Hybrid metaheuristics for the *Far From Most String Problem*

Daniele Ferone[1], Paola Festa[1], and Mauricio G.C. Resende[2]

[1]Department of Mathematics and Applications "R. Caccioppoli"
University of Napoli FEDERICO II, Italy
E-mails: paola.festa@unina.it, danieleferone@gmail.com

[2]AT&T Labs Research, Florham Park, NJ, USA
Email: mgcr@research.att.com

- an *alphabet* $\Sigma = \{c_1, c_2, \ldots, c_k\}$;

- an *alphabet* $\Sigma = \{c_1, c_2, \ldots, c_k\}$;
- $s^i = (s^i_1, s^i_2, \ldots, s^i_m)$ is a sequence of length m ($|s^i| = m$) on Σ ($s^i_j \in \Sigma,\ j = 1, 2, \ldots, m$);

- an *alphabet* $\Sigma = \{c_1, c_2, \dots, c_k\}$;
- $s^i = (s^i_1, s^i_2, \dots, s^i_m)$ is a sequence of length m ($|s^i| = m$) on Σ ($s^i_j \in \Sigma,\ j = 1, 2, \dots, m$);
- given two sequences s^i and s^l on Σ such that $|s^i| = |s^l|$, $d_H(s^i, s^l)$ denotes their Hamming distance and is given by

$$d_H(s^i, s^l) = \sum_{j=1}^{|s^i|} \Phi(s^i_j, s^l_j), \qquad (1)$$

where s^i_j and s^l_j are the characters in position j in s^i and s^l, respectively, and $\Phi : \Sigma \times \Sigma \to \{0, 1\}$ is the predicate function such that

$$\Phi(a, b) = \begin{cases} 0, & \text{if } a = b; \\ 1, & \text{otherwise.} \end{cases}$$

- an *alphabet* $\Sigma = \{c_1, c_2, \ldots, c_k\}$;
- $s^i = (s^i_1, s^i_2, \ldots, s^i_m)$ is a sequence of length m ($|s^i| = m$) on Σ ($s^i_j \in \Sigma,\ j = 1, 2, \ldots, m$);
- given two sequences s^i and s^l on Σ such that $|s^i| = |s^l|$, $d_H(s^i, s^l)$ denotes their Hamming distance and is given by

$$d_H(s^i, s^l) = \sum_{j=1}^{|s^i|} \Phi(s^i_j, s^l_j), \tag{1}$$

  where s^i_j and s^l_j are the characters in position j in s^i and s^l, respectively, and $\Phi : \Sigma \times \Sigma \to \{0, 1\}$ is the predicate function such that

$$\Phi(a, b) = \begin{cases} 0, & \text{if } a = b; \\ 1, & \text{otherwise.} \end{cases}$$

- a set of sequences $\Omega = \{s^1, s^2, \ldots, s^n\}$ on Σ ($s^i \in \Sigma^m$, $i = 1, \ldots, n$)

Definition (Far From Most String Problem)

The FFMSP consists in determining a sequence far from as many as possible sequences in the input set Ω. This can be formalized by saying that given a threshold t, a string s must be found maximizing the variable x such that

$$d_H(s, \bar{s}) \geq t, \mathit{for}\ \bar{s} \in P \subseteq \Omega\ \mathit{and}\ |P| = x.$$

Applications

FFMSP belong to the class of ploblem know as *sequences consensus*.
Molecular biology applications:

- creating diagnostic probes for bacterial infections;
- discovering potential drug targets.

State of the art

- J. Lanctot, M. Li, B. Ma, S. Wang, and L. Zhang. Distinguishing string selection problems. *Information and Computation*, 185(1):41–55, 2003

State of the art

- J. Lanctot, M. Li, B. Ma, S. Wang, and L. Zhang. Distinguishing string selection problems. *Information and Computation*, 185(1):41–55, 2003
- C.N. Meneses, C.A.S. Oliveira, and P.M. Pardalos. Optimization techniques for string selection and comparison problems in genomics. *IEEE Engineering in Medicine and Biology Magazine*, 24(3):81–87, 2005.

State of the art

- J. Lanctot, M. Li, B. Ma, S. Wang, and L. Zhang. Distinguishing string selection problems. *Information and Computation*, 185(1):41–55, 2003
- C.N. Meneses, C.A.S. Oliveira, and P.M. Pardalos. Optimization techniques for string selection and comparison problems in genomics. *IEEE Engineering in Medicine and Biology Magazine*, 24(3):81–87, 2005.
- P. Festa. On some optimization problems in mulecolar biology. *Mathematical Bioscience*, 207(2):219–234, 2007

State of the art

- J. Lanctot, M. Li, B. Ma, S. Wang, and L. Zhang. Distinguishing string selection problems. *Information and Computation*, 185(1):41–55, 2003
- C.N. Meneses, C.A.S. Oliveira, and P.M. Pardalos. Optimization techniques for string selection and comparison problems in genomics. *IEEE Engineering in Medicine and Biology Magazine*, 24(3):81–87, 2005.
- P. Festa. On some optimization problems in mulecolar biology. *Mathematical Bioscience*, 207(2):219–234, 2007
- S.R. Mousavi, M. Babaie, and M. Montazerian. An improved heuristic for the far from most strings problem. *Journal of Heuristics*, 18:239–262, 2012

State of the art

- J. Lanctot, M. Li, B. Ma, S. Wang, and L. Zhang. Distinguishing string selection problems. *Information and Computation*, 185(1):41–55, 2003
- C.N. Meneses, C.A.S. Oliveira, and P.M. Pardalos. Optimization techniques for string selection and comparison problems in genomics. *IEEE Engineering in Medicine and Biology Magazine*, 24(3):81–87, 2005.
- P. Festa. On some optimization problems in mulecolar biology. *Mathematical Bioscience*, 207(2):219–234, 2007
- S.R. Mousavi, M. Babaie, and M. Montazerian. An improved heuristic for the far from most strings problem. *Journal of Heuristics*, 18:239–262, 2012
- C.C. Ribeiro, I. Rosseti, and R. Vallejos. Exploiting run time distributions to compare sequential and parallel stochastic local search algorithms. *Journal of Global Optimization*, 54:405–429, 2012

