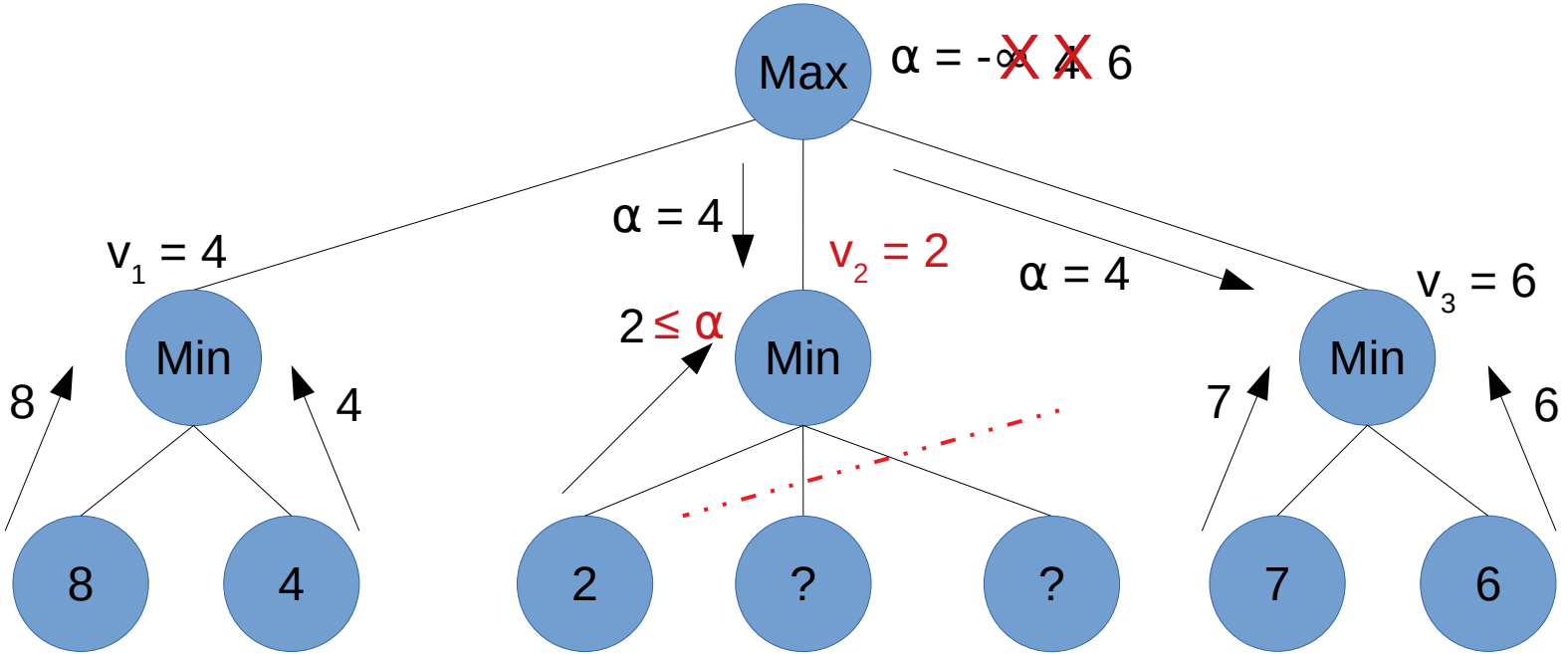
Améliorer l'efficacité : l'élagage α



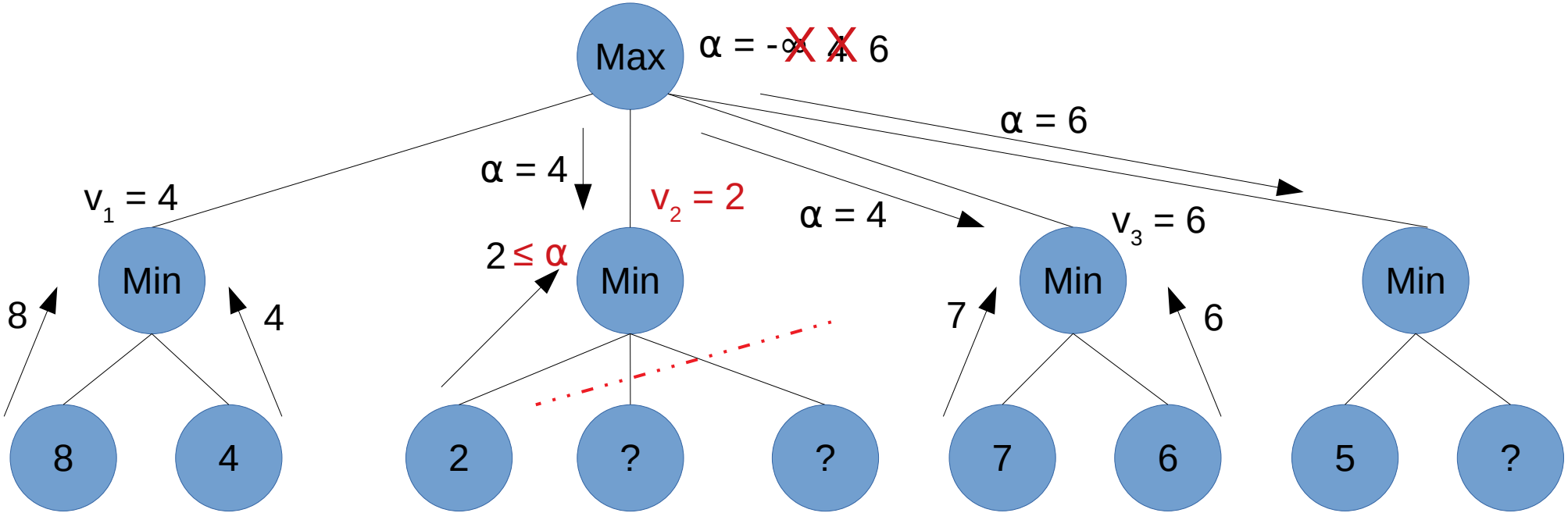
Améliorer l'efficacité : l'élagage α



