8. Metaheuristics

- Introduction
- Combinatorial Problems
- Search Methods
- Local Search Algorithms

Introduction

Stochastic search is the method of choice for solving many hard combinatorial problems.

Recent Progress & Successes :

- Ability of solving hard combinatorial problems has increased significantly
  - Solution of large propositional satisfiability problems
  - Solution of large traveling salesman problems
- Good results in new application areas

Introduction

Reasons & Driving Forces :

- New algorithmic ideas
  - Nature inspired algorithms
  - New randomized schemes
  - Hybrid and mixed search strategies
- Increased flexibility and robustness
- Improved understanding of algorithmic behaviour
- Sophisticated data structures
- Significantly improved software

Combinatorial Problems

- Involve finding a grouping, ordering, or assignment of a discrete set of objects which satisfies certain constraints.
- Arise in many domains of computer science and various application areas.
- Have high computational complexity (NP-hard).
- Are solved in practice by searching an exponentially large space of candidate/partial solutions.

Combinatorial Problems

Examples :

- CSPs
- find shortest/cheapest round trips (TSP)
- finding models of propositional formulas (SAT)
- planning, scheduling, time-tabling
- resource allocation
- . . . etc.

Combinatorial Problems

Combinatorial Decision Problems

- For a given problem instance, decide whether a solution (grouping, ordering or assignment) exists which satisfies the given constraints

Combinatorial Optimization Problem

- For a given problem instance, find a solution (grouping, ordering or assignment) with **maximal** (or **minimal**) value of the objective function. The objective function is an evaluation function that assigns a numerical value to each candidate solution.

The Propositional Satisfiability Problem (SAT)

Simple SAT instance (in CNF) :

$$(X_1 \vee X_2) \wedge (\neg X_1 \vee \neg X_2)$$

Possible solutions :

1. $X_1 = true, X_2 = false$
2. $X_1 = false, X_2 = true$

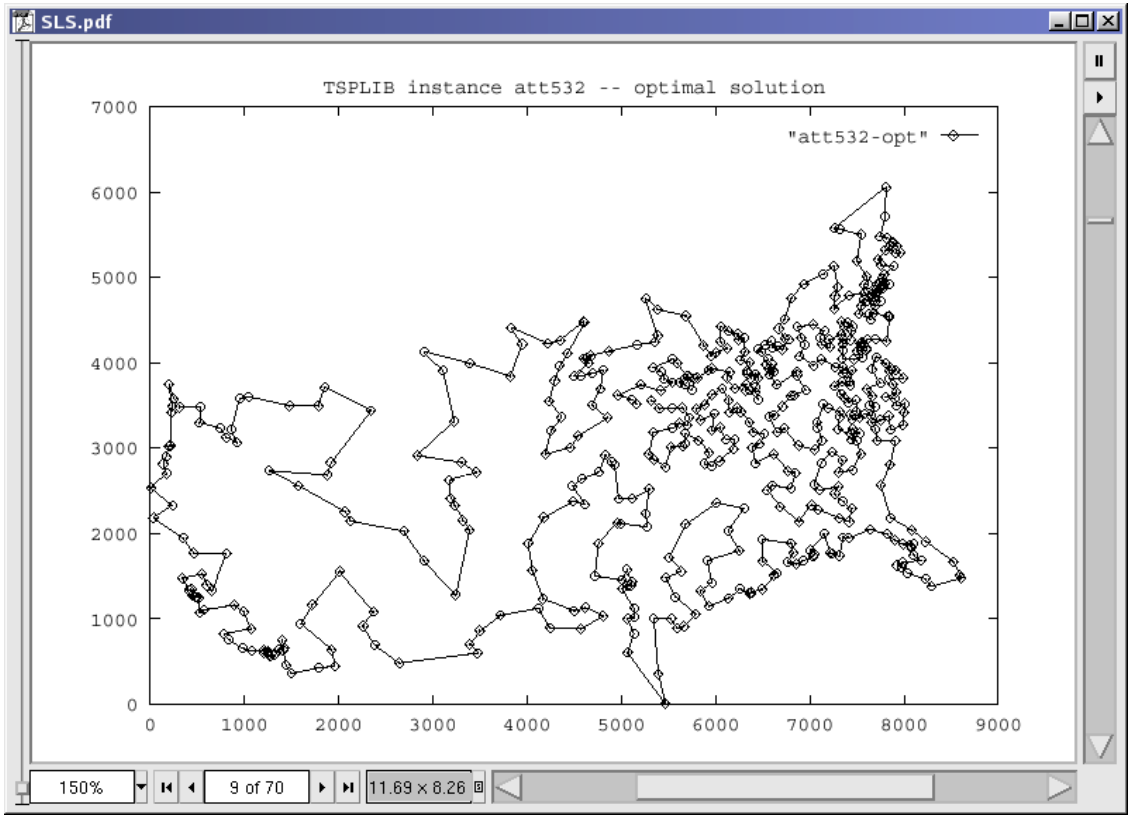
The Propositional Satisfiability Problem (SAT)

