

# Recuit simulé

---



Cet article est une ébauche concernant la science.

Vous pouvez partager vos connaissances en l’améliorant (**comment ?**) selon les recommandations des projets correspondants.

Le **recuit simulé** est une méthode empirique (métaheuristique) inspirée d'un processus utilisé en métallurgie. On alterne dans cette dernière des cycles de refroidissement lent et de réchauffage (*recuit*) qui ont pour effet de minimiser l'énergie du matériau. Cette méthode est transposée en optimisation pour trouver les extrema d'une fonction.

Elle a été mise au point par trois chercheurs de la société IBM, S. Kirkpatrick, C.D. Gelatt et M.P. Vecchi en 1983, et indépendamment par V. Černý en 1985.

La méthode vient du constat que le refroidissement naturel de certains métaux ne permet pas aux atomes de se placer dans la configuration la plus solide. La configuration la plus stable est atteinte en contrôlant le refroidissement et en le ralentissant par un apport de chaleur externe.

## Recuit réel

Article détaillé : recuit.

La description des phénomènes physiques et quantiques liés au processus de recuit s'appuie sur la statistique de Boltzmann.

## Déroulement du processus

Le recuit simulé s'appuie sur l'algorithme de Metropolis-Hastings, qui permet de décrire l'évolution d'un système thermodynamique. Par analogie avec le processus physique, la fonction à minimiser deviendra l'énergie  $E$  du système. On introduit également un paramètre fictif, la température  $T$  du système.

Partant d'une solution donnée, en la modifiant, on en obtient une seconde. Soit celle-ci améliore le critère que l'on cherche à optimiser, on dit alors qu'on a fait baisser l'énergie du système, soit celle-ci le dégrade. Si on accepte une solution améliorant le critère, on tend ainsi à chercher l'optimum dans le voisinage de la solution de départ. L'acceptation d'une « mauvaise » solution permet alors d'explorer une plus grande partie de l'espace de solution et tend à éviter de s'enfermer trop vite dans la recherche d'un optimum local.

## État initial de l'algorithme

La solution initiale peut être prise au hasard dans l'espace des solutions possibles. À cette solution correspond une énergie initiale  $E = E_0$ . Cette énergie est calculée en fonction du critère que l'on cherche à optimiser. Une température initiale  $T = T_0$  élevée est également choisie. Ce choix est alors totalement arbitraire et va dépendre de la loi de décroissance utilisée.

## Itérations de l'algorithme

À chaque itération de l'algorithme une modification élémentaire de la solution est effectuée. Cette modification entraîne une variation  $\Delta E$  de l'énergie du système (toujours calculée à partir du critère que l'on cherche à optimiser). Si cette variation est négative (c'est-à-dire qu'elle fait baisser l'énergie du système), elle est appliquée à la solution courante. Sinon, elle est acceptée avec une probabilité  $e^{-\frac{\Delta E}{T}}$ . Ce choix de l'exponentielle pour la probabilité s'appelle règle de Metropolis.

On itère ensuite selon ce procédé en gardant la température constante.

## Programme de recuit

Deux approches sont possibles quant à la variation de la température :

1. Pour la première, on itère en gardant la température constante. Lorsque le système a atteint un équilibre thermodynamique (au bout d'un certain nombre de changements), on diminue la température du système. On parle alors de *paliers* de température.
2. La seconde approche fait baisser la température de façon continue. On peut imaginer toute sorte de loi de décroissance, la plus courante étant  $T_{i+1} = \lambda T_i$  avec  $\lambda < 1$  (assez couramment  $\lambda = 0.99$ ).

Dans les deux cas, si la température a atteint un seuil assez bas fixé au préalable ou que le système devient figé, l'algorithme s'arrête.

La température joue un rôle important. À haute température, le système est libre de se déplacer dans l'espace des solutions (  $e^{-\frac{\Delta E}{T}}$  proche de 1) en choisissant des solutions ne minimisant pas forcément l'énergie du système. À basse température, les modifications baissant l'énergie du système sont choisies, mais d'autres peuvent être acceptées, empêchant ainsi l'algorithme de tomber dans un minimum local.

## Pseudo-code

Le pseudo-code suivant met en œuvre le recuit simulé tel que décrit plus haut, en commençant à l'état  $s_0$  et continuant jusqu'à un maximum de  $k_{max}$  étapes ou jusqu'à ce qu'un état ayant pour énergie  $e_{max}$  ou moins soit trouvé. L'appel `voisin(s)` génère un état voisin aléatoire d'un état  $s$ . L'appel `aléatoire()` renvoie une valeur aléatoire dans l'intervalle  $[0, 1]$ . L'appel `temp(r)` renvoie la température à utiliser selon la fraction  $r$  du temps total déjà dépensé.

```
s := s0
e := E(s)
k := 0
tant que k < kmax et e > emax
  sn := voisin(s)
  en := E(sn)
  si en < e ou aléatoire() < P(en - e, temp(k/kmax)) alors
    s := sn; e := en
  k := k + 1
retourne s
```

## Inconvénients

Les principaux inconvénients du recuit simulé résident dans le choix des nombreux paramètres, tels que la température initiale, la loi de décroissance de la température, les critères d'arrêt ou la longueur des paliers de température. Ces paramètres sont souvent choisis de manière empirique.

