Garnet

# 機能要件

* コンポーネントごとにテストできるようにしたい．
  + モジュールごとに，入出力をちゃんと定義して，任意のモジュールを起動できるようにしておきたい．
* 起動するモジュールの指定
  + １つの機能を実行するモジュール
  + 全体を流して実行するラッパーモジュール

全体の流れ

# モジュール

## 概要

* 個々のモジュールは単独では，メモリ上のデータをやりとりするだけ．なので，UIから呼び出すためには，設定ファイルの読み込みルーチンと結果の出力ルーチンを作る必要がある．
* 設定ファイルの形式は原則としてすべてConfigLoader形式．出力もそれを基本とする．

## garnet

* + 課題🡪 IVフィルタツリー（の集合）

## fitnessvector

Raw Fitness評価/Fitness Vector構築（fitnessvector）：IVツリーの集合🡪 Fitness Vectorの集合

## tree2graph

IVツリーアグリゲーション IVツリーの集合🡪 IVグラフ

## rawfitness

Raw Fitness評価：IVグラフ🡪 Raw Fitnessの集合（Archiveとして）

## iv2ppe

：IVグラフ🡪 PPEグラフ

## genscript

変換２：PPEグラフ🡪 PPEスクリプト

## runscript

PPEスクリプトの実行：PPEスクリプト🡪 Raw Fitnessの集合（実行結果を含むArchiveとして）

## genfitness

Fitness Vector構築 (genfitness)： Raw Fitnessの集合, IVツリーの集合🡪 Fitness Vectorの集合

* Standardize/Adjust/Normalize： Fitness Vectorの集合
* Rank/Sort ・・・リニア/パレートスコア
* Environmental Selection （Archive, Mating Pool, Selection）
* Crossover
* Mutation

# データ構造

* Tree
  + IVフィルタをノードとするツリー構造．S式表現と互換性がある．
  + IVGraphとしても使える．（派生クラスにする）
* IVGraph
  + IVフィルタをノードとするグラフ構造．複数のIVTreeを，共通部分を共有する形でマージしたもの．PPEGraphを生成する元となる．グラフなので，S式表現はできない．
* PPEGraph
  + PPEオペレータに相当する単位をノードとするグラフ構造．IVGraphから変換される．Copyオペレータは，IVGraphからの変換時に削除しているため，含まれない．
* PPEScript
  + PPEエンジンが解釈可能な，スクリプト．

設定ファイル

* Begin~Endによるセクション分けと，1行1コマンドによる変数ロードで構成される
* IVグラフ（IVGraphセクション）
  + Vのリストと，Eのリストで表現する．入力時は，登場順に登録される．出力時は，トポロジカルソートにより，Sinkに近い方から出力する．
  + V Sink  
    V Add  
    V Src1  
    V Src2  
    E Sink, Add  
    E Add, Src1  
    E Add, Src2
  + SrcN IVノード
    - Move PPEノードに変換
    - Nをレジスタ番号（color）としてセット
    - IはSI3B型，VはSV1D型
  + SinkN IVノード
    - Move PPEノードに変換
    - Nをレジスタ番号（color）としてセット
    - IはRI1B型，VはRV1D型
  + それ以外のIVノード
    - 辞書に基づいて変換
    - レジスタ番号（color）は-1として，彩色で決定される
    - I,Vの型は辞書に基づいて決定
* PPEグラフ（PPEGraphセクション）
  + Vのリストと，Eのリストで表現する．入力時は，登場順に登録される．出力時は，トポロジカルソートにより，Sinkに近い方から出力する．
  + V SinkI  
    V SinkV  
    V AddI  
    V Src1I  
    V Src1V  
    V Src2I  
    V Src2V  
    E SinkI, AddI  
    E SinkV, Src1V  
    E AddI, Src1I  
    E AddI, Src2I
  + Vは，OpCode，属性を持つ
    - 属性は，下記
    - 0：通常の命令．同じ名前の命令を生成する．Input Edgeが出力レジスタ，Output Edgeが入力レジスタ
    - 1：ソース命令．Move命令を生成する．Input Edgeが出力レジスタ．SrcNのNを読み取った数値をソースアーカイブ番号として入力に使う．
    - 2：シンク命令．Move命令を生成する．SinkNのNを読み取った数値をリザルトアーカイブ番号として出力に使う．Output Edgesが入力レジスタ．

## スクリプト生成

* 複数ツリーを統合すると，複数の出力を持ちたい部分が出てくる．
* Duplicatorを挟んで，複製する．

ビルディングブロック

* Const\_N
  + 定数はソースとして扱う．
  + ビルディングブロックおよびRawFit，FitnessVectorの式をスキャンして，Const\_Nがあれば，ソースに追加する．
  + ソースの追加
    - Configurationは変更できないので，UniversalEngineに持たせる．
    - 内容はSourceArchiveに入れてしまっているから，その後アクセスすることはない．
    - UniversalEngineの初期化時点で，Const\_Nを検索
    - Const\_NをSrcNに変換してしまう．
    - エイリアスに登録しないといけない
    - doubleへの変換は一度しかしないから，Archiveへの書き込み時点でよかろう．
    - 同じConst\_Nが登場する可能性があるので，それはまとめたい．
    - Map＜Const\_N，SrcN＞を作ればいいかな．
    - ビルディングブロックはテンポラリに作り直す必要がある．
    - ビルディングブロックに対してはGrayNとして与える
    - FVectorのSourceは，RawFitなので，RawFitへも追加する必要があるわ