Metaheuristics

Pure metaheuristics:

- GRASP
- VNS

Hybrid metaheuristics:

- GRASP + Path-relinking
- VNS + Path-relinking
- GRASP + VNS + Path-relinking

GRASP

GRASP (Greedy Randomized Adaptive Search Procedure) is a multi-start metaheuristic, where each iteration consists of two phases.

algorithm $\texttt{GRASP}(f(\cdot), g(\cdot), \mathcal{N}, \texttt{Seed})$
s_{best}:=$\emptyset$; $f(s_{best})$:=$-\infty$;
while stopping criterion not satisfied$\rightarrow$
  s:=$\texttt{BuildGreedyRandomSolution}(\texttt{Seed}, g(\cdot))$;
  s:=$\texttt{LocalSearch}(s, f(\cdot), \mathcal{N})$;
  if $(f(s) > f(s_{best}))$ **then**
    s_{best}:=s;
  endif
endwhile
return(s_{best});
end $\texttt{GRASP}$

Construction phase

In a typical iteration, let S be a partial solution.
Let g_{min} and g_{max} be the smallest and the largest greedy values among the $|L|$ candidates, respectively, i.e.

$$g_{min} = \min_{e \in L} g(e), \quad g_{max} = \max_{e \in L} g(e).$$

A **R**estricted **C**andidate **L**ist is made up of all elements $e \in L$ with the best greedy values $g(e)$.

Random component: $e :=$`select`(RCL); $S := S \cup \{e\}$;

Adaptive component: greedy function values depend on the partial solution constructed so far.

Construction phase

For each position $j = 1, \ldots, m$ and for each $c \in \Sigma$ we compute

$$V_j(c) = \left| \{ i = 1, \ldots, n | s_j^i = c \} \right| .$$

To define the construction mechanism for the RCL, let

$$V_j^{\min} = \min_{c \in \Sigma} V_j(c), \quad V_j^{\max} = \max_{c \in \Sigma} V_j(c).$$

To build the RCL we have adopted a *value-based* (VB) mechanism: RCL is associated with a **parameter** $\alpha \in [0, 1]$ and a threshold value $\mu_j = V_j^{\min} + \alpha \cdot (V_j^{\max} - V_j^{\min})$:

$$\mathrm{RCL}_j = \{ c \in \Sigma \colon V_j(e) \leq \mu_j \}.$$

Construction phase

function GrRand(m, Σ, $\{V_j(c)\}_{j\in\{1,\ldots,m\}}^{c\in\Sigma}$, $V_j^{\min}$, $V_j^{\max}$, Seed)
for $j = 1$ **to** $m \rightarrow$
    RCL$_j$:=$\emptyset$; α:=Random([0, 1], Seed);
    $\mu := V_j^{\min} + \alpha \cdot (V_j^{\max} - V_j^{\min})$;
    for all $c \in \Sigma \rightarrow$
        if $(V_j(c) \leq \mu)$) **then**
            RCL$_j$:=RCL$_j \cup \{c\}$;
        endif
    endfor
    s_j:=Random(RCL$_j$, Seed);
endfor
return(s, $\{\text{RCL}_j\}_{j=1}^{m}$);
end GrRand

Construction phase

function $\texttt{GrRand}(m,\ \Sigma,\ \{V_j(c)\}_{j\in\{1,\dots,m\}}^{c\in\Sigma},\ V_j^{\min},\ V_j^{\max},\ \texttt{Seed})$

for $j = 1$ **to** $m \rightarrow$

  $\mathrm{RCL}_j := \emptyset;\ \alpha := \texttt{Random}([0,1],\ \texttt{Seed});$

  $\mu := V_j^{\min} + \alpha \cdot (V_j^{\max} - V_j^{\min});$

  for all $c \in \Sigma \rightarrow$

    if $(V_j(c) \le \mu))$ **then**

      $\mathrm{RCL}_j := \mathrm{RCL}_j \cup \{c\};$

    endif

  endfor

  $s_j := \texttt{Random}(\mathrm{RCL}_j,\ \texttt{Seed});$

endfor

return$(s,\ \{\mathrm{RCL}_j\}_{j=1}^{m});$

end $\texttt{GrRand}$

Construction phase

function $\texttt{GrRand}(m,\ \Sigma,\ \{V_j(c)\}_{j\in\{1,\ldots,m\}}^{c\in\Sigma},\ V_j^{\min},\ V_j^{\max},\ \texttt{Seed})$

for $j = 1$ **to** $m \rightarrow$
  $\mathrm{RCL}_j{:=}\emptyset;\ \alpha{:=}\texttt{Random}([0,1],\ \texttt{Seed});$
  $\mu{:=}V_j^{\min} + \alpha \cdot (V_j^{\max} - V_j^{\min});$
  for all $c \in \Sigma \rightarrow$
    if $(V_j(c) \leq \mu))$ **then**
      $\mathrm{RCL}_j{:=}\mathrm{RCL}_j \cup \{c\};$
    endif
  endfor
  $s_j{:=}\texttt{Random}(\mathrm{RCL}_j,\ \texttt{Seed});$
endfor
return$(s,\ \{\mathrm{RCL}_j\}_{j=1}^{m});$

end $\texttt{GrRand}$

Construction phase

function $\texttt{GrRand}(m,\ \Sigma,\ \{V_j(c)\}_{j\in\{1,\dots,m\}}^{c\in\Sigma},\ V_j^{\min},\ V_j^{\max},\ \texttt{Seed})$