- SAT is a pervasive problem in computer science (Theory, AI, Hardware, . . .)
- SAT is computationally hard (NP-hard)
- SAT can encode many other combinatorial problems (NP completeness)
- SAT is one of the conceptually simplest combinatorial decision problems
  - facilitates the development and evaluation of algorithmic ideas

The Traveling Salesperson Problem (TSP)

- **TSP - optimization variant :**
  - For a given weighted graph $G = (V, E, w)$, find a Hamiltonian cycle in G with minimal weight (i.e find the shortest round-trip visiting each vertex exactly once).
- **TSP - decision variant :**
  - For a given weighted graph $G = (V, E, w)$, decide whether a Hamiltonian cycle with minimal weight $\leq b$ exists in G.

TSP instance : shortest round trip through 532 US cities



The Traveling Salesperson Problem (TSP)

- TSP is one of the most prominent and widely studied combinatorial optimization problems in computer science and operations research
- TSP is computationally hard (NP-hard)
- TSP is one of the conceptually simplest combinatorial optimization problems
  - facilitates the development and evaluation of algorithmic ideas

Generalization of combinatorial problems

- Many combinatorial decision problems naturally generalize to optimization problems (SAT to MAX-SAT, CSP to MAX-CSP).
- Many combinatorial problems have practically relevant dynamic variants (dynamic SAT, dynamic CSP, dynamic TSP, Internet routing, dynamic scheduling).
- Often, algorithms for decision problems can be generalized to optimization and/or dynamic variants.
- Typically, good solutions to generalized problems require additional heuristics.

CSP versus MAX-CSP

- A **Constraint Satisfaction Problem** (CSP) consists of:
  - a set of variables $X = \{x_1, \ldots, x_n\}$,
  - for each variable x_i, a finite set D_i of possible values (its domain),
  - and a set of constraints restricting the values that the variables can simultaneously take.
- A **solution to a CSP** is an assignment of a value from its domain to every variable, in such a way that every constraint is satisfied. We may want to find:
  - just one solution, with no preference as to which one,
  - all solutions,
  - an optimal, or at least a good solution, given some objective function defined in terms of some or all of the variables.
- The **Maximal Constraint Satisfaction Problem** is an optimization problem consisting of looking for an assignment that satisfies the maximal number of constraints.

Search Methods

Types of search methods :

- Systematic versus Local Search
- Deterministic versus Stochastic
- Sequential versus Parallel

Properties of search methods :

- Decision problems : complete versus incomplete
- Optimization problems : exact versus approximate

Systematic Backtrack Search

- Iteratively construct candidate solution
- Use backtracking to explore the full search space
- Use appropriate techniques (constraint propagation, branch and bound, . . .) for pruning search space.

Local Search

- Start from initial position.
- Iteratively move from current position to neighboring position.
- Uses objective function for guidance.

