AMMM PROJECT

ARNAU ABELLA

ANDREA DE LAS HERAS

UNIVERSITAT POLITÈCNICA DE CATALUNYA

Contents:

MODEL FOR THE PROBLEM

Model for the Problem - Notation

L	Set of possible locations, index l.
C	Ser of cities, index c.
T	Set of types, index t.
d_centre	Minimum distance between centres.
cap_t	capacity of the type t.
d_city_t	Working distance of the type t.
$cost_t$	Installation cost of the type t.
p_c	Population of the city c.
p_{cl}	binary. Equals to 1 if a centre in location l serve as primary centre the city c; 0 otherwise.
s_{cl}	binary. Equals to 1 if a centre in location l serve as secondary centre the city c; 0 otherwise.
x_{lt}	binary. Equals to 1 if a centre of type t is emplaced in the location l; 0 otherwise.

Model for the Problem - Restrictions

(1) Each city has assigned exactly one primary centre.

(2) Each city has assigned exactly one secondary centre.

(3) For each city, its primary centre must be different of its secondary centre.

(4) The distance between each pair of centres must be at least the minimum.

(5) The distance between a city and its primary centre cannot exceed the working distance of that centre's type.

(6) The distance between a city and its secondary centre cannot exceed three times the working distance of that centre's type.

(7) For each centre, the capacity of the centre's type cannot be exceeded by the sum of the populations of the cities it serves as a primary centre and the 10% of the populations of the cities it serves as a secondary centre.

Model for the Problem

(1) $\sum_{l \in L} p_{cl} = 1 \quad \forall c \in C$

(2) Each city has assigned exactly one secondary centre.

(3) For each city, its primary centre must be different of its secondary centre.

(4) The distance between each pair of centres must be at least the minimum.

(5) The distance between a city and its primary centre cannot exceed the working distance of that centre's type.

(6) The distance between a city and its secondary centre cannot exceed three times the working distance of that centre's type.

(7) For each centre, the capacity of the centre's type cannot be exceeded by the sum of the populations of the cities it serves as a primary centre and the 10% of the populations of the cities it serves as a secondary centre.

Model for the Problem

(1) $\sum_{l \in L} p_{cl} = 1 \quad \forall c \in C$

(2) $\sum_{l \in L} s_{cl} = 1 \quad \forall c \in C$

(3) For each city, its primary centre must be different of its secondary centre.

(4) The distance between each pair of centres must be at least the minimum.

(5) The distance between a city and its primary centre cannot exceed the working distance of that centre's type.

(6) The distance between a city and its secondary centre cannot exceed three times the working distance of that centre's type.

(7) For each centre, the capacity of the centre's type cannot be exceeded by the sum of the populations of the cities it serves as a primary centre and the 10% of the populations of the cities it serves as a secondary centre.

Model for the Problem

(1) $\sum_{l \in L} p_{cl} = 1 \quad \forall c \in C$

(2) $\sum_{l \in L} s_{cl} = 1 \quad \forall c \in C$

(3) $p_{cl} + s_{cl} \leq 1 \quad \forall c \in C, l \in L$

(4) The distance between each pair of centres must be at least the minimum.

(5) The distance between a city and its primary centre cannot exceed the working distance of that centre's type.

(6) The distance between a city and its secondary centre cannot exceed three times the working distance of that centre's type.

(7) For each centre, the capacity of the centre's type cannot be exceeded by the sum of the populations of the cities it serves as a primary centre and the 10% of the populations of the cities it serves as a secondary centre.

Model for the Problem

(1) $\sum_{l \in L} p_{cl} = 1 \quad \forall c \in C$

(2) $\sum_{l \in L} s_{cl} = 1 \quad \forall c \in C$

(3) $p_{cl} + s_{cl} \leq 1 \quad \forall c \in C, l \in L$

(4) $d(l_1, l_2) \geq x_{l_1 t_1} \cdot x_{l_2 t_2} \cdot d\_centre\ ; l_1, l_2 \in L,\ t_1, t_2 \in T \quad s.t. \quad l_1 \leq l_2 \ \ and\ if\ l_1 = l_2\ then\ t_1 \neq t_2$

(5) The distance between a city and its primary centre cannot exceed the working distance of that centre's type.