## レジスタのアサイン

* 彩色でまず「論理レジスタ」を割り振る
* Duplicatorでは，複数の論理レジスタが１つの物理レジスタに，マッピングされる．
* Copyでは，出力の論理レジスタは出力の論理レジスタと同じ物理レジスタに，マッピングされる．
* 論理レジスタから物理レジスタへのマッピング
  + データ構造は，各エッジに（エッジレジスタ番号）と，テーブルA[論理レジスタ]=（参照カウント，物理レジスタ），テーブルB[物理レジスタ]=（使用中フラグ）
    - 参照カウントは，出力として使われたときに1増えて，入力として使われたときに１減る
    - テーブルAは，彩色によって得られた個数分，参照カウントは０，物理レジスタは不定で，初期化する．
    - テーブルBは，実行中に個数が変化するものとし，すべて未使用で初期化する．
  + まず，エッジレジスタを，彩色によってマッピングする
  + Duplicatorの，全ての出力のエッジレジスタを，入力のエッジレジスタと同じ値に書き換える．
  + Copyの，出力のエッジレジスタを，入力のエッジレジスタと同じ値に書き換える．
  + ノードについてトポロジカル整列順に，エッジレジスタを物理レジスタに更新していく
    - ノードの出力エッジについて，テーブルA[エッジレジスタ].参照カウント==0ならば，未使用の物理レジスタを１つ割り当て，参照カウントを１増やす．その物理レジスタをエッジレジスタとして使用する．
    - ノードの入力エッジについて，テーブルA[エッジレジスタ].物理レジスタをエッジレジスタとして使用する．テーブルA[エッジレジスタ].参照カウントを1減らし，0になったら物理レジスタを未使用に変更する．
* Intron（およびDuplicatorやCopy）がスクリプトに生成されない方法
  + Intron
    - ルートにつながっていないノードは，ライングラフへの変換で現れないので，レジスタ番号は与えられない．
      * ルート＝Sink＝ResultストレージへのMove．
    - よって，出力レジスタ，入力レジスタいずれかが初期値（-1）の場合は，Intronであると判断できる
    - Sink以外は出力があるので，実際には出力レジスタが-1であることのみ調べればよい．
    - 出力レジスタが決定するのは，彩色後なので，PPEGraphからScriptを生成する時点で削除する．
  + Duplicator，Copy
    - これらは，出力レジスタ，入力レジスタは同じ物理レジスタ番号を持つ
    - よって，出力レジスタと，入力レジスタが，同じ値の場合は，Duplicator，Copyであると判断できる
    - Intron以外は，入力・出力の各レジスタはすべて異なるので，Intronを調べる際もすべての入出力の組を調べる必要はなく，出力の最初のものと入力の最初のものが一致していれば，それで判断してよい．
    - しかし，彩色前にDup/Copyを削除できると，PPEGraph，彩色のコストの大幅な削減になる
    - IVGraphからPPEGraphへの変換時に削除する
* PPEグラフへの変換
  + 頂点生成フェーズ
    - SrcN
      * 頂点と型は， Move(SI3B), Move(SV1D), Move(SA1D)に変換される．
      * 色として，特別な値（ソースアーカイブのレジスタ番号N）を与えておく．
    - TreeN
      * 頂点と型は， Move(SI1B), Move(SV1D), Move(SA1D)に変換される．
      * 色として，特別な値（ソースアーカイブのレジスタ番号N）を与えておく．
    - SinkN/VASinkN
      * 頂点と型は， Move(RI1B), Move(RV1D), Move(RA1D)に変換される．
      * 色として，特別な値（リザルトアーカイブのレジスタ番号N）を与えておく．
    - それ以外
      * 頂点と型は，辞書に従い変換される．
      * 色は，初期値-1を与える．あとの彩色で決定する．
  + エッジ生成フェーズ
    - 入力エッジの先の頂点がMoveだった場合
      * 色が決まっているMoveの場合は，その頂点と接続する
      * 色が決まっていないMoveの場合は，その頂点の1番目の入力エッジを基点として再帰的に実行する．
    - 入力エッジの先の頂点がMove以外だった場合
      * その頂点と接続する．
    - いずれもPPE頂点はまだ接続されていないので，IV頂点から辿る必要がある．
* スクリプトへの変換
  + トポロジカルソート順に頂点を命令として生成する
  + Intronはスキップ
    - 出力レジスタが-1であるもの．（最終結果に結びつかない頂点，またはスキップされたMove）
    - 型がソースアーカイブであるもの．（ソースの疑似Move）
    - 入力がないもの
* 辞書
  + Intron削除処理のため，“Move”は型解決しない名前で登録しておく．
* 入力がゼロのフィルタ
  + SrcN
    - 事前に出力レジスタが決められていて，彩色には関与しない
    - 処理不要なので，命令を生成したくない
    - 彩色後の出力レジスタ＝値有り
  + Zero
    - 事前に出力レジスタは決められていなくて，彩色で決定される
    - 処理必要なので，命令を生成したい
      * ただし，値は固定なので引数不要
    - 彩色後の出力レジスタ＝値有り
  + Constant\_N
    - 事前に出力レジスタは決められていなくて，彩色で決定される
    - 処理必要なので，命令を生成したい
      * ただし，値は決まっていないので引数必要
    - 彩色後の出力レジスタ＝値有り
  + \_SrcN
    - Constant\_NとSrcNを合わせて順序づけされたSrc

RunScript

* スクリプト実行時のデータ出し入れルール
  + 入力データは，ソースアーカイブのSI3B(0)～およびSV1D(0)～に入れる．それぞれSrc0，Src1，・・・に対応する．
  + 計算結果は，リザルトアーカイブのRI1B(0)～およびRV1D(0)～に入れる．それぞれTree0，Tree1，・・・に対応する．

IVツリー：TreeNをルートとするツリー．染色体が複数ある場合はIVツリーも複数ある．個体I,染色体J の場合に，N=I\*10+Jとして計算した，

拡大IVツリー：EvalNをルートとするツリー．IVツリーをサブツリーとする，変換関数の意味合いを持つ．複数個指定できる．１つの拡大IVツリーの中で，複数の染色体の結果を取り込むことができる．