Two main classes :

- Local search on partial solutions
- Local search on complete solutions

Solving CSPs and MAX-CSPs

- CSP
  - Exact methods : constraint propagation (local consistency) + backtrack search.
- MAX-CSP
  - Exact methods : Partial Constraint Satisfaction Techniques based on the branch and bound algorithm.
  - Approximation methods : Local search
    - Hill-Climbing
    - Min-Conflicts
    - Min-Conflicts-Random-Walk
    - Steepest-Descent-Random-Walk
    - Tabu-Search
    - GSAT

Partial Constraint Satisfaction Techniques

- Perform Direct Arc Consistency Algorithms in the pre-processing phase.
  - Count the number of inconsistencies counts associated with each variable value.
- Backtrack-Search based on Branch and Bound algorithm.
  - The cost function corresponds to the number of violated constraints.

Local Search Algorithms

- Hill-Climbing
- Min-Conflicts and Min-Conflicts-Random-Walk (MCRW)
- Steepest-Descent-Random-Walk (SDRW)
- Tabu-Search (TS)
- GSAT
- Genetic Algorithms

Local Search

- Local search algorithms are based on common idea known under the notion local search.
- In local search, an initial configuration (valuation of variables) is generated and the algorithm moves from the current configuration to a neighborhood configurations until a solution (decision problems) or a good solution (optimization problems) has been found or the resources available are exhausted.

```
procedure local-search(Max_Moves,Max_Iter)
  s <- random valuation of variables;
  for i:=1 to Max_Tries while Gcondition do
    for j:=1 to Max_Iter while Lcondition do
      if eval(s)=0 then
        return s
      endif;
      select n in neighborhood(s);
      if acceptable(n) then
        s <- n
      end if
    endfor
    s <- restartState(s);
  endfor
  return s
end local-search
```

Hill-Climbing

```
procedure hill-climbing(Max_Flips)
   restart: s <- random valuation of variables;
   for j:=1 to Max_Flips do
     if eval(s)=0 then return s endif;
     if s is a strict local minimum then
         goto restart
     else
       s <- neighborhood with smallest evaluation value
     endif
    endfor
    goto restart
   end hill-climbing
```

Hill-Climbing

Disadvantage of the method

- The hill-climbing algorithm has to explore all neighbors of the current state before choosing the move. This can take a lot of time.

Min-Conflicts

- To avoid exploring all neighbors of the current state some heuristics were proposed to find a next move.
- Min-conflicts heuristics chooses randomly any conflicting variable, i.e., the variable that is involved in any unsatisfied constraint, and then picks a value which minimizes the number of violated constraints (break ties randomly). If no such value exists, it picks randomly one value that does not increase the number of violated constraints (the current value of the variable is picked only if all the other values increase the number of violated constraints).

Min-Conflicts

```
procedure MC(Max_Moves)
 s <- random valuation of variables;
 nb_moves <- 0;
 while eval(s)>0 & nb_moves<Max_Moves do
   choose randomly a variable V in conflict;
   choose a value v' that minimizes the
           number of conflicts for V;
   if v' # current value of V then
     assign v' to V;
     nb_moves <- nb_moves+1;
   endif
 endwhile
  return s
end MC
```

Min-Conflicts

Disadvantage of the method

- The pure min-conflicts algorithm presented previously is not able to leave local-minimum. In addition, if the algorithm achieves a strict local-minimum it does not perform any move at all and, consequently, it does not terminate.