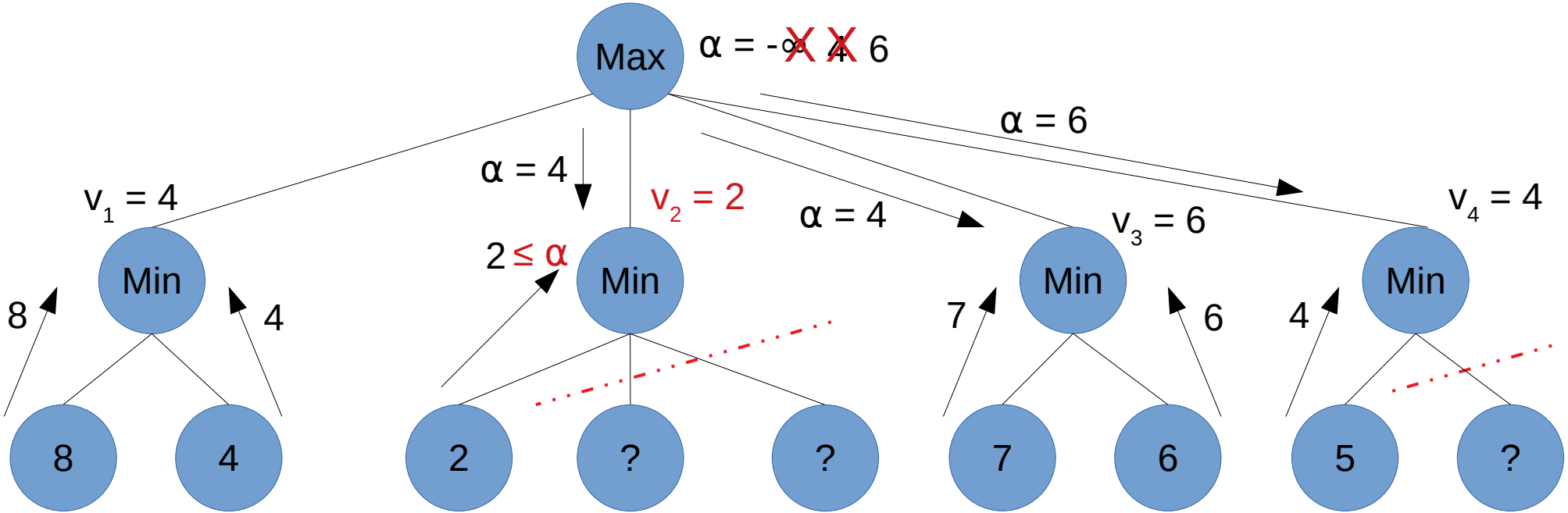
Améliorer l'efficacité : l'élagage α



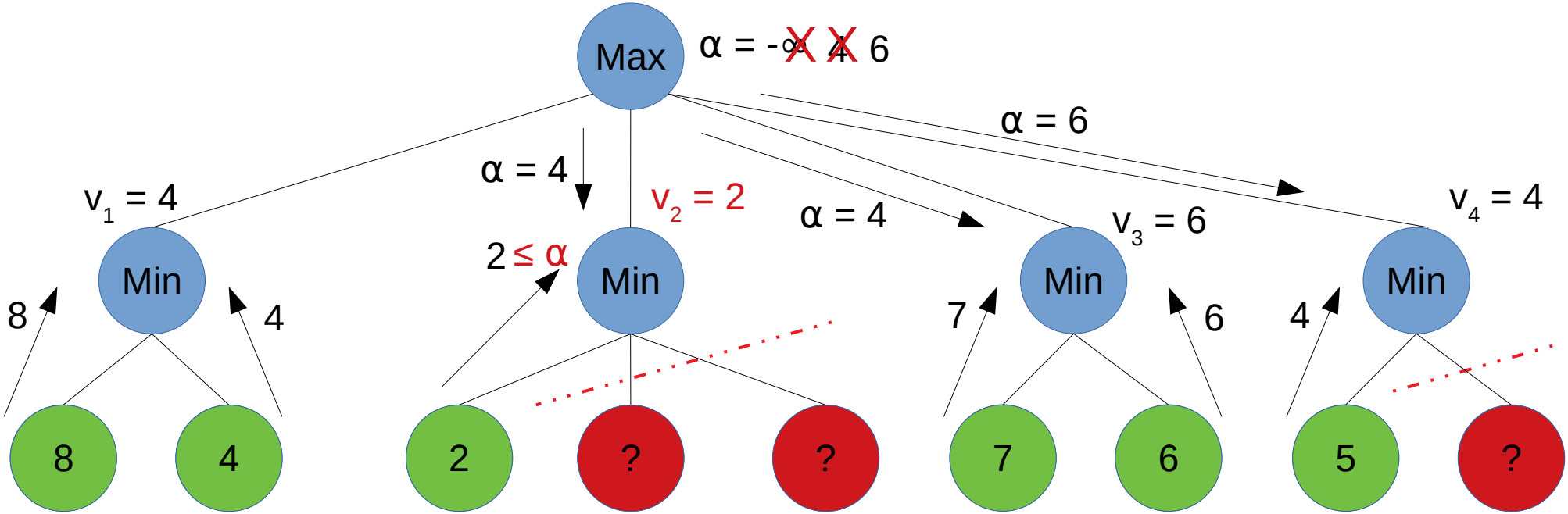
Améliorer l'efficacité : l'élagage α



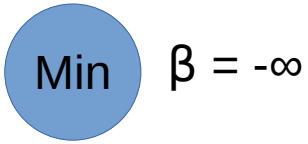