(6) The distance between a city and its secondary centre cannot exceed three times the working distance of that centre's type.

(7) For each centre, the capacity of the centre's type cannot be exceeded by the sum of the populations of the cities it serves as a primary centre and the 10% of the populations of the cities it serves as a secondary centre.

Model for the Problem

(1) $\sum_{l \in L} p_{cl} = 1 \quad \forall c \in C$

(2) $\sum_{l \in L} s_{cl} = 1 \quad \forall c \in C$

(3) $p_{cl} + s_{cl} \leq 1 \quad \forall c \in C, l \in L$

(4) $d(l_1, l_2) \geq x_{l_1 t_1} \cdot x_{l_2 t_2} \cdot d\_centre \,; l_1, l_2 \in L,\ t_1, t_2 \in T \quad s.t. \quad l_1 \leq l_2 \ \ and\ if\ l_1 = l_2\ then\ t_1 \neq t_2$

(5) $p_{cl} \cdot d(l, c) \leq \sum_{t \in T} x_{lt} \cdot d\_city_t \quad \forall c \in C, l \in L$

(6) The distance between a city and its secondary centre cannot exceed three times the working distance of that centre's type.

(7) For each centre, the capacity of the centre's type cannot be exceeded by the sum of the populations of the cities it serves as a primary centre and the 10% of the populations of the cities it serves as a secondary centre.

Model for the Problem

(1) $\sum_{l \in L} p_{cl} = 1 \quad \forall c \in C$

(2) $\sum_{l \in L} s_{cl} = 1 \quad \forall c \in C$

(3) $p_{cl} + s_{cl} \leq 1 \quad \forall c \in C, l \in L$

(4) $d(l_1, l_2) \geq x_{l_1 t_1} \cdot x_{l_2 t_2} \cdot d\_centre \,; l_1, l_2 \in L,\, t_1, t_2 \in T \quad s.t. \quad l_1 \leq l_2 \;\; and\; if\; l_1 = l_2\; then\; t_1 \neq t_2$

(5) $p_{cl} \cdot d(l, c) \leq \sum_{t \in T} x_{lt} \cdot d\_city_t \quad \forall c \in C, l \in L$

(6) $s_{cl} \cdot d(l, c) \leq 3 \cdot \sum_{t \in T} x_{lt} \cdot d\_city_t \quad \forall c \in C, l \in L$

(7) For each centre, the capacity of the centre's type cannot be exceeded by the sum of the populations of the cities it serves as a primary centre and the 10% of the populations of the cities it serves as a secondary centre.

Model for the Problem

(1) $\sum_{l \in L} p_{cl} = 1 \quad \forall c \in C$

(2) $\sum_{l \in L} s_{cl} = 1 \quad \forall c \in C$

(3) $p_{cl} + s_{cl} \leq 1 \quad \forall c \in C, l \in L$

(4) $d(l_1, l_2) \geq x_{l_1 t_1} \cdot x_{l_2 t_2} \cdot d\_centre\ ; l_1, l_2 \in L,\ t_1, t_2 \in T \quad s.t. \quad l_1 \leq l_2 \ \ and\ if\ l_1 = l_2\ then\ t_1 \neq t_2$

(5) $p_{cl} \cdot d(l, c) \leq \sum_{t \in T} x_{lt} \cdot d\_city_t \quad \forall c \in C, l \in L$

(6) $s_{cl} \cdot d(l, c) \leq 3 \cdot \sum_{t \in T} x_{lt} \cdot d\_city_t \quad \forall c \in C, l \in L$

(7) $\sum_{c \in C} p_{cl} \cdot p_c + 0.1 \cdot \sum_c s_{cl} \cdot p_c \leq \sum_{t \in T} x_{lt} \cdot cap_t \quad \forall\ l \in L$

Heuristic Algorithms

GREEDY, LOCAL SEARCH & GRASP

Greedy Algorithm - Pseudocode

Algorithm COSTFUNCTION $(c, l, T, assignation\_type)$
Input: A city c, a location l, the set of types of centres and the type of assignation (Primary or Secondary)
Output: The cost increment of assigning the city to the location.

```
if the location l contains a facility, then:
    Check if the assignment is feasible with the current facility (range, capacity considering the
    type of assignation...)
    if feasible then:
        cost ← 0
    otherwise
        Try to upgrade the facility type
        if there is a feasible one in T, then:
            cost ← cost facility type - cost previous facility type
        otherwise
            return Infeasible
otherwise
    if you can place a facility in the location (d_center), then:
        Pick the facility with the smallest cost that is feasible (range, capacity...)
        cost ← cost facility type
    otherwise
        return Infeasible
return cost
```