Min-Conflicts-Random-Walk

- Because the pure min-conflicts algorithm cannot go beyond a local-minimum, some noise strategies were introduced in MC. Among them, the random-walk strategy becomes one of the most popular.
- For a given conflicting variable, the random-walk strategy picks randomly a value with probability p, and apply the MC heuristic with probability 1-p.

```
procedure MCRW(Max_Moves,p)
  s <- random valuation of variables;
  nb_moves <- 0;
  while eval(s)>0 & nb_moves<Max_Moves do
    if probability p verified then
     choose randomly a variable V in conflict;
     choose randomly a value v' for V;
    else
     choose randomly a variable V in conflict;
     choose a value v' that minimizes the number of conflicts for V;
    endif
    if v' # current value of V then
      assign v' to V;
      nb_moves <- nb_moves+1;
    endif
  endwhile
  return s
end MCRW
```

Min-Conflicts-Random-Walk

This algorithm is controlled by the random probability p, it should be clear that the value for this parameter has a big influence on the performance of the algorithm. The preliminary studies determined the following feasible ranges of parameter values $0.02 \leq p \leq 0.1$.

Steepest-Descent-Random-Walk

- Steepest-Descent algorithm is a hill-climbing version of the min-conflicts algorithm. Instead of selecting the variable in conflict randomly, this algorithm explores the whole neighborhood of the current configuration and selects the best neighbor according to the evaluation value.
- Again, the algorithm can be randomized by using random-walk strategy to avoid getting stuck at ”local optima”.

Steepest-Descent-Random-Walk

```
procedure SDRW(Max_Moves,p)
  s <- random valuation of variables;
  nb_moves <- 0;
  while eval(s)>0 & nb_moves<Max_Moves do
    if probability p verified then
      choose randomly a variable V in conflict;
      choose randomly a value v' for V;
    else
      choose a move <V,v'> with the best performance
    endif
    if v' # current value of V then
      assign v' to V;
      nb_moves <- nb_moves+1;
    endif
  endwhile
  return s
end SDRW
```

Tabu-Search

- Tabu search (TS) is another method to avoid cycling and getting trapped in local minimum. It is based on the notion of Tabu list, that is a special short term memory that maintains a selective history, composed of previously encountered configurations or more generally pertinent attributes of such configurations.
- A simple TS strategy consist in preventing configurations of Tabu list from being recognized for the next k iterations (k, called Tabu tenue, is the size of Tabu list).
- Such a strategy prevents Tabu from being trapped in short term cycling and allows the search process to go beyond local optima.
- Tabu restrictions may be overridden under certain conditions, called aspiration criteria.
- Aspiration criteria define rules that govern whether next configuration is considered as a possible move even it is Tabu. One widely used aspiration criterion consists of removing a Tabu classification from a move when the move leads to a solution better than that obtained so far.

Tabu-Search

```
procedure tabu-search(Max_Iter)
  s <- random valuation of variables;
  nb_iter <- 0;
  initialize randomly the tabu list;
  while eval(s)>0 & nb_iter<Max_Iter do
   choose a move <V,v'> with the best performance among the non-
    tabu  moves  and the moves satisfying the aspiration criteria;
    introduce <V,v> in the tabu list,
              where v is the current values of V;
    remove the oldest move from the tabu list;
    assign v' to V;
    nb_iter <- nb_iter+1;
  endwhile
  return s
end tabu-search
```

Tabu-Search

- The performance of Tabu Search is greatly influenced by the size of Tabu list tl. A preliminary studies determined the following feasible range of parameter values $10 \leq tl \leq 35$.

GSAT

- GSAT is a greedy local search procedure for satisfying logic formulas in a conjunctive normal form (CNF). Such problems are called SAT or k-SAT (k is a number of literals in each clause of the formula) and are known to be NP-c (each NP-hard problem can be transformed to NP-complex problem).
- The procedure starts with an arbitrary instantiation of the problem variables and offers to reach the highest satisfaction degree by succession of small transformations called repairs or flips (flipping a variable is changing its value).

GSAT

```
procedure GSAT(A,Max_Tries,Max_Flips)
  A: is a CNF formula
  for i:=1 to Max_Tries do
    S <- instantiation of variables
    for j:=1 to Max_Iter do
      if A satisfiable by S then
        return S
      endif
      V <- the variable whose flip yield the most important
       raise in the number of satisfied clauses;
      S <- S with V flipped;
   endfor
  endfor
  return the best instantiation found
end GSAT
```

GSAT

- Several heuristics have been implemented within GSAT in order to efficiently solve structured problems.