Améliorer l'efficacité : l'élagage α



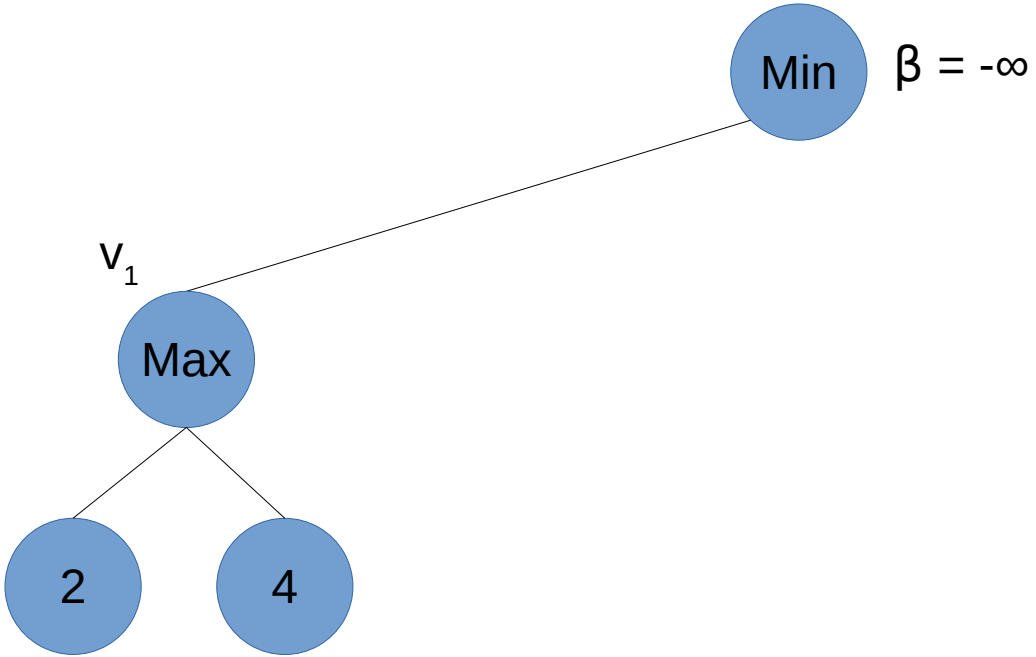
Améliorer l'efficacité : l'élagage α



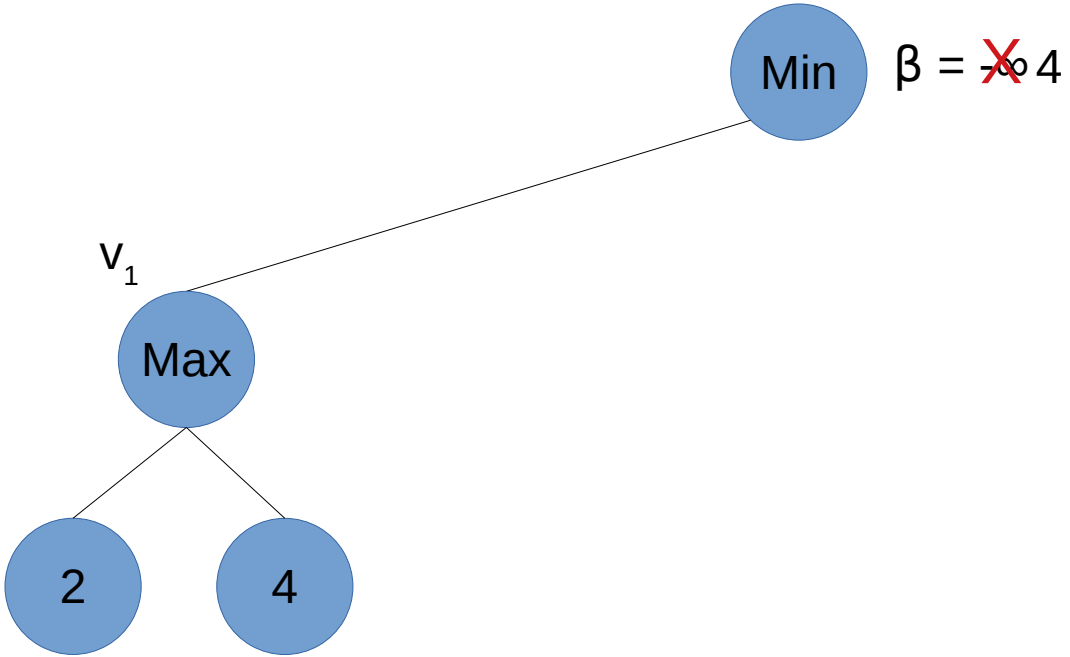