for $j = 1$ **to** $m \rightarrow$
    $\mathrm{RCL}_j{:=}\emptyset$; $\alpha{:=}\texttt{Random}([0,1],\ \texttt{Seed})$;
    $\mu{:=}V_j^{\min} + \alpha \cdot (V_j^{\max} - V_j^{\min})$;
    for all $c \in \Sigma \rightarrow$
        if $(V_j(c) \leq \mu))$ **then**
            $\mathrm{RCL}_j{:=}\mathrm{RCL}_j \cup \{c\}$;
        endif
    endfor
    $s_j{:=}\texttt{Random}(\mathrm{RCL}_j,\ \texttt{Seed})$;
endfor
return$(s,\ \{\mathrm{RCL}_j\}_{j=1}^{m})$;

end $\texttt{GrRand}$

Neighborhood

To define local search, one needs to specify a local neighborhood structure $N(s)$ of a solution s:

$$N(s) = \{\bar{s} | d_H(s, \bar{s}) = 1 \wedge (\forall j = 1, \ldots, n, s_j \neq \bar{s}_j \implies \bar{s}_j \in RCL_j)\}$$

Local search

```
function LocalSearch(m, s, f(·), {RCL_j}_{j=1}^m)
  max:=f(s); change:=.TRUE.;
  while (change)→
    change:=.FALSE.;
    for j = 1 to m→
        temp:=s_j;
        for all c ∈RCL_j→
            s_j:=c;
            if (f(s) > max) then
                max:=f(s); temp:=c; change:=.TRUE.; break;
            endif
        endfor
        s_j:=temp;
    endfor
  endwhile
  return(s);
end LocalSearch
```

Local search

```
function LocalSearch(m, s, f(·), {RCL_j}_{j=1}^m)
  max:=f(s); change:=.TRUE.;
  while (change)→
    change:=.FALSE.;
    for j = 1 to m→
        temp:=s_j;
        for all c ∈RCL_j→
            s_j:=c;
            if (f(s) > max) then
                max:=f(s); temp:=c; change:=.TRUE.; break;
            endif
        endfor
        s_j:=temp;
    endfor
  endwhile
  return(s);
end LocalSearch
```

VNS

VNS (Variable Neighborhood Search) is a metaheuristic based on the exploration of a dynamic neighborhood model.

- Let N_k, $k = 1, \ldots, k_{max}$, be a set of pre-defined neighborhood structures;

VNS

VNS (Variable Neighborhood Search) is a metaheuristic based on the exploration of a dynamic neighborhood model.

- Let N_k, $k = 1, \ldots, k_{max}$, be a set of pre-defined neighborhood structures;
- let s the current solution, in the first phase, a solution $s' \in N_k(s)$ is randomly selected;

VNS

VNS (Variable Neighborhood Search) is a metaheuristic based on the exploration of a dynamic neighborhood model.

- Let N_k, $k = 1, \ldots, k_{max}$, be a set of pre-defined neighborhood structures;
- let s the current solution, in the first phase, a solution $s' \in N_k(s)$ is randomly selected;
- in the second phase, a solution s'' is obtained by applying local search to s';

VNS

VNS (Variable Neighborhood Search) is a metaheuristic based on the exploration of a dynamic neighborhood model.

- Let N_k, $k = 1, \ldots, k_{max}$, be a set of pre-defined neighborhood structures;
- let s the current solution, in the first phase, a solution $s' \in N_k(s)$ is randomly selected;
- in the second phase, a solution s'' is obtained by applying local search to s';
- if s'' is better than s, s'' is set as current solution; otherwise k is increased;

VNS

VNS (Variable Neighborhood Search) is a metaheuristic based on the exploration of a dynamic neighborhood model.

- Let N_k, $k = 1, \ldots, k_{max}$, be a set of pre-defined neighborhood structures;
- let s the current solution, in the first phase, a solution $s' \in N_k(s)$ is randomly selected;
- in the second phase, a solution s'' is obtained by applying local search to s';
- if s'' is better than s, s'' is set as current solution; otherwise k is increased;
- the above steps are repeated until some stopping condition is satisfied.

VNS

VNS (Variable Neighborhood Search) is a metaheuristic based on the exploration of a dynamic neighborhood model.

- Let N_k, $k = 1, \ldots, k_{max}$, be a set of pre-defined neighborhood structures;
- let s the current solution, in the first phase, a solution $s' \in N_k(s)$ is randomly selected;
- in the second phase, a solution s'' is obtained by applying local search to s';
- if s'' is better than s, s'' is set as current solution; otherwise k is increased;
- the above steps are repeated until some stopping condition is satisfied.

$$N_k(s) = \{\bar{s} | d_H(s, \bar{s}) = k \wedge (\forall j = 1, \ldots, n, s_j \neq \bar{s}_j \implies \bar{s}_j \in RCL_j)\}$$

algorithm $\texttt{VNS}(m, \Sigma, f(\cdot), k_{max}, \texttt{Seed})$

$s_{best}{:=}\emptyset;\ f(s_{best}){:=}-\infty;$

while stopping criterion not satisfied$\rightarrow$

  $k{:=}1;\ s{:=}\texttt{BuildRand}(m, \Sigma, \texttt{Seed});$ /* pure randomly */

  while $(k \leq k_{max})\rightarrow$

    $s'{:=}\texttt{Random}(N_k(s), \texttt{Seed}, \{\mathrm{RCL}_j\}_{j=1}^{m});$

    $s''{:=}\texttt{locsearch}(m, s', f(\cdot), \{\mathrm{RCL}_j\}_{j=1}^{m});$

    if $(f(s'') > f(s))$ **then**

      $s{:=}s'';\ k{:=}1;$

      if $(f(s'') > f(s_{best}))$ **then** $s_{best}{:=}s'';$

      endif

    else $k{:=}k+1;$

    endif

  endwhile

endwhile

return$(s_{best});$

end $\texttt{VNS}$

Path-relinking

- It explores trajectories that connect high quality solutions.
- Path is generated by selecting modifications (moves) that introduce attributes of the guiding solution G in the initial solution I.
- At each step, all moves ($d(I, G)$) that incorporate attributes of the guiding solution are analyzed and best move is taken.