  Random-Walk The implementation of random walk within the GSAT algorithm is similar to MCRW.

  Clause weight This technique results from the observation that for some problems, several resolution attempts reach the same unsatisfied final set of clauses. So, each clause has not the same weight on the resolution, some clauses will be much harder to solve. The resolution process must offer more importance to these ”hard” clauses. A way to deal with this kind of problems is to associate a weight to each clause, in order to modify its influence on the global score. Thanks to this weight heuristic, the participation of a satisfied ”hard” clause is more important. Furthermore the weight can be automatically found using the following method:

  1. initialize each clause weight to ’1’
  2. at the end of each try, add ’1’ to each unsatisfied clause weight.

  Averaging in previous near solutions After each attempt, GSAT restarts with a random initial problem variables. This heuristic offers to reuse parts of the best assignments issued from the two previous states. Therefore, the starting variables vector for i-th attempt is computed from the bitwise$(*)$ of the two best reached states during the attempts (i-2)-th and (i-1)-th.

$(*)$: identical bits representing identical variable are reused, the other ones are randomly chosen.

Evolutionary Algorithms

Combinatorial search technique inspired by the evolution of species

- population of strings which are manipulated via evolutionary operators and compete for survival
- population is manipulated via *evolutionary operators*
  - mutation
  - crossover
  - selection

Evolutionary Algorithms

- Several variants have been developed
  - Genetic algorithms
  - Evolution strategies
  - Evolutionary programming
  - Genetic programming
- for combinatorial optimization the most widely used and most effective variant are genetic algorithms

Genetic algorithms

Basic notions :

- Individual and random Individual.
- Population.
- Fitness (evaluation) function.
- Mutation.
- Crossover.

Genetic algorithms

1. **begin**
2. $t \leftarrow 1$
3. // $P(t)$ denotes the population containing the current solution
4. $eval \leftarrow \ evaluate\ P(t)$
5. **while** termination condition is not satisfied **do**
6. $t \leftarrow t + 1$
7. select $P(t)$ from $P(t-1)$
8. alter $P(t)$
9. evaluate $P(t)$
10. **endwhile**
11. **if** solution found **then**
12. return P(t)
13. **end**

GA application to SAT

GA application to SAT

- solution representation : a binary string
- evaluation function : no of violated clauses
- mutation : with a fixed probability flip a variable's truth value
- selection : choose best p strings for the next population avoiding duplicate solutions
- additional local search: after each crossover or mutation apply a 1-opt local search