Améliorer l’efficacité : l’élagage β



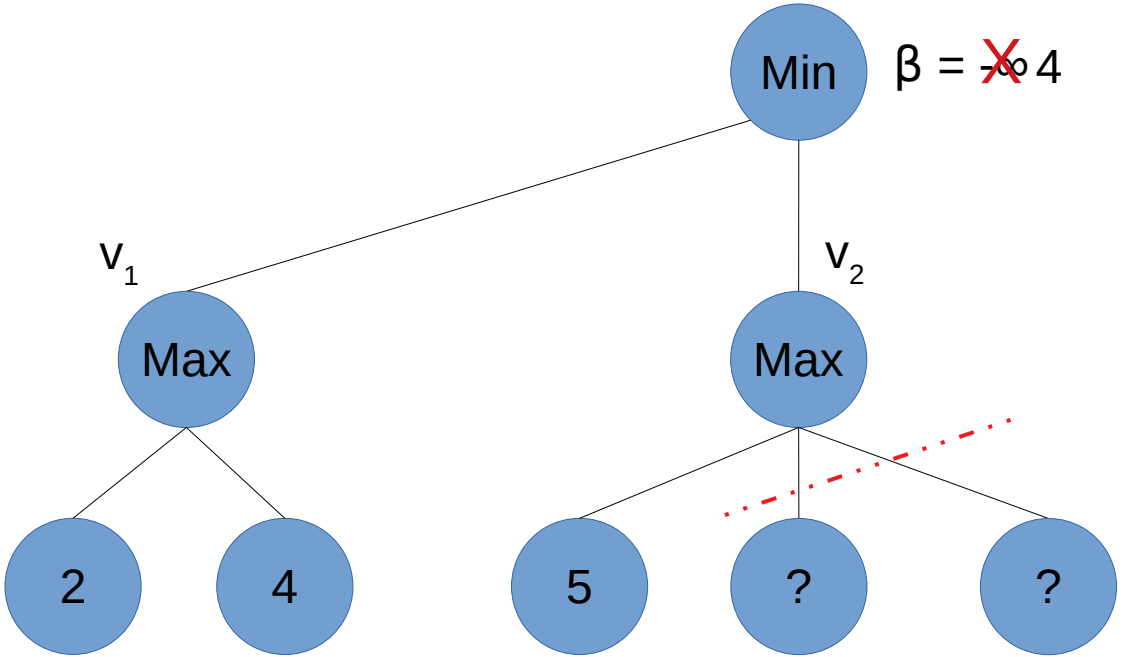
Améliorer l’efficacité : l’élagage β



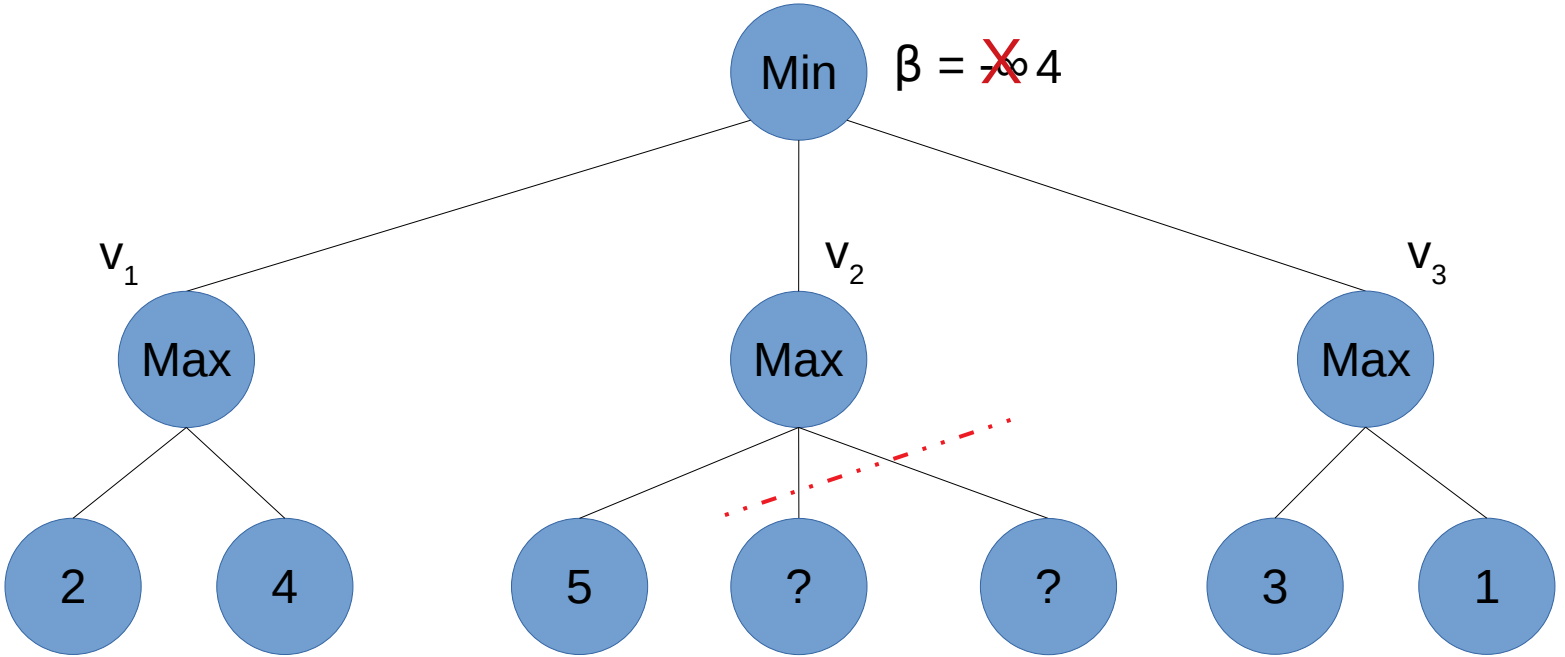
Améliorer l’efficacité : l’élagage β