## Étude théorique

Des études théoriques du recuit simulé ont pu montrer que sous certaines conditions, l'algorithme du recuit convergeait vers un optimum global<sup>[1]</sup>. Ce résultat est important car il nous assure, contrairement à d'autres métaheuristiques, que le recuit simulé peut trouver la meilleure solution, si on le laisse chercher indéfiniment.

Les preuves de convergence reposent sur le principe de construction de l'algorithme du recuit simulé :

- Sous réserve qu'elle respecte la condition de Doeblin<sup>[2]</sup> une chaîne de Markov admet une unique distribution stationnaire

- L'algorithme de Metropolis-Hastings permet, étant donnée une distribution de probabilité, de construire une chaîne de Markov dont la distribution stationnaire est cette distribution de probabilité.

Par ailleurs, les mesures de Gibbs, très utilisées en physique statistique, sont une famille de mesures dépendant d'un paramètre température de telle sorte que lorsque la température tend vers 0, la mesure converge vers un Dirac en un point donné (état d'énergie minimale).

En composant les deux, on obtient une chaîne de Markov qui converge vers une mesure invariante qui évolue dans le même temps vers une mesure désignant l'optimum recherché (au sens de la fonction d'énergie associée à la mesure de Gibbs).

Une analyse fine des vitesses de convergence et des écarts asymptotique permet alors d'aboutir à diverses conditions de convergence, dont celle de Hajek : si le schéma de température utilisé vérifie  $T_n \geq \frac{K}{\ln n}$ , alors l'algorithme

converge en probabilité vers l'optimum global.

En pratique, ce schéma converge beaucoup trop lentement et l'on préfère des températures à décroissance linéaire ou quadratique, quitte à n'obtenir qu'un minimum local (qu'on espère proche du minimum global).

## Applications

Comme pour toute métaheuristique, la méthode trouve son application dans de nombreux domaines dans lesquels on a à résoudre des problèmes d'optimisation difficile. On peut citer entre autres la conception des circuits électroniques (placement-routage) ou l'organisation de réseaux informatiques.

## Exemple sous Mathematica

```
F[X_] := Abs[X[[1]]] + Abs[X[[2]]];
```

```
g[T_] := N[T^0.99];
```


```
Recuit[F_, g_, Xi_, Ti_, Tf_, Ampli_, MaxTconst_, itermax_] := Module[{T, Xopt, iter=1, DF, p, L, X, compteur},
  X = Xi; T = Ti; L = {Xi}; Xopt = X;
  While [T > Tf && iter < itermax,
    compteur = 1;
    While[compteur < MaxTconst,
      Y = X + Table[Ampli*Random[Real, {-1, 1}], {i, 1, Length[X]}];
      DF = F[Y] - F[X];
      If[DF < 0, X = Y; AppendTo[L, X];
        If[F[X] < F[Xopt], Xopt = X],
        p = Random[];
      If [p ≤ Exp[-DF/T], X = Y; AppendTo[L, X]];
    ];
    compteur = compteur + 1;
  ];
  T = g[T];
  iter = iter + 1;
];
Return[L]
```

```
resR = Recuit[F, g, {2, 3}, 10, 1, 0.5, 1.5, 1000];
ListPlot[resR, Joined -> True];
```

## Notes et références

- [1] Aarts, E. H. L., and v. Laarhoven (1985), Statistical cooling: A general approach to combinatorial optimization problems, Philips J. Res., 40(4), 193-226.
- [2] <http://www.proba.jussieu.fr/cours/dea/telehtml/node37.html>

## Liens externes

- recuit simulé ([http://interstices.info/jcms/c\\_43811/le-recuit-simule](http://interstices.info/jcms/c_43811/le-recuit-simule)) (Applet java)
- recuit simulé pour le voyageur de commerce, application en ligne ([http://www.irisa.fr/prive/fschwarz/mit1\\_algo2\\_2013/tsp/](http://www.irisa.fr/prive/fschwarz/mit1_algo2_2013/tsp/))
-  Portail de l'algorithmique

# Sources et contributeurs de l'article

**Recuit simulé** *Source:* <http://fr.wikipedia.org/w/index.php?oldid=94441285> *Contributeurs:* Accel, Ancmin, Anne Bauval, Badmood, Corax26, Creuzal, David Berardan, Dorbec, En passant, Erasmus, Francois Trazzi, Gh, Gzen92, Hemorragie, Jerome66, Jill-Jënn, Kelson, Lc, Levochik, Léna, Med, NicoV, Nojhan, Perditax, Phe, R, Rhadamante, Sam Hocevar, Sylenius, Verbex, Wakarex, Xofc, 35 modifications anonymes

## Source des images, licences et contributeurs

**Image: Science-symbol-2.svg** *Source:* <http://fr.wikipedia.org/w/index.php?title=Fichier:Science-symbol-2.svg> *Licence:* Creative Commons Attribution 3.0 *Contributeurs:* en:User:AllyUnion, User:Stannered

**Fichier:Konigsburg graph.svg** *Source:* [http://fr.wikipedia.org/w/index.php?title=Fichier:Konigsburg\\_graph.svg](http://fr.wikipedia.org/w/index.php?title=Fichier:Konigsburg_graph.svg) *Licence:* GNU Free Documentation License *Contributeurs:* AnonMoos, Piotrus, Riojajar, Squizz

## Licence

---

Creative Commons Attribution-Share Alike 3.0  
[//creativecommons.org/licenses/by-sa/3.0/](http://creativecommons.org/licenses/by-sa/3.0/)

---