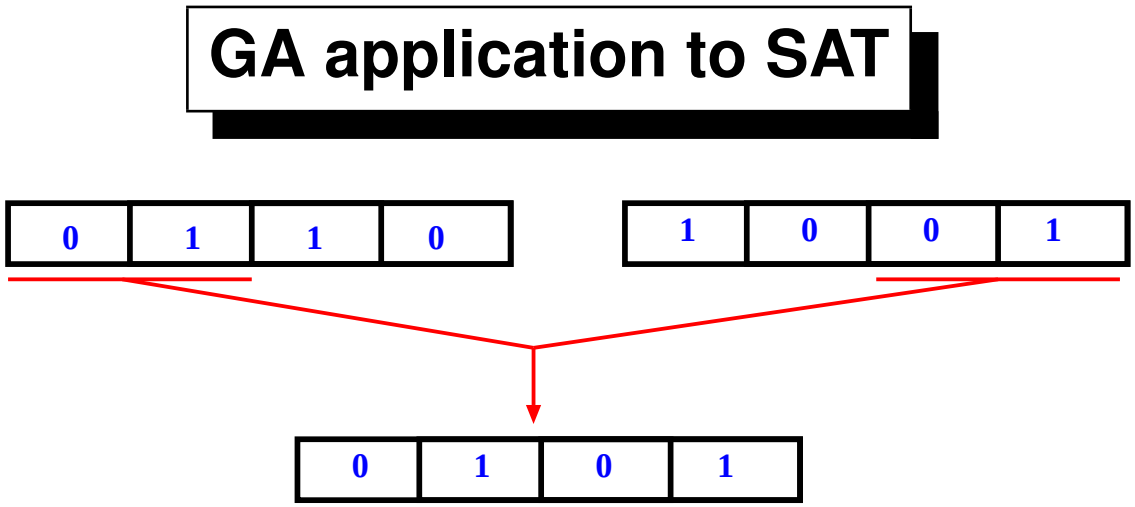


Figure 1: Crossover operator for SAT

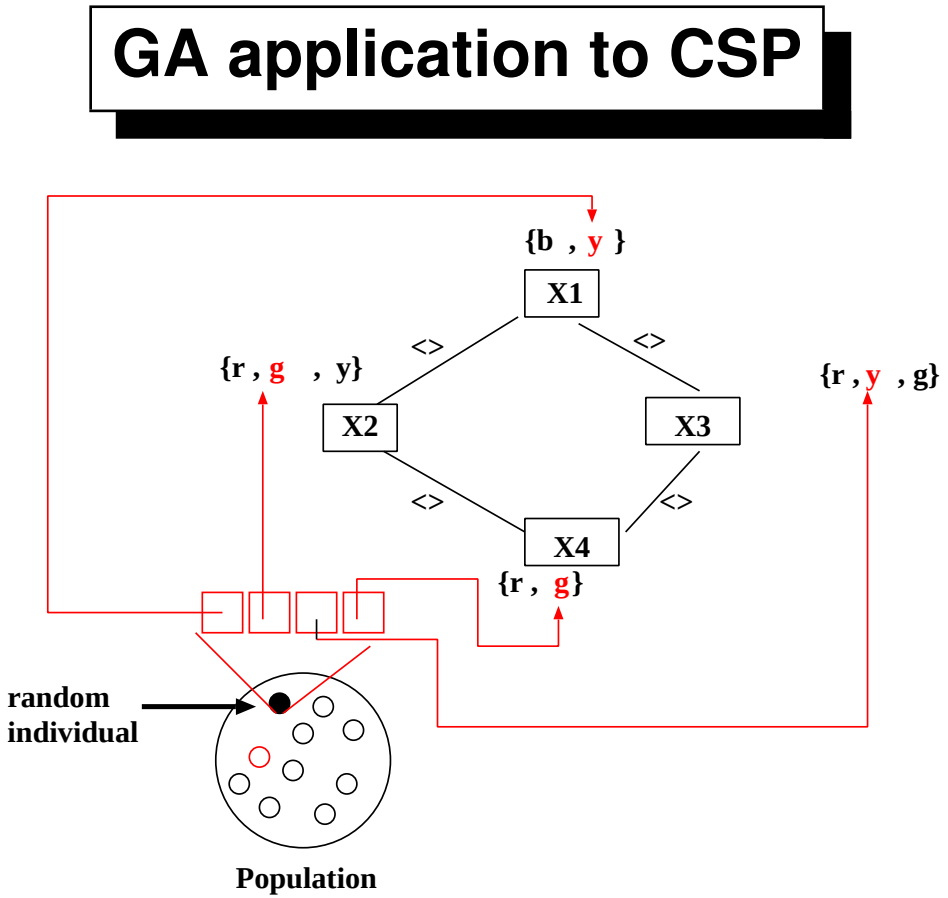


Figure 2: GA representation of