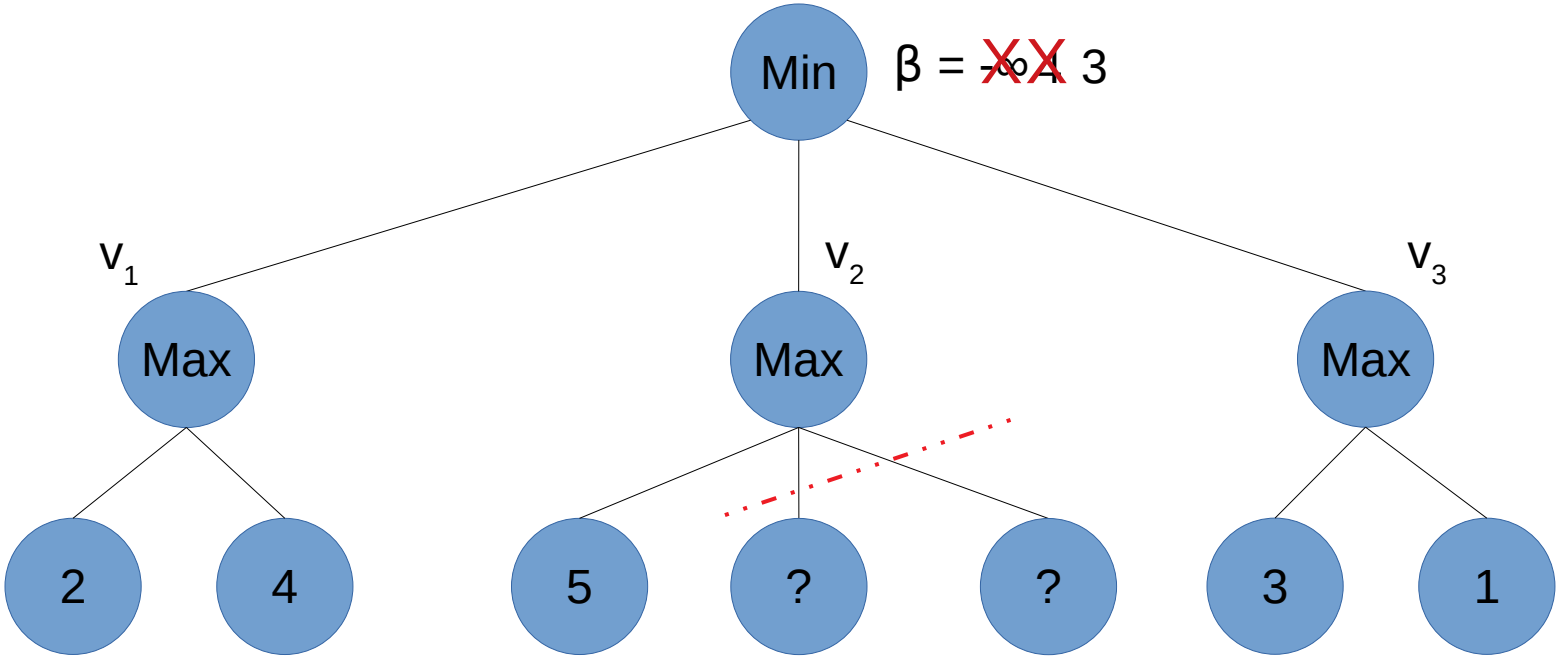
Améliorer l’efficacité : l’élagage β



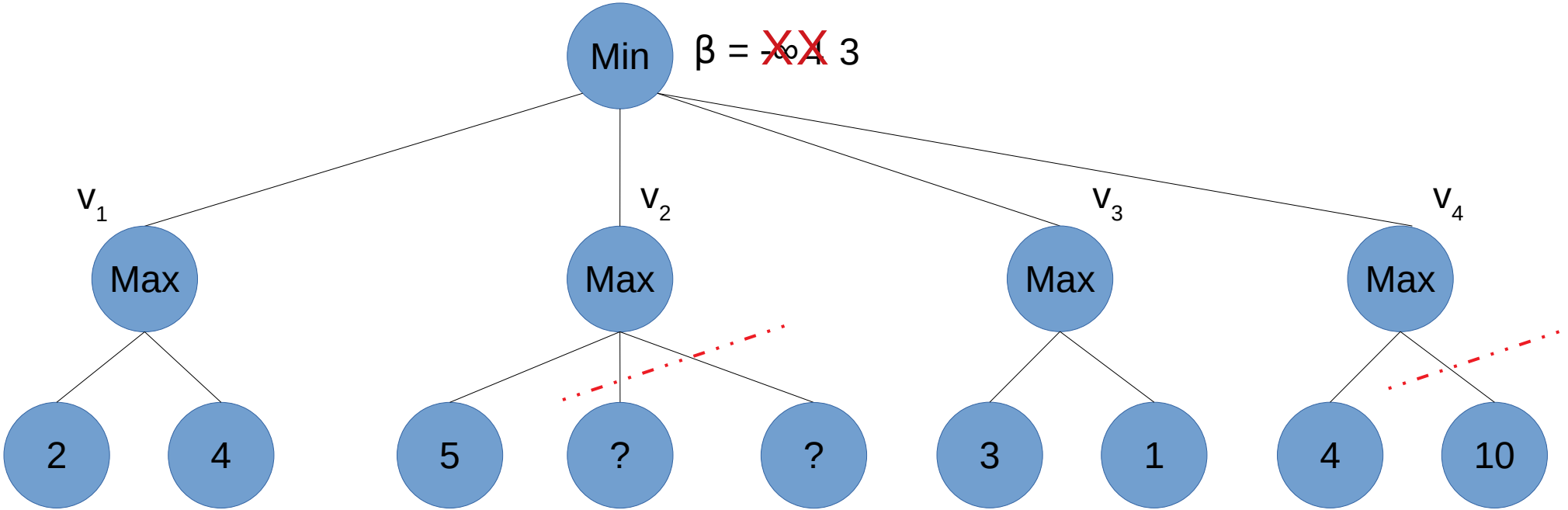
Améliorer l'efficacité : l'élagage β