Algorithm CONSTRUCTIVE$(C, L, T, assignation_type)$
Input: The set of cities C, the set of locations L, the set of types of centres T and the type of assignation (Primary or Secondary)
Output: the assignation of primary or secondary centre.

```
for each city c do:
    min_cost ← Infinity
    for each location l do:
        cost ← COSTFUNCTION (c, l, T, assignation_type)
        if cost < min_cost then:
            min_cost ← cost
            optimal_choice ← l
    Assign to c the optimal_choice as primary/secondary centre
return all the assignments
```

Local Search Pseudocode

Algorithm LOCALSEARCH($assignation,\ mode$)

Input: The assignation given by the greedy algorithm and the criterion, $mode$, used to improve the solution. The possible criterions are FIRSTIMPROVEMENT (Use the first candidate that is found to make the improvement) or BESTIMPROVEMENT (Look for all the candidates to make an improvement and choose the best of them).

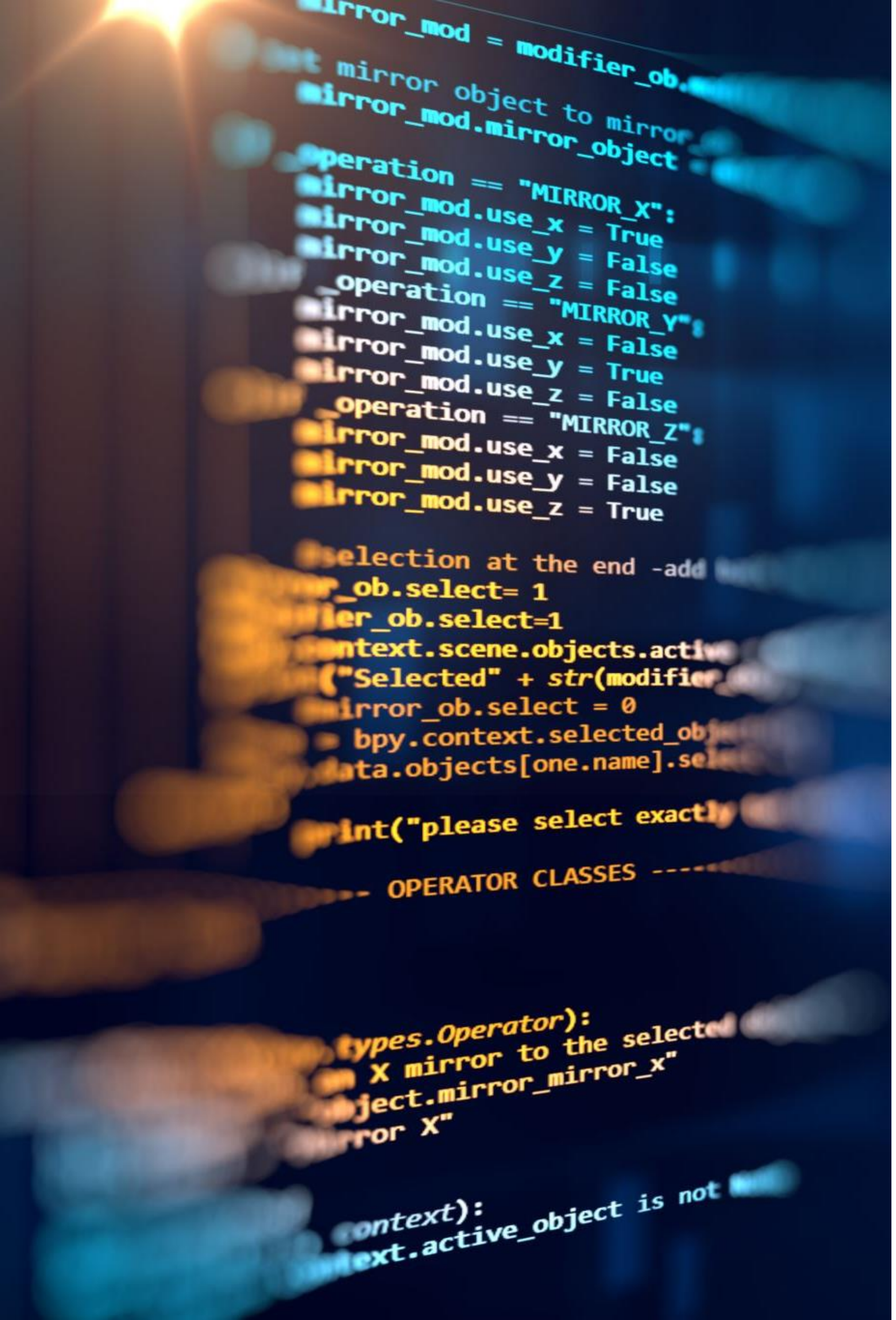
Output: An improvement in the assignation.

```
while there has been an improvement do:
    for each facility f used in the assignation do:
        for each city c assigned to the facility f (as primary or secondary) do:
            if the facility can be downgraded (improving its cost) by removing c then:
                for the rest of facilities f' do:
                    if c can be assigned to f' then:
                        if mode = FIRSTIMPROVEMENT then:
                            Update the solution with this reassignment and facility downgrade
                            break
                        if mode = BESTIMPROVEMENT then:
                            candidates ← candidates ∪ {f'}
                If mode = BESTIMPROVEMENT then:
                    Pick the candidate with smallest cost and upgrade the solution.
```