the graph coloring problem

GA application to CSP

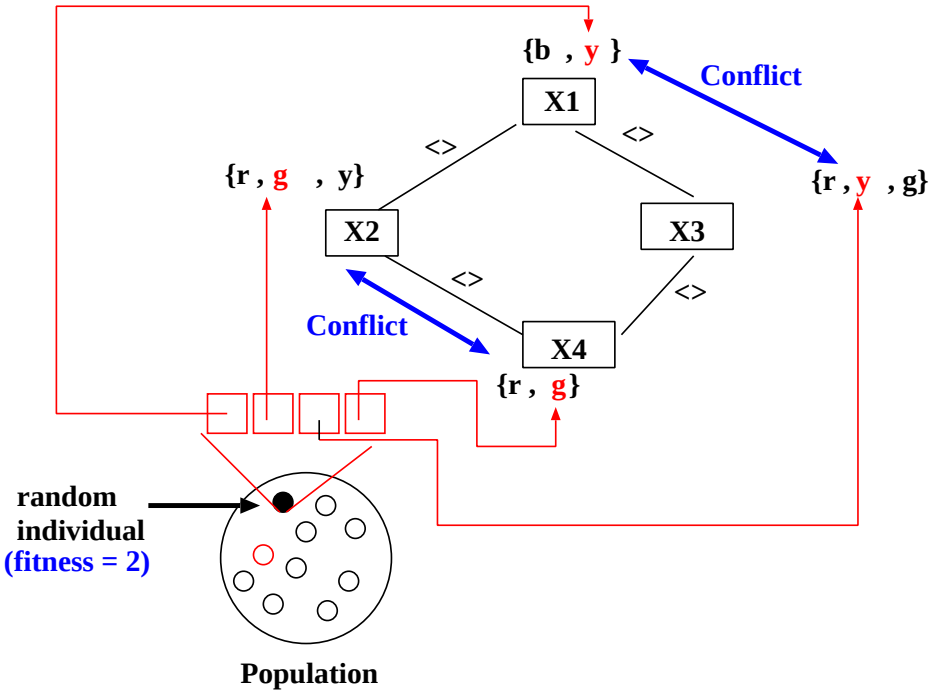


Figure 3: GA representation of the graph coloring problem