ソースN：ソース番号は，SrcN．1個～5個程度

タスクI：タスクはアーカイブを切り替えることで実現するので，プログラム中には現れない．5個～100個程度

リザルトN：リザルトは，TreeNのNとして格納される．100個程度

Fitness Vector：ベクトル．個々の要素に何を入れるかは，プログラマブルとする．

Fitness Vector

* 個体のFitnessを表すベクトル．
* 各要素は，特別な演算子を使った，拡大IVツリーとして定義する．
  + 要素数1の場合は，これまでのGarnetと同じになる．
  + 演算子の種類は，その個体の演算結果を参照するもの，および全個体の演算結果を参照するもの，の2種類がある．自分以外の特定の個体の演算結果は参照できない．
* 実行方法
  + Fitness Vectorの各要素の結果も，PPEを使って行う．
  + 各染色体の評価が終わったら，その結果をFitness Vector計算用のArchiveにコピーして，Fitness Vector計算用のExecutableを実行する．
  + 実行後，各要素の結果は，RV1D(0)～RV1D(N-1)に格納される．
  + 使える場合は，OpenCLユニットを使う．
* 演算子リスト
  + Chromosome\_n（n=0~）（Src0エイリアス）
    - I: その個体の，n番目の染色体のI成分．
    - V: その個体の，n番目の染色体のV成分．
  + NumOp\_n（Src1エイリアス）
    - I: 黒
    - V: その個体の，n番目の染色体の，IV演算子の総数．
  + Depth\_n（Src2エイリアス）
    - I: 黒
    - V: その個体の，n番目の染色体の，ルートからソースまでの最大深さ．
  + Time\_n（Src3エイリアス）
    - I: 黒
    - V: その個体の，n番目の染色体の，実行時間．
  + AvgNumOp（Src4エイリアス）
    - I: 黒
    - V:全個体の，IV演算子の平均値
  + AvgDepth（Src5エイリアス）
    - I: 黒
    - V: 全個体の，ルートからソースまでの最大深さの平均値
  + AvgTime（Src6エイリアス）
    - I: 黒
    - V: 全個体の，実行時間の平均値
  + PSNR(?)(?)
    - I: 黒
    - V: 2つの入力のPSNR．
  + それ以外の各演算も使える．
* タスク，個体
* アーカイブ＝タスクごと
  + IVツリー
    - 依存関係： 1タスク， 1個体
  + Fitness Vector計算
    - 依存関係：全タスク，全個体
  + IVツリーのコンパイル結果は，全タスクで共通で使える．1個体のみ有効．
  + IVグラフのコンパイル結果は，全タスクで共通に使える．全個体で共通に使える．
  + アーカイブは，1タスクのみ有効．全個体で共通に使える．
* 前考えていたやつ
  + V(A)　フィルタツリーを処理したときのV
  + Nodes(A)　ノード数
  + Depth(A)　深さ
  + Time(A)　実行時間
  + Invert(B)　1-B
  + Nearest(A)　Aと最も近いツリーまでの距離：多様性の維持に
  + Negate(B)　–B
  + Normalize(B)　他のツリーの同じFitnessVector座の中で、最小値、最大値をとり、0~1にノーマライズする
    - raw fitness: r(i, t)計算値
    - standardized fitness: s=f(r) ex.) s=rmax-r　　s>0を保証
    - adjusted fitness: a=1/(1+s) 0~1で大ほど良に統一する
    - normalized fitness: n=a\_i/sum(a) 全体の比率。このままルーレットやトーナメントで使える。
  + Hits
    - HitsはTrue/Falseの値。
    - なんらかの基準に達しかなかった場合は、Hits=Falseとなり、Selectionから即座に漏れる。たとえばタイムアウトやFitnessを計算できる数が十分であるか

# ツリー単体処理モード

* 要件
  + ソースデータ（ツリー，ソースIV）の入力は下記から１つ選べる
    - コマンドラインから入力
      * ツリーの入力は–t XXXX．複数指定可能．前から順に0，1，・・・
      * ソースIVの入力は -0 XXXX,YYY -1 同・・・．引数はクオート推奨．ファイル名だけ（,なし，または,で終わる）の場合は，値は0に初期化．ファイル名なし（,から始まる）の場合は，画像は\_\_BLACK\_\_に初期化．ファイル名に,を含む場合は，値を指定するか最後に,を指定する．（最後の,で切り分けるので）
      * エイリアス辞書を複数指定できる (-a)
      * 画像サイズ (-x, -y)
      * ワーキングディレクトリー (-d)
        + 画像の読み込み，保存の起点として使われる
    - テキストファイルから入力 (-b XXXX, XXXXXは-bの引数ではなくて，プロジェクト名（最後の引数)）
      * -bが指定された場合，-t,-0,-a等は無視される
  + 結果の出力は下記のものを複数選べる
    - 数値の画面出力 (-n)
    - 画像の画面出力(-w)
    - 画像のファイル出力（ファイル名指定あり）(-f XXXX，printf形式)
    - IVのCSV出力（画像ファイル名，数値）（ファイル名指定あり）(-c XXXX)
    - HTML形式のサマリーファイルの生成 (-h XXXX)

# コマンドライン

* モード
  + 以下の優先順位
  + -m XXXX： XXXXモード
  + -t XXXX：ツリーモード
  + -mと-tが同時に指定されていたら，エラー．
  + デフォルト：ガーネットモード
    - その他にモード指定がない場合
  + -T トレース
  + -T
  + -a エイリアスファイル
* モード
  + garnet
  + tree
  + fitnessvector
  + tree2graph
  + rawfitness
  + iv2ppe
  + genscript
  + runscript

# サマリーファイル

* 結果のIとV
  + Iは保存したファイルへのリンク
* HTML形式
* EvaluatorとIVFilterの結果を分けたい
  + EvaluatorのV
  + IVFilterのIとV
  + 今後はFitnessVectorのそれぞれの値も知りたい
* どうやって入力するか
  + Individual = IVFilter
  + Evaluatorは別に入力する　🡪FitnessVector定義を入力する

ストレージ

・データはすべて、1つの物理ストレージに格納

・評価時は、論理ストレージにマッピングして並び替えたモノでアクセスする

* + ストレージレイアウト
    - Constant Nk個
    - Task Nt個
      * Src (Task) Ns個
    - Test Nu個
      * Src (Test) Ns個
    - Task Nt個
      * Individual Ni個
        + Tree (Task) Nc個
    - Test Nu個
      * Individual Ni個
        + Tree (Test) Nc個
    - Task Nt個
      * Individual Ni個
        + RawFit (Task) Nr個
    - Test Nu個
      * Individual Ni個
        + RawFit (Test) Nr個
    - Individual Ni
      * RawFit List (Task) Nr個
      * RawFit List (Test) Nr個
      * Fitness Nf個

# 評価の仕方

## 用語

* ビルディングブロックカテゴリ
  + A：いつでも使えるフィルタ（IVフィルタ）
  + B：GPで生成されたツリーの評価後に使えるフィルタ（IVフィルタ）
  + C：RawFitness評価後に使えるフィルタ（VALフィルタ）

## 定数

* + Ni=個体(individual)数．Populationともいう．
  + Nj=ランタイム個体数．Archive個体との合計であるなど．
  + Nc=染色体(chromosome)数．個体当たりのツリー数．定数．
  + Nt=Task (Training Fitness Case)数．定数．
  + Nu=Test (Test Fitness Case)数．定数．
  + Nr=Raw Fitness次元数．定数．
  + Nf=Fitness Vector次元数．定数．
  + Ns=TaskあたりのSrc数．Constantは含まない．定数。
  + Nk=Constantの数．