Path-relinking

algorithm `Path-relinking`$(m,\ f(\cdot), \mathtt{s}, \mathcal{E},$ `Seed`$)$

$\hat{\mathbf{s}}$:=`Random`$(\mathcal{E},$ `Seed`$)$;

$f^* := \max\{f(\mathbf{s}), f(\hat{\mathbf{s}})\}$; $\mathbf{s}^* := \arg\max\{f(\mathbf{s}), f(\hat{\mathbf{s}})\}$;

$\mathbf{s}' := \arg\min\{f(\mathbf{s}), f(\hat{\mathbf{s}})\}$; $\hat{\mathbf{s}}$:=$\mathbf{s}^*$;

$\Delta(\mathbf{s}', \hat{\mathbf{s}})$:=$\{i = 1, \ldots, m \mid \mathbf{s'}_i \neq \hat{\mathbf{s}}_i\}$;

while $(\Delta(\mathbf{s}', \hat{\mathbf{s}}) \neq \emptyset) \rightarrow$

    $i^* := \arg\max\{f(\mathbf{s}' \oplus i) \mid i \in \Delta(\mathbf{s}', \hat{\mathbf{s}})\}$;

    $\Delta(\mathbf{s}' \oplus i^*, \hat{\mathbf{s}}) := \Delta(\mathbf{s}', \hat{\mathbf{s}}) \setminus \{i^*\}$;

    $\mathbf{s}' := \mathbf{s}' \oplus i^*$;

    if $(f(\mathbf{s}') > f^*)$ **then**

        $f^* := f(\mathbf{s}')$; $\mathbf{s}^* := \mathbf{s}'$;

    endif;

endwhile;

return $(\mathbf{s}^*)$;

end `Path-relinking`;

Path-relinking

algorithm Path-relinking(m, $f(\cdot)$, s, $\mathcal{E}$, Seed)
 $\hat{\mathbf{s}}$:=Random($\mathcal{E}$, Seed);
 $f^* := \max\{f(\mathbf{s}), f(\hat{\mathbf{s}})\}$; $\mathbf{s}^* := \arg\max\{f(\mathbf{s}), f(\hat{\mathbf{s}})\}$;
 $\mathbf{s}' := \arg\min\{f(\mathbf{s}), f(\hat{\mathbf{s}})\}$; $\hat{\mathbf{s}} := \mathbf{s}^*$;
 $\Delta(\mathbf{s}', \hat{\mathbf{s}}) := \{i = 1, \ldots, m \mid \mathbf{s}'_i \neq \hat{\mathbf{s}}_i\}$;
 while $(\Delta(\mathbf{s}', \hat{\mathbf{s}}) \neq \emptyset) \rightarrow$
   $i^* := \arg\max\{f(\mathbf{s}' \oplus i) \mid i \in \Delta(\mathbf{s}', \hat{\mathbf{s}})\}$;
   $\Delta(\mathbf{s}' \oplus i^*, \hat{\mathbf{s}}) := \Delta(\mathbf{s}', \hat{\mathbf{s}}) \setminus \{i^*\}$;
   $\mathbf{s}' := \mathbf{s}' \oplus i^*$;
   if $(f(\mathbf{s}') > f^*)$ **then**
     $f^* := f(\mathbf{s}')$; $\mathbf{s}^* := \mathbf{s}'$;
   endif;
 endwhile;
 return $(\mathbf{s}^*)$;
end Path-relinking;

Path-relinking

algorithm Path-relinking(m, $f(\cdot)$, s, $\mathcal{E}$, Seed)

$\hat{s}$:=Random($\mathcal{E}$, Seed);

$f^* := \max\{f(\mathbf{s}), f(\hat{\mathbf{s}})\}$; $\mathbf{s}^* := \arg\max\{f(\mathbf{s}), f(\hat{\mathbf{s}})\}$;

$\mathbf{s}' := \arg\min\{f(\mathbf{s}), f(\hat{\mathbf{s}})\}$; $\hat{\mathbf{s}} := \mathbf{s}^*$;

$\Delta(\mathbf{s}', \hat{\mathbf{s}}) := \{i = 1, \ldots, m \mid \mathbf{s}'_i \neq \hat{\mathbf{s}}_i\}$;

while $(\Delta(\mathbf{s}', \hat{\mathbf{s}}) \neq \emptyset) \rightarrow$

  $i^* := \arg\max\{f(\mathbf{s}' \oplus i) \mid i \in \Delta(\mathbf{s}', \hat{\mathbf{s}})\}$;

  $\Delta(\mathbf{s}' \oplus i^*, \hat{\mathbf{s}}) := \Delta(\mathbf{s}', \hat{\mathbf{s}}) \setminus \{i^*\}$;

  $\mathbf{s}' := \mathbf{s}' \oplus i^*$;

  if $(f(\mathbf{s}') > f^*)$ **then**

    $f^* := f(\mathbf{s}')$; $\mathbf{s}^* := \mathbf{s}'$;

  endif;

endwhile;

return $(\mathbf{s}^*)$;

end Path-relinking;

Path-relinking

algorithm `Path-relinking`$(m,\ f(\cdot), \mathrm{s}, \mathcal{E},$ `Seed`$)$

$\hat{\mathrm{s}}$:=`Random`$(\mathcal{E},$ `Seed`$)$;

$f^* := \max\{f(\mathbf{s}), f(\hat{\mathbf{s}})\}$; $\mathbf{s}^* := \arg\max\{f(\mathbf{s}), f(\hat{\mathbf{s}})\}$;