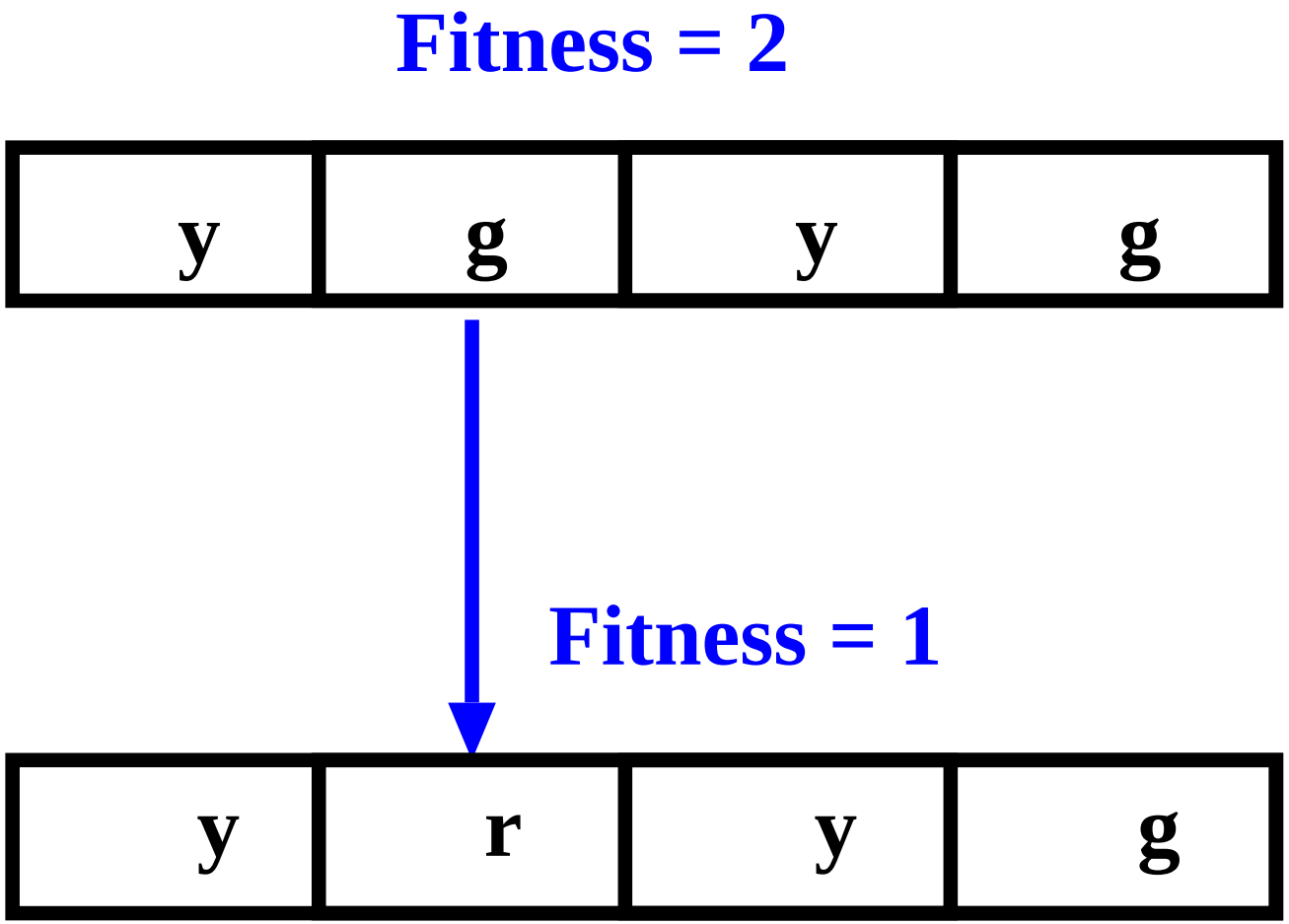


Figure 4: Mutation Operator for CSP

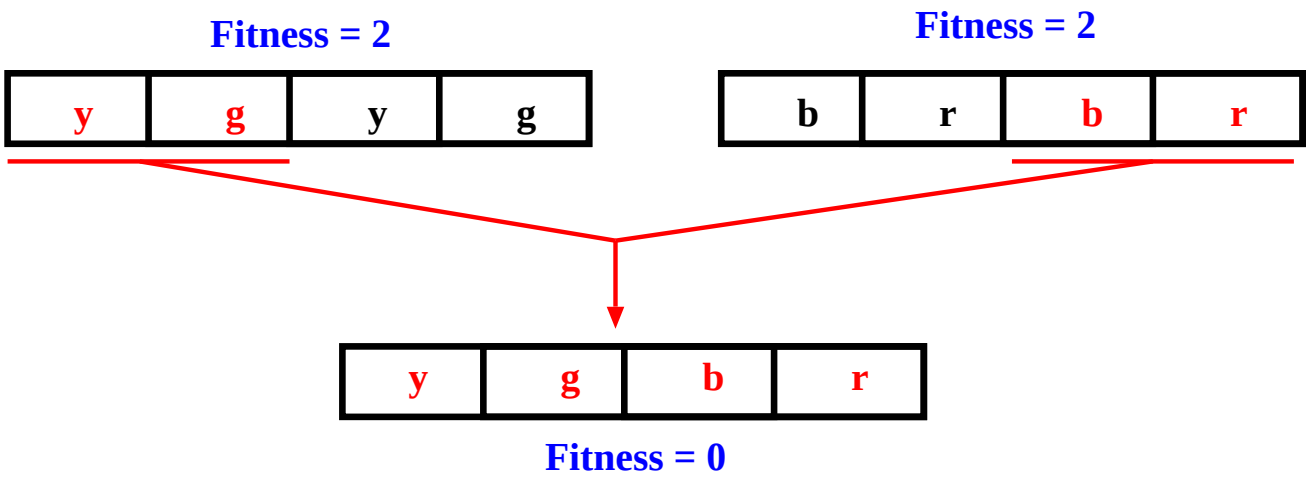


Figure 5: Crossover Operator for CSP