## ターミナルノード

* + ユーザー指定のSrcは，TreeとEvaluatorでは使える必要がある．
* \_LoadN（カテゴリA）
  + ストレージアドレスNの値をロードする
* \_StoreN（カテゴリA）
  + ストレージアドレスNに値をストアする
* Constant\_R（カテゴリA）
  + 実数定数を埋め込む場合に使う
  + I ゼロ
  + V 指定された値
  + A 空集合
  + L 空集合
  + ビルディングブロックに定義するとTree生成で使えるほか、いきなりEvaluatorやFVector定義式に入れても自動的にターミナルとして使える
  + ストレージアドレス: k
  + 相対ストレージアドレス: k
* Src<s>（カテゴリA）
  + 人口全体を一度に評価するので~~，タスクごとにストレージを分ける~~．
  + 各個体で同じモノを使う．
  + ~~Treeの格納にも使う．~~
  + I 現在アクティブなタスクtのs番目のデータ
  + V 現在アクティブなタスクtのs番目のデータ
  + A 空集合
  + L 空集合
  + ストレージアドレス: Nk+t(Nk+Ns+NiNc+NiNr) + Nk+s
  + 相対ストレージアドレス: Nk+s
* TaskSrc<s>（カテゴリA）
  + I ゼロ
  + V ゼロ
  + A 空集合
  + L 全タスクのs番目のデータ
  + ストレージアドレス:Nk+[0:t-1](Nk+Ns+NiNc+NiNr)+Nk+s
* Tree<c>（カテゴリB）
  + 現在Activeなタスクtのi番目のTreeの、c番目の染色体の評価結果。
  + I 染色体評価結果
  + V 染色体評価結果
  + A 空集合
  + L 空集合
  + ストレージアドレス:Nk+t(Nk+Ns+NiNc+NiNr)+Nk+Ns+iNc+c
  + 相対ストレージアドレス: Nk + Ns + iNc + c
* RawFit<r>（カテゴリB）
  + 現在Activeなタスクtのi番目のTreeのr番目の中間適合度。
  + I ゼロ
  + V 現在Activeなタスクtのi番目のTreeのr番目の中間適合度
  + A 現在Activeなタスクtのi番目のTreeのr番目の中間適合度
  + L 現在Activeなタスクtのi番目のTreeのr番目の中間適合度
  + ストレージアドレス:Nk+t(Nk+Ns+NiNc+NiNr)+Nk+Ns+NiNc+iNr+r
  + 相対ストレージアドレス: Nk + Ns + NiNc + iNr + r
* TaskRawFit<r>（カテゴリC）
  + 全タスクのi番目のTreeのr番目の適合度。
  + I ゼロ
  + V ゼロ
  + A 全タスクのr番目の中間適合度
  + L 全タスクのr番目の中間適合度
  + ストレージアドレス:Nk+[0:t-1](Nk+Ns+NiNc+NiNr)+Nk+Ns+NiNc +iNr+r
* TestSrc<s>（カテゴリA）
  + I ゼロ
  + V ゼロ
  + A 空集合
  + L 全テストのs番目のデータ
  + ストレージアドレス:Nk+(Nt+[0:u-1])(Nk+Ns+NiNc+NiNr) +Nk+s
* TestRawFit<r>（カテゴリC）
  + 全テストのi番目のTreeのr番目の適合度。
  + このターミナルが使われていない場合は評価されない
  + I ゼロ
  + V ゼロ
  + A 全タスクのr番目の中間適合度
  + L 全テストのr番目の中間適合度
  + ストレージアドレス:Nk+(Nt+[0:u-1])(Nk+Ns+NiNc+NiNr)+Nk+Ns+NiNc+iNr+r

## ルート（シンク）ノード

* Tree<c>（カテゴリA）
  + 現在Activeなタスクtのi番目のTreeの、c番目の染色体の評価結果。
  + IVのみ格納
  + ストレージアドレス:Nk+t(Nk+Ns+NiNc+NiNr)+Nk+Ns+iNc+c
  + 相対ストレージアドレス: Nk + Ns + iNc + c
* RawFit<r>（カテゴリB）
  + 現在Activeなタスクtのi番目のTreeのr番目の適合度。
  + IVAのみ格納
  + ストレージアドレス:Nk+t(Nk+Ns+NiNc+NiNr)+Nk+Ns+NiNc+iNr+r
  + 相対ストレージアドレス: Nk+Ns+NiNc+iNr+r
* Fitness<f>（カテゴリC）
  + i番目のTree のf番目の適合度。
  + Vのみ格納
  + ストレージアドレス:iNf+f
  + 相対ストレージアドレス: f

# 手順

* + ストレージの排他制御を避けるため、全部Touchしておく
  + ステージA、B、Cは並列処理し、AとBもWait For Allではなく、Bが始められるところまで終わったら次に行くように検討する。BからCはWait For All。

## ステージA： Treeの評価

* + S式で記述された，IVペアの処理シーケンス
  + ステージA1
    - Taskの評価
  + ステージA2
    - Testの評価
    - Testターミナルが使用されていない場合は実行されない
  + 評価は並列実行する
    - 全Treeを１つのIVGraphスレッドにまとめる。ただし、ノード数が非常に多い場合は分ける
    - 各Task、各Testは常に別のスレッド
  + ストレージレイアウト
    - Const

## ステージB：RawFitnessの評価

* + ステージB1
    - Taskの評価
  + ステージB2
    - Testの評価
    - Testターミナルが使用されていない場合は実行されない
  + 評価は並列実行する
    - 各RawFitness座は１つのIVGraphにまとめて実行する。
    - 各Treeは常に別のスレッド
    - 各Task、各Testは常に別のスレッド