$\mathbf{s}' := \arg\min\{f(\mathbf{s}), f(\hat{\mathbf{s}})\}$; $\hat{\mathbf{s}} := \mathbf{s}^*$;

$\Delta(\mathbf{s}', \hat{\mathbf{s}}) := \{i = 1, \ldots, m \mid \mathbf{s}'_i \neq \hat{\mathbf{s}}_i\}$;

while $(\Delta(\mathbf{s}', \hat{\mathbf{s}}) \neq \emptyset) \rightarrow$

  $i^* := \arg\max\{f(\mathbf{s}' \oplus i) \mid i \in \Delta(\mathbf{s}', \hat{\mathbf{s}})\}$;

  $\Delta(\mathbf{s}' \oplus i^*, \hat{\mathbf{s}}) := \Delta(\mathbf{s}', \hat{\mathbf{s}}) \setminus \{i^*\}$;

  $\mathbf{s}' := \mathbf{s}' \oplus i^*$;

  if $(f(\mathbf{s}') > f^*)$ **then**

    $f^* := f(\mathbf{s}')$; $\mathbf{s}^* := \mathbf{s}'$;

  endif;

endwhile;

return $(\mathbf{s}^*)$;

end `Path-relinking`;

Path-relinking

algorithm $\texttt{Path-relinking}(m,\ f(\cdot), \mathtt{s}, \mathcal{E},\ \texttt{Seed})$

$\hat{\mathbf{s}} := \texttt{Random}(\mathcal{E},\ \texttt{Seed})$;

$f^* := \max\{f(\mathbf{s}), f(\hat{\mathbf{s}})\}$; $\mathbf{s}^* := \arg\max\{f(\mathbf{s}), f(\hat{\mathbf{s}})\}$;

$\mathbf{s}' := \arg\min\{f(\mathbf{s}), f(\hat{\mathbf{s}})\}$; $\hat{\mathbf{s}} := \mathbf{s}^*$;

$\Delta(\mathbf{s}', \hat{\mathbf{s}}) := \{i = 1, \ldots, m \mid \mathbf{s}'_i \neq \hat{\mathbf{s}}_i\}$;

while $(\Delta(\mathbf{s}', \hat{\mathbf{s}}) \neq \emptyset) \rightarrow$

$\quad i^* := \arg\max\{f(\mathbf{s}' \oplus i) \mid i \in \Delta(\mathbf{s}', \hat{\mathbf{s}})\}$;

$\quad \Delta(\mathbf{s}' \oplus i^*, \hat{\mathbf{s}}) := \Delta(\mathbf{s}', \hat{\mathbf{s}}) \setminus \{i^*\}$;

$\quad \mathbf{s}' := \mathbf{s}' \oplus i^*$;

$\quad$ **if** $(f(\mathbf{s}') > f^*)$ **then**

$\quad\quad f^* := f(\mathbf{s}')$; $\mathbf{s}^* := \mathbf{s}'$;

$\quad$ **endif**;

endwhile;

return $(\mathbf{s}^*)$;

end $\texttt{Path-relinking}$;

algorithm $\texttt{GRASP+PR}(t,\ m,\ \Sigma,\ f_t(\cdot),\ \{V_j(c)\}_{j\in\{1,\ldots,m\}}^{c\in\Sigma},\ \texttt{Seed},\ \texttt{MaxElite})$

$s_{best}{:=}\emptyset$; $f_t(s_{best}){:=}-\infty$; $\mathcal{E} := \emptyset$; $iter{:=}0$;

for $j = 1$ **to** $m \rightarrow$

  $V_j^{\min}{:=}\min_{c\in\Sigma} V_j(c)$; $V_j^{\max}{:=}\max_{c\in\Sigma} V_j(c)$;

while stopping criterion not satisfied$\rightarrow$

  $iter{:=}iter+1$;

  $[s, \{\mathrm{RCL}_j\}_{j=1}^{m}]{:=}\texttt{GrRand}(m,\ \Sigma,\ \{V_j(c)\}_{j\in\{1,\ldots,m\}}^{c\in\Sigma},\ V_j^{\min},\ V_j^{\max},\ \texttt{Seed})$;

  $s{:=}\texttt{LocalSearch}(t,\ m,\ s,\ f_t(\cdot),\ \{\mathrm{RCL}_j\}_{j=1}^{m})$;

  if $(iter \leq \texttt{MaxElite})$ **then**

    $\mathcal{E} := \mathcal{E} \cup \{s\}$;

    if $(f_t(s) > f_t(s_{best}))$ **then** $s_{best}{:=}s$;

  else

    $\overline{s}{:=}\texttt{Path-relinking}(t,\ m,\ f_t(\cdot),\ s,\ \mathcal{E},\ \texttt{Seed})$;

    $\texttt{AddToElite}(\mathcal{E}, \overline{s})$;

    if $(f_t(\overline{s}) > f_t(s_{best}))$ **then** $s_{best}{:=}\overline{s}$;

return(s_{best});

end $\texttt{GRASP+PR}$

algorithm $\texttt{GRASP+PR}(t,\ m,\ \Sigma,\ f_t(\cdot),\ \{V_j(c)\}_{j\in\{1,\dots,m\}}^{c\in\Sigma},\ \texttt{Seed},\ \texttt{MaxElite})$

$s_{best}{:=}\emptyset;\ f_t(s_{best}){:=}-\infty;\ \mathcal{E} := \emptyset;\ iter{:=}0;$

for $j = 1$ **to** $m \rightarrow$

  $V_j^{\min}{:=}\min_{c\in\Sigma} V_j(c);\ V_j^{\max}{:=}\max_{c\in\Sigma} V_j(c);$

while stopping criterion not satisfied$\rightarrow$

  $iter{:=}iter + 1;$

  $[s, \{\mathrm{RCL}_j\}_{j=1}^{m}]{:=}\texttt{GrRand}(m,\ \Sigma,\ \{V_j(c)\}_{j\in\{1,\dots,m\}}^{c\in\Sigma},\ V_j^{\min},\ V_j^{\max},\ \texttt{Seed});$