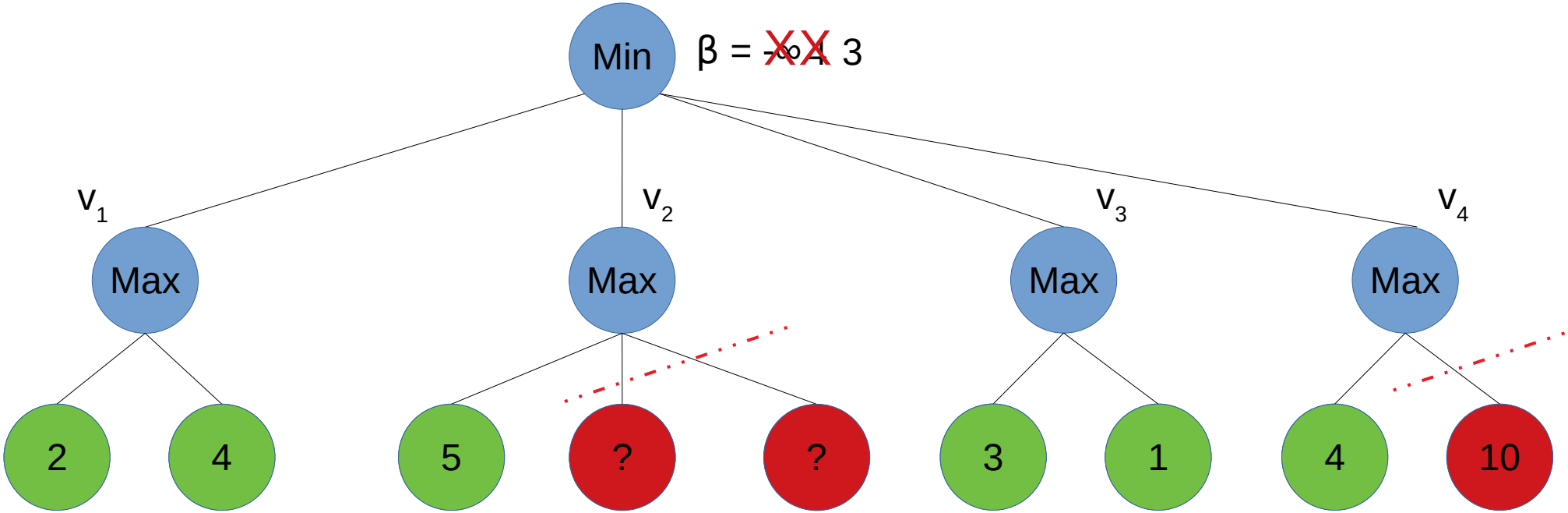
Améliorer l’efficacité : l’élagage β



Améliorer l'efficacité : l'élagage β



Améliorer l'efficacité : l'élagage β

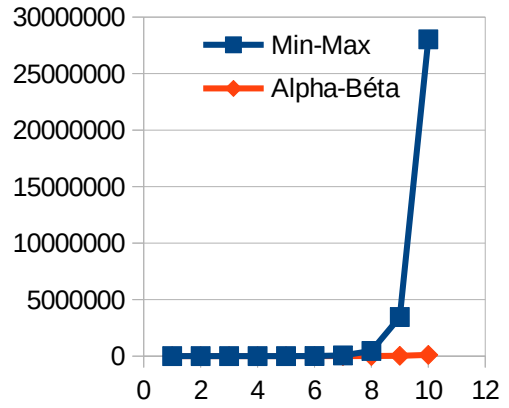


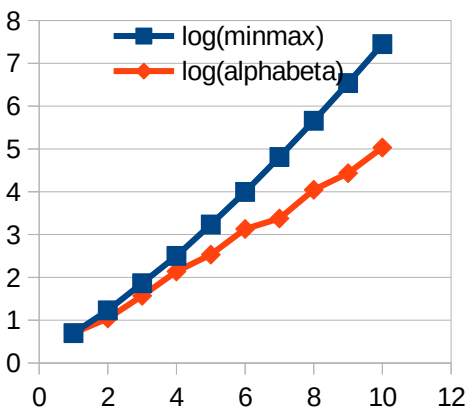
Améliorer l’efficacité : l’élagage α-β

- Appliquer un élagage α sur les nœuds « min »
- Appliquer un élagage β sur les nœuds « max