## ステージC： Fitness Vectorの生成

## Tree

* + S式で記述された，IVペアの処理シーケンス
  + ビルディングブロックにはカテゴリAが使える．
  + 個体の染色体ごとにTreeを持つ．
  + 個体にNc個ずつ，世代合計NiNc個のTreeがある．
  + さらに，タスクの数だけ評価されるので，評価回数は世代合計NiNcNt．
  + 評価は，人口全体を一度に評価するので，Nt回のループ．アーカイブはNtを含まないフォーマットになる．
  + Srcと同じアーカイブを共有する．RawFitness計算でSrc，Tree両方にアクセスしたいので．
  + ストレージアドレス:
    - （Ni, Nc）=アドレスとして，以下のような順でV1DアドレスNに保存される．SinkNのNに該当する．
    - (0,0)=Ns, (0,1)=Ns+1, …, (0, Nc-1)=Ns+Nc-1, (1,0)=Ns+Nc, (1,1)=Ns+Nc+1, …, (1, Nc-1)=Ns+2Nc-1, …, (i, c)=Ns+iNc+c, (Ni-1,0)=Ns+ (Ni-1)Nc, (Ni-1,1)=Ns+(Ni-1)Nc+1,…,(Ni-1,Nc-1)=Ns+NiNc-1

## RawFitness（Evaluator）

* + S式で記述された，IVペアの処理シーケンス．評価関数をつないだ拡大IVフィルターとしてRawFitnessを計算する．
  + 個体ごとに，その個体の染色体Treeを使って，Nr個のRawFitnessを計算する．
  + ビルディングブロックにはカテゴリA，カテゴリBが使える．
  + RawFitnessは個体にNr個ずつ，世代合計NiNr個．
  + デフォルトはNr=1で，（Tree0）．Tree0の結果がそのまま使われる．
  + さらに，タスクの数だけ評価されるので，評価回数は世代合計NiNrNt．
  + Children個体に対して実行されるため，Niではない可能性がある．
  + 評価は，人口全体を一度に評価するので，Nt回のループ．アーカイブはNtを含まないフォーマットになる．
    - (Ni,Nr)=Nとすると，
    - (0,0)=0, (0,1)=1, …, (0, Nr-1)=Nr-1, (1,0)=Nr, (1,1)=Nr+1, …, (1, Nr-1)=2Nr-1, …, (i,r)=iNr+r, (Ni-1,0)=(Ni-1)Nr, (Ni-1,1)=(Ni-1)Nr+1,…,(Ni-1,Nr-1)=NiNr-1

## RawFitnessからFitnessVector用ソースへの変換．

* + RawFitnessはNt個のアーカイブだが，Fitness Vectorに合わせて，Ni個のアーカイブに変換する．
  + その際，平均，最大値，最小値，などの計算も行うが，比較計数演算などに備えて全ての要素は保存する．配列型に入れておく．
  + エリート戦略が有効な場合，通常FitnessVectorはArchive＋Childrenで再計算されるため，RawFitnessはいったん個体に保存する．FitnessVector計算時にふたたびアーカイブに展開する．

## FitnessVectorを構築

### 概要

* + S式で記述された，VAペアの処理シーケンス．配列に変換されたRawFitnessのV成分を使って，FitnessVectorを構築する．
  + 個体ごとに，その個体のRawFitnessを使って，Nf次元のFitness Vectorを計算する．
    - 複数のRawFitnessは区別できる．\_R0, \_R1,..などとして指定する．
    - Nt個の平均，最大値，最小値，Cを超えている数，Cを下回っている数，などの演算がある．
    - 個々のタスクは区別できず，平均値，最大値，最小値，など全タスクの結果全体への操作のみが指定可能．
  + FitnessVectorは個体に1つずつ，世代合計Ni個．要素はNiNf個．
  + FitnessVectorのNf個の要素は一度に評価するので，Ni回のループ．アーカイブはNiを含まないフォーマットになる．
    - (Nf)=Nとすると
    - (0)=0, (1)=1, …, (Nf-1)=Nf-1

### 処理対象

* 個体ごと

### ビルディングブロック

* カテゴリC

### 入力アーカイブ

|  |  |  |  |
| --- | --- | --- | --- |
| Address | Alias | V1D | A1D |
| <n>  <n>=0,…,Nr-1 | Src<n>, | average(RawFitness[0:Nt-1] | RawFitness[0:Nt-1] |
| Nr | Nodes  Src<Nr> | 個体の全ての染色体に含まれる頂点数． | [] |
| Nr+1 | ExecTime  Src<Nr+1> | 実行時間推定値．個体のすべての染色体に含まれる頂点種別と，与えられた時間統計値を使って計算した推定値． | [] |
| Nr+2 | DistCollins  Src<Nr+2> | Solution Spaceにおいて最も近い個体ペアとの距離．対応する染色体座同士のCollinsのTree Kernel値を計算し，その総和．小さいほど遠い． | [] |
| Nr+3 | DistKashima  Src<Nr+3> | Solution Spaceにおいて最も近い個体ペアとの距離．対応する染色体座同士のKashimaのTree Kernel値を計算し，その総和．小さいほど遠い． | [] |
| Nr+4 | DistNSGA2  Src<Nr+4> | Objective Spaceにおいて最も近い個体ペアとの距離．NSGA2のCrowindg Distance Assignmentを使用する．大きいほど遠い． | [] |
|  |  |  |  |

### 出力アーカイブ

|  |  |  |  |
| --- | --- | --- | --- |
| Address | Alias | V1D | A1D |
| <n>  <n>=0,…,Nf-1 | Dist<n> | FitnessVector[n] | [] |
|  |  |  |  |

### デフォルト

* + (RawFit0)．RawFitness[0]のV平均値．

## Rankを計算

* + FitnessVectorの次元が1ならば，数値順のランキング．１～N
  + 2以上ならば，Pareto的ランキング．１～R≦N
* 終了または世代交代

# 処理の流れ

* Tree評価で，タスクごとに1回のPPE．世代ごとに変化する．
  + グラフ：全個体（Tree）を結合したDAG
  + ソースStorage：TaskSource
    - [0]Src0，[1]Src1, …
  + リザルトStorage：TreeResult
    - [0]Tree0, [1]Tree1, …
  + 必要に応じてResult結果を保存
* RawFitness評価で，タスクごとに1回のPPE．変化しない．
  + グラフ：全RawFitness定義（拡大フィルタツリー）を結合したDAG
  + ソースStorage：TreeResult
    - [0]Tree0, [1]Tree1, …
  + リザルトStorage：RawFitness
    - [0]RFit0, [1]RFit1, …
* FitnessVector計算で，個体ごとに1回のPPE．変化しない．
* グラフ：全Fitness要素定義を結合したDAG
* ソースStorage：AllRawFitness．
  + - [0]タスク0のRFit0, [1]タスク1のRFit0, …  
      [N]タスク0のRFit1, [N+1]タスク1のRFit1, …  
      [(Nt-1)N]タスク0のRFit(l-1),[(Nt-1)N+1]タスク1のRFit(l-1),…
* リザルトStorage：FitnessVector
  + - [0]VFit0, [1]VFit1, …