  $s{:=}\texttt{LocalSearch}(t,\ m,\ s,\ f_t(\cdot),\ \{\mathrm{RCL}_j\}_{j=1}^{m});$

  if $(iter \leq \texttt{MaxElite})$ **then**

      $\mathcal{E} := \mathcal{E} \cup \{s\};$

      if $(f_t(s) > f_t(s_{best}))$ **then** $s_{best}{:=}s;$

   else

      $\overline{s}{:=}\texttt{Path-relinking}(t,\ m,\ f_t(\cdot),\ s,\ \mathcal{E},\ \texttt{Seed});$

      $\texttt{AddToElite}(\mathcal{E}, \overline{s});$

      if $(f_t(\overline{s}) > f_t(s_{best}))$ **then** $s_{best}{:=}\overline{s};$

return$(s_{best});$

end $\texttt{GRASP+PR}$

algorithm GRASP+PR(t, m, Σ, $f_t(\cdot)$, $\{V_j(c)\}_{j\in\{1,\ldots,m\}}^{c\in\Sigma}$, Seed, MaxElite)
s_{best}:=$\emptyset$; $f_t(s_{best})$:=$-\infty$; $\mathcal{E} := \emptyset$; $iter$:=0;
for $j = 1$ **to** $m\rightarrow$
  $V_j^{\min}$:=$\min_{c\in\Sigma} V_j(c)$; $V_j^{\max}$:=$\max_{c\in\Sigma} V_j(c)$;
while stopping criterion not satisfied$\rightarrow$
  $iter$:=$iter + 1$;
  $[s, \{\mathrm{RCL}_j\}_{j=1}^{m}]$:=GrRand($m$, Σ, $\{V_j(c)\}_{j\in\{1,\ldots,m\}}^{c\in\Sigma}$, $V_j^{\min}$, $V_j^{\max}$, Seed);
  s:=LocalSearch(t, m, s, $f_t(\cdot)$, $\{\mathrm{RCL}_j\}_{j=1}^{m}$);
  if ($iter \leq$ MaxElite) **then**
    $\mathcal{E} := \mathcal{E} \cup \{s\}$;
    if ($f_t(s) > f_t(s_{best})$) **then** s_{best}:=s;
  else
    $\overline{s}$:=Path-relinking(t, m, $f_t(\cdot)$, s, $\mathcal{E}$, Seed);
    AddToElite($\mathcal{E}$, $\overline{s}$);
    if ($f_t(\overline{s}) > f_t(s_{best})$) **then** s_{best}:=$\overline{s}$;
return(s_{best});
end GRASP+PR

algorithm $\texttt{GRASP+PR}(t,\ m,\ \Sigma,\ f_t(\cdot),\ \{V_j(c)\}_{j\in\{1,\dots,m\}}^{c\in\Sigma},\ \texttt{Seed},\ \texttt{MaxElite})$

$s_{best}{:=}\emptyset;\ f_t(s_{best}){:=}-\infty;\ \mathcal{E} := \emptyset;\ iter{:=}0;$

for $j = 1$ **to** $m \rightarrow$

  $V_j^{\min}{:=}\min_{c\in\Sigma} V_j(c);\ V_j^{\max}{:=}\max_{c\in\Sigma} V_j(c);$

while stopping criterion not satisfied$\rightarrow$

  $iter{:=}iter + 1;$

  $[s, \{\mathrm{RCL}_j\}_{j=1}^{m}]{:=}\texttt{GrRand}(m,\ \Sigma,\ \{V_j(c)\}_{j\in\{1,\dots,m\}}^{c\in\Sigma},\ V_j^{\min},\ V_j^{\max},\ \texttt{Seed});$

  $s{:=}\texttt{LocalSearch}(t,\ m,\ s,\ f_t(\cdot),\ \{\mathrm{RCL}_j\}_{j=1}^{m});$

  if $(iter \leq \texttt{MaxElite})$ **then**

    $\mathcal{E} := \mathcal{E} \cup \{s\};$

    if $(f_t(s) > f_t(s_{best}))$ **then** $s_{best}{:=}s;$

  else

    $\overline{s}{:=}\texttt{Path-relinking}(t,\ m,\ f_t(\cdot),\ s,\ \mathcal{E},\ \texttt{Seed});$

    $\texttt{AddToElite}(\mathcal{E}, \overline{s});$

    if $(f_t(\overline{s}) > f_t(s_{best}))$ **then** $s_{best}{:=}\overline{s};$

return$(s_{best});$

end $\texttt{GRASP+PR}$

Further hybrids

- GRASP+VNS: VNS applied as local search;
- VNS+PR: Path-relinking applied as intensification strategy;
- GRASP+VNS+PR: VNS applied as local search and Path-relinking as intensification strategy.

Compiler: *cc (GCC) 4.1.3 20070929 (prerelease) (Ubuntu 4.1.2-16ubuntu2*
Hardware: *Intel Core i7 Quad core @ 2.67 GHz RAM 6GB*
OS: *Linux (Ubuntu 11.10)*

- m ranges from 300 to 800;
- n ranges from 100 to 200;
- threshold t varies from $75\%m$ to $85\%m$;
- 100 random istances for each problem size;
- $k_{max} = 30$
- `MaxElite` $= 10$
- in Path-relinking a solution s is *sufficiently different* if $d_H(s, \epsilon) \geq \frac{m}{2}$, for all $\epsilon \in \mathcal{E}$.