Élagage α-β
Exemple d'amélioration sur Othello

Nb demi-coups	Nb situations minmax	Nb situations « finales » minmax	Nb situations alphabeta	Nb situations « finales » alphabeta
1	5	4	5	4
2	17	12	11	6
3	73	56	37	25
4	317	244	137	90
5	1 713	1 396	339	238
6	9 913	8 200	1 348	881
7	65 005	55 092	2 369	1 684
8	455 221	390 216	11 132	7 467
9	3 460 541	3 005 320	27 190	19 697
10	28 031 961	24 571 420	107 714	74 481





Applicabilité en Python ?

Nb demi-coup	Tps calcul Min-Max	Tps calcul α-β
1	0,0025	0,0025
2	0,01	0,007
3	0,046	0,029
4	0,26	0,09
5	1,17	0,22
6	6,1	0,86
7	42	1,55
8	276	7,65
9	2 079	16,9
10		82

α-β en Python (1)

```
def alpha_beta(othello, joueur, prof_courante, prof_max, heuristique, alpha, beta):
    if prof_courante > prof_max:
        retour = ([], heuristique(othello, joueur))
    else:
        possibilites = othello.liste_coups()
        if len(possibilites) == 0:
            #Si aucun coup n'est jouable, on change de joueur et on continue
            situation = Othello(othello)
            situation.changer_joueur()
            enchainement, recompense = alpha_beta(situation, joueur, prof_courante+1, prof_max,
heuristique, alpha, beta)
            retour = (([None] + enchainement), recompense)
        else:
```

Il manque notamment :
- le traitement des victoires et défaites prématurées
- la gestion des situations bloquées

α-β en Python (2)

```
            #On s'intéresse maintenant aux cas où au moins 1 coup est jouable
            coup_retenu = None
            if tour_joueur_courant(prof_courante):
                # C’est le tour du joueur courant, on fait un « max »
                # et un éventuel élagage beta
                recompense_retenue = None
                enchainement_retenu = None
                for coup in possibilites:
                    nouvelle_situation = othello.tenter_coup(coup[0], coup[1])
                    liste_coups, recompense = alpha_beta(nouvelle_situation, joueur, prof_courante+1,
prof_max, heuristique, alpha, beta)
                    if recompense_retenue is None or recompense > recompense_retenue:
                        recompense_retenue = recompense
                        enchainement_retenu = liste_coups
                        coup_retenu = coup
                        if recompense_retenue >= beta:
                            break
                    if recompense > alpha:
                        alpha = recompense
```

α-β en Python (3)

```
            Else:
                # C'est le tour du joueur adverse, on fait un « min »
                # et un éventuel élagage alpha
                recompense_retenue = None
                enchainement_retenu = None
                for coup in possibilites:
                    nouvelle_situation = othello.tenter_coup(coup[0], coup[1])
                    liste_coups, recompense = alpha_beta(nouvelle_situation, joueur, prof_courante+1,
prof_max, heuristique, alpha, beta)
                    if recompense_retenue is None or recompense < recompense_retenue:
                        recompense_retenue = recompense
                        enchainement_retenu = liste_coups
                        coup_retenu = coup
                        if recompense_retenue <= alpha:
                            break
                    if recompense < beta:
                        beta = recompense
            retour = [coup] + enchainement_retenu, recompense_retenue
    if prof_courante == 1:
        print("nombre de situations examinées :",nb_situations_examinees)
    return retour
```

Applicabilité de α-β ?

- Stockfish (jeu d'échecs) utilise un algorithme type α-β. Mais avec :
  - Optimisations
  - Utilisation d'une mémoire colossale
  - Implantation pouvant utiliser jusqu'à 128 cœurs
- α-β n'est pas applicable à un jeu avec un arbre très large (comme le go)
- Applicable tel quel que sur des jeux à information complète (chaque joueur connaît complètement l'état du jeu)

Explorer en largeur ou en profondeur ?

- Inconvénient du parcours en profondeur
  - Souvent, le niveau de réflexion est plutôt défini par un temps

    → difficile de savoir jusqu'à quelle profondeur aller
- Inconvénient du parcours en largeur
  - Multiplication souvent exponentielle du nombre de situations à partir de la position initiale

    → occupation en mémoire colossale