# 引数

* 必要なパラメータ
  + title : タイトル．ディレクトリと拡張子を含まない．拡張子とは，最初のドット以降をすべて取り除いたもの．
  + prj : 設定ファイルのフルパス
  + dir : 作業ディレクトリ．世代ごとの結果が保存されるディレクトリ
* 引数
  + \_Project，\_WorkingDirectory (-d)
  + \_Project，-dなし
    - \_Projectにパスあり，拡張子あり
      * title :\_Projectからパスと拡張子を取り除いたもの
      * dir : \_Projectのパス部分 + title
      * prj : \_Project
    - \_Projectにパスなし，拡張子あり
      * title :\_Projectからパスと拡張子を取り除いたもの
      * dir : title
      * prj : \_Project
    - \_Projectにパスなし，拡張子なし
      * title :\_Projectからパスと拡張子を取り除いたもの
      * dir : title
      * prj : \_Project + “.garnet”
  + \_Project＋\_WorkingDirectory (-d)
    - \_Projectにパスあり，拡張子あり
      * title :\_Projectからパスと拡張子を取り除いたもの
      * dir : \_WorkingDirectory
      * prj : \_Project
    - \_Projectにパスなし，拡張子あり
      * title :\_Projectからパスと拡張子を取り除いたもの
      * dir : \_WorkingDirectory
      * prj : \_Project
    - \_Projectにパスなし，拡張子なし
      * title :\_Projectからパスと拡張子を取り除いたもの
      * dir : \_WorkingDirectory
      * prj : \_Project + “.garnet”

評価用演算子

* Nodes：ツリーのノード数
  + カテゴリC，ターミナル．
  + グラフの頂点数を数える．個体に含まれる全ての染色体の総和．
* DistCollins，DistKashima：最近傍ツリーとの距離
  + カテゴリC，ターミナル．
  + 同じ染色体座（Chromosomal Locus）同士の距離を測って，すべての染色体の距離の総和．
  + 距離の計算には，Tree Kernel法を使う．DistCollinsはCollinsによる定義，DistKashimaは鹿島による定義．
* ExecTime：ツリーの実行時間
  + カテゴリC，ターミナル．
  + 複数のツリーがアグリゲートされて実行されるため，実際の実行時間は取得できないため，グラフの頂点リストと，ExecutionUnitの実行時間統計（各演算子の平均時間）を使って，推定値を求める．個体に含まれる全ての染色体の総和．

RawFitnes定義のコンパイル

* Evaluatorは学習全体を通じて変化しないので，予めコンパイルする．
* 定義に使えるターミナル（葉ノード）はSrcNとTreeN
* SrcNは全個体で共通．TreeNは自分のもののみアクセスできる．
* よって，木構造は個体ごとに変化ないが，TreeNの葉が違うので，個体の数だけ別々なものになる．
* 個体i, i=0,1,..のターミナルの実際のアドレスは，SrcN=N, TreeN=Ns+iNr+N
* 定義の各座は，SinkNに保存される．個体iのSinkNの実際のアドレスは，SinkN=iNr+N

Fitness定義のコンパイル．

* IV用のSinkフィルターは，IとVしか生成しない．
* 一方，Fitness用は，VASinkフィルタを用いて，VとAしか生成しないことにする．
* 厳密には，PPEGaphへのクラックの際に，頂点はI,V,Aそれぞれ生成するが，Sink頂点からのエッジを不要なものについては引かない．入力のない（且つ出力型がレジスタ以外の）エッジはスクリプト化の時にIntronとして削除されるので，それを利用する．
* 辞書はIVAになる．
* IVTree，IVGraphは変化なし．
* PPEGaphは頂点数が増えるが，処理の仕組みに変化はない．
* スクリプト化は影響なし．
* PPEはすでにAへの対応を終えているのでOK．
* 辞書と，IVGraphToPPEGraphを，慎重に拡張すればOK．

スクリプト生成

* 実行順序はトポロジカルソート順
* ライングラフに変換してレジスタ割り付けをする
  + PPEグラフのエッジが，ライングラフの頂点になる．
  + PPEグラフの頂点（オペレータ）実行時点で，同時に存在しているエッジ（ライングラフの頂点）について，ライングラフのエッジでつなぐ（依存関係がある）．同時に存在というのは，演算に直接関係あるものだけでなく，木の別の部分で処理された結果がまだ使用されておらずに残っている場合も含む．

GP処理

ランキング

単純ランキング

Paretoランキング

サマリーファイル

* Fitnessサマリー (<title>.csv)
  + プロジェクトに１つ
  + CSV
    - Generation 0が初期世代．
    - Best Fitness
    - Average Fitness
    - Worst Fitness
    - Max Tree Size (in # of Nodes)
    - Average Tree Size (in # of Nodes)
    - Min Tree Size (in # of Nodes)
    - Seconds
  + 書き込み条件（以下のOR）
    - 100世代経過
    - Best Fitness変化
    - 1分経過
    - 終了条件マッチ
    - ※書き込み条件にマッチした場合，書き込み時点の世代までの書き込まれていないデータ全てを書き込む．
* Generationログ (<title>.000000.txt)
  + 世代に１つ
  + テキスト（ConfigLoader互換）
    - Generation N 第N世代. N=0が初期世代．
    - RandomSeed N 乱数シード
    - Rank N ランク
    - Chromosomes N, .. 染色体S式
    - FitnessV N, .. Fitnessベクトル．Doubleのビット表現16進表示
    - Rank, Chromosomes, FitnessVは３つで1セットとして，個体数分繰り返す．セットNが個体番号Nになる．通常はRankでソートされている．Rankが同じ個体同士の順序を維持する必要に注意（再開時に選択への影響が変わらないように）
  + 書き込み条件（以下のOR）
    - 100世代経過
    - Best Fitness変化
    - 1分経過
    - 終了条件マッチ
    - ※書き込み条件にマッチした世代のみ保存される．
* 自動保存チェックポイント (.garnet\_cp.txt)
  + 世代に1つ
  + 内容はGenerationログと同じ
  + 書き込み条件も同じ
  + 最後に保存されたGenerationログだと思えばいい