n, m, t	GRASP		VNS		VNS+PR	
	z	Time	z	Time	z	Time
$100, 300, 0.75$	**100**	1.37	94.47	72.45	**100**	7.44
$100, 300, 0.80$	67.86	1.67	19.98	71.02	48.58	77.43
$100, 300, 0.85$	3.56	1.72	1.12	37.65	3.53	44.19
$100, 600, 0.75$	**100**	1.56	91.78	278.94	**100**	31.91
$100, 600, 0.80$	65.35	2.31	8.51	264.66	20.72	295.49
$100, 600, 0.85$	1.21	1.28	0.04	152.64	0.91	186.71
$100, 800, 0.75$	**100**	1.84	87.36	549.60	**100**	63.00
$100, 800, 0.80$	67.76	1.42	4.41	450.63	10.94	527.76
$100, 800, 0.85$	0.30	2.98	0.66	273.98	0.63	329.48
$200, 300, 0.75$	197.78	1.22	180.08	135.61	**200**	47.91
$200, 300, 0.80$	76.50	1.39	36.71	150.51	66.81	160.37
$200, 300, 0.85$	2.83	1.59	2.16	86.60	4.62	104.13
$200, 600, 0.75$	**200**	1.94	178.11	545.50	**200**	75.43
$200, 600, 0.80$	62.80	1.63	11.93	588.35	33.41	625.66
$200, 600, 0.85$	0.98	1.79	0.71	305.29	0.96	369.29
$200, 800, 0.75$	**200**	1.04	175.06	947.80	**200**	193.30
$200, 800, 0.80$	44.66	1.75	6.37	987.35	17.12	1102.47
$200, 800, 0.85$	0.86	1.55	0.15	544.21	0.49	659.02

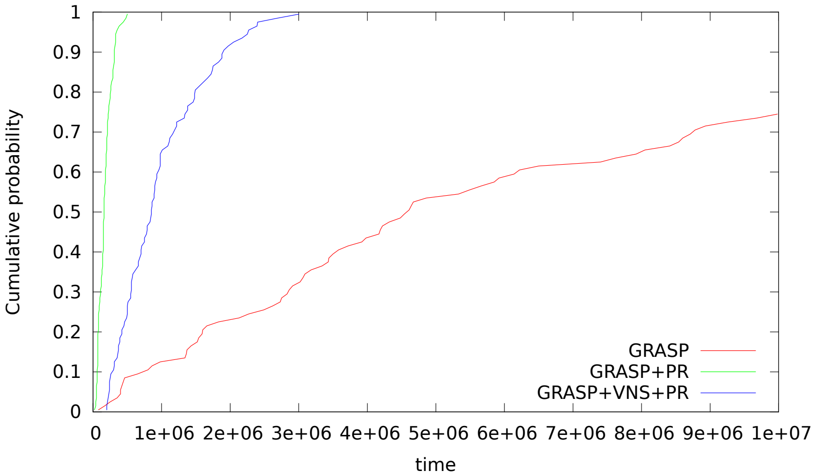
n, m, t	GRASP		GRASP+PR		GRASP+VNS+PR	
	z	Time	z	Time	z	Time
$100, 300, 0.75$	**100**	1.37	**100**	1.41	**100**	1.71
$100, 300, 0.80$	67.86	1.67	76.17	3.21	**76.68**	33.28
$100, 300, 0.85$	3.56	1.72	10.17	4.17	**12.23**	31.36
$100, 600, 0.75$	**100**	1.56	**100**	1.89	**100**	1.11
$100, 600, 0.80$	65.35	2.31	78.83	11.77	**80.47**	122.65
$100, 600, 0.85$	1.21	1.28	3.58	11.07	**4.36**	91.87
$100, 800, 0.75$	**100**	1.84	**100**	1.34	**100**	2.48
$100, 800, 0.80$	67.76	1.42	80.37	21.10	**82.30**	251.26
$100, 800, 0.85$	0.30	2.98	0.97	31.45	**2.51**	148.98
$200, 300, 0.75$	197.78	1.22	**200**	1.85	**200**	3.70
$200, 300, 0.80$	76.50	1.39	93.70	6.52	**94.45**	65.44
$200, 300, 0.85$	2.83	1.59	9.26	8.59	**11.12**	68.20
$200, 600, 0.75$	**200**	1.94	**200**	1.61	**200**	3.39
$200, 600, 0.80$	62.80	1.63	83.91	24.91	**86.17**	274.93
$200, 600, 0.85$	0.98	1.79	1.51	31.22	**2.38**	174.73
$200, 800, 0.75$	**200**	1.04	**200**	1.33	**200**	1.61
$200, 800, 0.80$	44.66	1.75	71.28	43.63	**72.44**	519.59
$200, 800, 0.85$	0.86	1.55	1.93	59.20	**3.71**	311.81

n, m, t	GRASP	GRASP+PR	GRASP+VNS+PR	VNS	VNS+PR
100, 300, 0.75	**100**	**100**	**100**	94	**100**
100, 300, 0.8	71.07	**79.61**	78.12	23.43	49.54
100, 300, 0.85	6.41	**13.18**	11.86	1.02	3.27
100, 600, 0.75	**100**	**100**	**100**	91.79	**100**
100, 600, 0.8	70.24	**80.13**	78.05	6.38	11.29
100, 600, 0.85	2.73	**4.98**	4.48	0.03	0.12
100, 800, 0.75	**100**	**100**	**100**	85.18	**100**
100, 800, 0.8	70.07	**82.64**	79.45	3.71	8.42
100, 800, 0.85	1.17	**1.84**	1.65	0	0.03
200, 300, 0.75	199.81	**200**	**200**	179.34	**200**
200, 300, 0.8	81.75	**100**	95.11	34.71	61.67
200, 300, 0.85	4.82	**11.90**	11.03	2.32	3.70
200, 600, 0.75	**200**	**200**	**200**	172.41	**200**
200, 600, 0.8	66.23	**88.49**	80.31	10.25	19.10
200, 600, 0.85	1.03	**2.42**	1.73	0.09	0.97
200, 800, 0.75	**200**	**200**	**200**	164.01	194.45
200, 800, 0.8	49.87	**73.08**	62.36	4.23	8.17
200, 800, 0.85	0.08	**0.21**	0.17	0.06	0.85

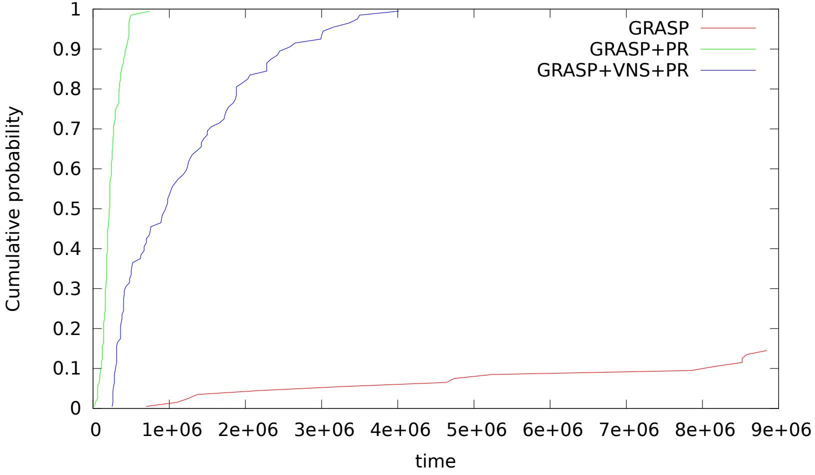
Time-to-target

- Target value $\hat{z}$;
- 100 different random instances;
- to the i^{th} sorted running time (t_i) we associate a probability $p_i = \frac{i-1/2}{100}$;
- plot the points $z_i = (t_i, p_i)$ for $i = 1, \ldots, 100$.

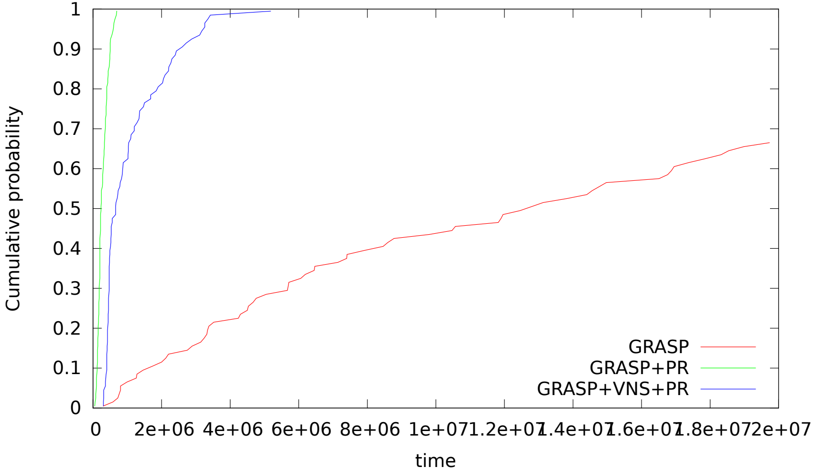
$n = 100, m = 300, t = 240, \hat{z} = 70$



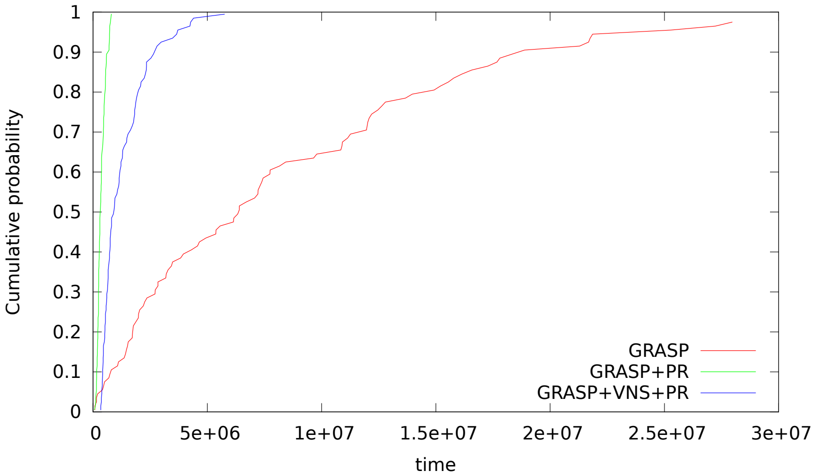
$n = 100, m = 300, t = 252, \hat{z} = 12$



$n = 200, m = 300, t = 240, \hat{z} = 80$



$n = 300, m = 300, t = 240, \hat{z} = 84$



Future work

- Validate the numerical results;

Future work

- Validate the numerical results;
- To perform at the end of computation a post-optimization phase;

Future work

- Validate the numerical results;
- To perform at the end of computation a post-optimization phase;
- To implement alternative linking strategies in Path-relinking;

Future work

- Validate the numerical results;
- To perform at the end of computation a post-optimization phase;
- To implement alternative linking strategies in Path-relinking;
- To introduce the tool designed by Ribeiro et al. for characterizing stochastic algorithms running times;

Future work

- Validate the numerical results;
- To perform at the end of computation a post-optimization phase;
- To implement alternative linking strategies in Path-relinking;
- To introduce the tool designed by Ribeiro et al. for characterizing stochastic algorithms running times;
- To integrate in the local search of the algorithms the new function devised by Mousavi et al. and to be used in alternative to the objective function when evaluating neighbor solutions.

Future work

- Validate the numerical results;
- To perform at the end of computation a post-optimization phase;
- To implement alternative linking strategies in Path-relinking;
- To introduce the tool designed by Ribeiro et al. for characterizing stochastic algorithms running times;
- To integrate in the local search of the algorithms the new function devised by Mousavi et al. and to be used in alternative to the objective function when evaluating neighbor solutions.

THANK YOU!