初期世代

Initializing Generation

Generating … Done (ランダムに生成) (\*この行は下記のバリエーション有り)

Evaluating … (タスク×個体の数だけ.を表示) Done (この行はロードの時はない)

Saving … Done (この行はロードの時はない)

Saving Summary … Done (この行はロードの時はない)

\*のバリエーション

Loading Generation #N … Done (指定世代からロードして再開の場合)

Loading Auto-Saved Check Point … Done (Generation #N) (チェックポイントからロードして再開の場合)

Generation #N

Reproducing … Done

Evaluating … (タスク×個体の数だけ.を表示) Done

Saving Log… Done (この行はログ保存条件に合致したときだけ

Saving Summary … Done (この行はログ保存条件に合致したときだけ

Best Individual: (xx, yy, zz) (Rank最上位の個体のFitnessVectorを表示)

Time: xx.xxx [sec]

Number of generations (3000).

Finished. Xx.xxx [sec]

開始

* 世代が指定されていて，かつ世代0の場合は初期世代からスタート
* 世代が指定されていて，かつ1以上の場合はそれをロードしてスタート．ファイルがない場合はエラー終了．
* チェックポイントファイルがあればそこから再開
* 上記のいずれもマッチしない場合は初期世代からスタート

終了条件チェック

* チェック自体は下請けを使うので，表示も

Alias

* コンフィグレーションファイルのUnitはBuildingBlock
* Aliasは別途定義しておく必要がある．

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | NGSA-II |  |  |  |  |  |
|  | Population | Children |  |  |  |  |
| Init | P0={random} |  |  |  |  |  |
| Eval | Pt |  |  |  |  |  |
| Mating |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |

Population

Geneticプロセス中の，Selection, Mating, Cross overの入力は，現世代個体のリストと，そのインデックスのリスト，の組で指定することにする．冗長なTreeの生成を避けるため．

Evaluation

Configurationではなく，与えられたパラメータを参照しないといけない．

理由は，Archiveが与えられたのか，Childrenが与えられたのか，下請けでは先入観を持ってはいけないから．

# 遺伝的アルゴリズム

## Overview

GPの具体的なアルゴリズムには様々なバリエーションがあるが，Garnetでは以下の構造に一般化しており，その枠内で各GPを実装している．各ステージは，用意された組み合わせのほか，個々のステージを別のアルゴリズムに差し替えることも出来る．

### 1.Selection

現世代から，エリートとしてそのまま残る個体（Archive）と次世代の親になる個体（Selection）を選ぶ．

### 2.Mating

Selectionから，両親ペア（Parents）を選ぶ．

### 3.Crossover

両親ペアから，子（Children）を生成する．

### 4.Mutation

子に対して，突然変異を適用する．

### 5.Evaluation

子（突然変異適用済み）について，RawFitnessを計算する

### 6.Aggregation/FitnessVector

ArchiveとChildrenを合わせた個体群（Candidates）について，FitnessVectorを生成する．

### 7.Ranking/Sorting/Truncation

Candidates（FitnessVector生成済み）について，ランク付け，ソーティングを行う．また，必要ならば個体数を調整する．

### 0.Initial Generation

初期世代の生成．

## Basic GP

動作試験用，Single Objectiveの，基本的なGP．

### 1.Selection

Roulette + Elitism  
(BasicStrategy::RoulettetSelector)

### 2.Mating

Random Mating   
(BasicStrategy::RandomMating)

### 3.Crossover

Tree 1X   
(BasicStrategy::OnePointCrossover)

### 4.Mutation

Uniform Tree Mutation (BasicStrategy::UniformMutation)

### 5.Evaluation

IVFilter Evaluation (CommonStrategy::Evaluator)

### 6.Aggregation/FitnessVector

Fitness Vector Evaluation (CommonStrategy::FitnessVectorGenerator)

Crowding Distanceには，オリジナル（DistNSGA2）のほか，TreeKernelベースのDistCollinsとDistKashimaを使用可能．

### 7.Ranking/Sorting/Truncation

Single Objective Sort (BasicStrategy::SingleObjectiveSorter)

### 0.Initial Generation

ランダム

## NSGA-II based

### 1.Selection

Binary Tournament  
(BasicStrategy::BinaryTournamentSelector)

### 2.Mating

Random Mating (BasicStrategy::RandomMating)

### 3.Crossover

Tree 1X (BasicStrategy::OnePointCrossover)

### 4.Mutation

Uniform Tree Mutation (BasicStrategy::UniformMutation)

### 5.Evaluation

IVFilter Evaluation (CommonStrategy::Evaluator)

### 6.Aggregation/FitnessVector

Fitness Vector Evaluation (CommonStrategy::FitnessVectorGenerator)

Crowding Distanceには，オリジナル（DistNSGA2）のほか，TreeKernelベースのDistCollinsとDistKashimaを使用可能．

### 7.Ranking/Sorting/Truncation

Fast Non-Dominated Sort (NSGA2::Sorter)

### 0.Initial Generation

ランダム

## SPEA2 based

### 1.Selection

### 2.Mating

### 3.Crossover

### 4.Mutation

### 5.Evaluation

### 6.Aggregation/FitnessVector

### 7.Ranking/Sorting/Truncation

### 0.Initial Generation

サマリー画像保存

* BestIndividuals（RankトップのIndividual．ParetoOptimizationの場合は，通常複数になる）について，保存する
* 評価時の個体番号と，ソート後の個体番号は違うので，Rankトップが評価時にどの個体番号だったかを知っておく必要がある．（「履歴」を参照）

履歴

* 個体の履歴を取っておく
  + 親：前世代のソート後個体番号．個体ごとに２個ずつある．１６ビット整数値．
  + 交叉：各親の交叉点となった頂点番号．頂点番号は深さ優先で決定しているので，S式でノードを前から数えたときの値と一致する．染色体ごとに２個ずつある．１６ビット整数値．
  + 突然変異：突然変異した頂点番号．染色体ごとに１個ずつある．１６ビット整数値．
  + 評価時個体番号：評価時点での個体番号．個体ごとに１個ずつある．１６ビット整数値．
  + ランク後個体番号：ソート／ランク後の個体番号．陽には保存されず，配列の１で判断する．１６ビット整数値．

Intron

VASinkを使うと，Result StorageへのMoveについて，ValueとArrayだけしか生成されない．Imageにつながるノードは，出力先なしになってIntron化する．

IVSinkを使うと，ImageとValueのみ

トレースレベル

* 0 （常に実行）
  + スクリプトの命令数（サマリーファイルに）
  + スクリプトのレジスタ数（サマリーファイルに）
  + 頂点数（世代ファイルに）
* １ LOCATION
  + 現在の処理と実行時間
* ２ SCRIPT
  + スクリプトのファイルへの保存
    - 毎世代を別のファイルに保存．
    - トレース用のテキストストリームを開いておく
* ３ IMAGE
  + （要検討）ストレージ画像のファイルへの保存
    - 毎世代を別のファイルに保存．
    - Imageはアーカイブごとに1枚のPNGに合成して保存する．ファイル名は，traceI.000000.png
    - Valueは，CSVに保存する．traceV.000000.csv
    - Arrayは，CSVに保存する．traceA.000000.csv
* ４PROCESS
  + IVGraph
  + PPEGaph
  + ライングラフ
  + 彩色結果
  + 交叉の途中経過
* -Tのみの場合，LOCATIONトレースが設定される
* -T –Tと複数指定された場合，１🡪２🡪とトレースレベルが上がり，それ以下のすべてのフラグがセットされる．
* -T 1001 などと引数を与えた場合，右から順にLOCATION，SCRIPT，IMAGE，PROCESSのON(1)/OFF(0)を個別に設定できる

改善前

Generation 21

Reproducing 11

[Garnet::generateScript] # of vertices: 7569

[Garnet::generateScript] # of edges: 12018412

[Garnet::generateScript] # of colors: 143

[Garnet::generateScript] max # of degree: 75648.

[Garnet::generateScript] # of vertices: 7622

[Garnet::generateScript] # of edges: 8806306

[Garnet::generateScript] # of colors: 140

[Garnet::generateScript] max # of degree: 61991.

[Garnet::generateScript] # of vertices: 7739

[Garnet::generateScript] # of edges: 10341539

[Garnet::generateScript] # of colors: 126

[Garnet::generateScript] max # of degree: 111476.

[Garnet::generateScript] # of vertices: 7580

[Garnet::generateScript] # of edges: 12012168

[Garnet::generateScript] # of colors: 163

[Garnet::generateScript] max # of degree: 141316.

[Garnet::generateScript] # of vertices: 4295

[Garnet::generateScript] # of edges: 8458044

[Garnet::generateScript] # of colors: 139

[Garnet::generateScript] max # of degree: 82456

重複されて登録されているエッジがあったのを解消

Generation 21

Reproducing 11

[Garnet::generateScript] # of vertices: 7569

[Garnet::generateScript] # of edges: 336551

[Garnet::generateScript] # of colors: 143

[Garnet::generateScript] max # of degree: 863.

[Garnet::generateScript] # of vertices: 7622

[Garnet::generateScript] # of edges: 288595

[Garnet::generateScript] # of colors: 140

[Garnet::generateScript] max # of degree: 923.

[Garnet::generateScript] # of vertices: 7739

[Garnet::generateScript] # of edges: 305430

[Garnet::generateScript] # of colors: 126

[Garnet::generateScript] max # of degree: 1441.

[Garnet::generateScript] # of vertices: 7580

[Garnet::generateScript] # of edges: 317546

[Garnet::generateScript] # of colors: 163

[Garnet::generateScript] max # of degree: 1651.

[Garnet::generateScript] # of vertices: 4295

[Garnet::generateScript] # of edges: 215199

[Garnet::generateScript] # of colors: 139

[Garnet::generateScript] max # of degree: 1021

深いサブツリー優先

Generation 21

Reproducing 11

Garnet::generateScript 291 #v:9228

[Garnet::generateScript] # of vertices: 7569

[Garnet::generateScript] # of edges: 133455

[Garnet::generateScript] # of colors: 55

[Garnet::generateScript] max # of degree: 923.

Garnet::generateScript 291 #v:9225

[Garnet::generateScript] # of vertices: 7622

[Garnet::generateScript] # of edges: 128242

[Garnet::generateScript] # of colors: 52

[Garnet::generateScript] max # of degree: 952.

Garnet::generateScript 291 #v:9471

[Garnet::generateScript] # of vertices: 7739

[Garnet::generateScript] # of edges: 111234

[Garnet::generateScript] # of colors: 57

[Garnet::generateScript] max # of degree: 1475.

Garnet::generateScript 291 #v:9222

[Garnet::generateScript] # of vertices: 7580

[Garnet::generateScript] # of edges: 120086

[Garnet::generateScript] # of colors: 55

[Garnet::generateScript] max # of degree: 1698.

Garnet::generateScript 291 #v:5190

[Garnet::generateScript] # of vertices: 4295

[Garnet::generateScript] # of edges: 75207

[Garnet::generateScript] # of colors: 52

[Garnet::generateScript] max # of degree: 1073

＜Evaluator, Fitness Vector＞

１のみ

Garnet::generateScript] # of vertices: 2200

Garnet::generateScript] # of edges: 2400

Garnet::generateScript] # of colors: 4

Garnet::generateScript] max # of degree: 8

Garnet::generateScript] # of vertices: 48

Garnet::generateScript] # of edges: 171

Garnet::generateScript] # of colors: 9

Garnet::generateScript] max # of degree: 23

１＆２

[Garnet::generateScript] # of vertices: 2200

[Garnet::generateScript] # of edges: 2300

[Garnet::generateScript] # of colors: 4

[Garnet::generateScript] max # of degree: 8

[Garnet::generateScript] # of vertices: 48

[Garnet::generateScript] # of edges: 109

[Garnet::generateScript] # of colors: 10

[Garnet::generateScript] max # of degree: 12

PPEGaphの頂点数（5グループに分割されている）

#v:9228

#v:9225

#v:9471

#v:9222

#v:5190

トレース

-T script

スクリプトを表示する

-T script>file

スクリプトをfileというファイルに保存する。<T><I><C><G>が使える

-T executable

実行形式のダンプを表示する

-T executable>file

実行形式のダンプをfileというファイルに保存する。<T><I><C><G>が使える

マルチ染色体のときに、全染色体に交叉や突然変異を適用するとどれかがよくてもどれかが悪くなるという状況を引き起こすので、適用する染色体もランダムに選ぶことにする。

Crossover