GRASP Algorithm - Pseudocode

Algorithm GREEDYRANDOMIZEDCONSTRUCTION$(C, L, T, assignation_type, \alpha)$
Input: The set of cities C, the set of locations L, the set of types of centres T, the type of assignation (Primary or Secondary) and the factor α which defines the margin applied in the construction of the RCL.
Output: A possible assignation
for each city c in C **do:**
  Build the restricted candidate list (RCL)
  for each location l in L **do :**
    CostList ← CostList ∪ { COSTFUNCTION $(c, l, T, assignation\_type)$}
  min ← min(CostList)
  max ← max(CostList)
  RCL ← $\{l \in L : min \leq cost \leq min + \alpha \cdot (max - min)\}$
  Assign to c a location chooses u.a.r. from RCL
return all the assignations

Algorithm GRASP$(C, L, T, assignation_type,\ mode, \alpha, time_limit, iterations\ _limit)$
Input: The set of cities C, the set of locations L, the set of types of centres T, the type of assignation (Primary or Secondary), the criterion for the local search $mode$,the factor α which defines the margin applied in the construction of the RCL, the maximum limit of time for the algorithm $time\_limit$ and the maximum iterations without an improvement $iterations\ \_limit$.
Output: A solution to the problem assignation
while it is not exceeded the $time\_limit$ or the $iterations\ \_limit$ **do:**
  $assignation$ ← GREEDYRANDOMIZEDCONSTRUCTION$(C, L, T, assignation_type, \alpha)$
  $assignation$ ← LOCALSEARCH$(assignation,\ mode)$
return $assignation$



IMPLEMENTATION

```haskell
data City = City
  { _cId :: Id,
    _cLocation :: Location,
    _cPopulation :: Population
  }
  deriving stock (Show, Eq, Ord, Generic)

-- | Logistic Center Type
data FacilityType = FacilityType
  { _dCity :: Distance,
    _cap :: Int,
    _cost :: Cost
  }
  deriving stock (Show, Eq, Ord, Generic)

data Problem = Problem
  { _cities :: [City],
    _facilitiesLocation :: [Location],
    _facilityTypes :: [FacilityType],
    _dCenter :: Distance
  }
  deriving stock (Show, Eq)
```

```haskell
data Facility = Facility
  { _fLocation :: Location,
    _fType :: FacilityType,
    _tier :: Tier
  }
  deriving stock (Show, Eq, Ord, Generic)

data Assignment = Assignment
  { _primary :: Facility,
    _secondary :: Facility
  }
  deriving stock (Show, Eq, Generic)

newtype Solution = Solution
  { _assignments :: Map City Assignment
  }
  deriving stock (Show)
```

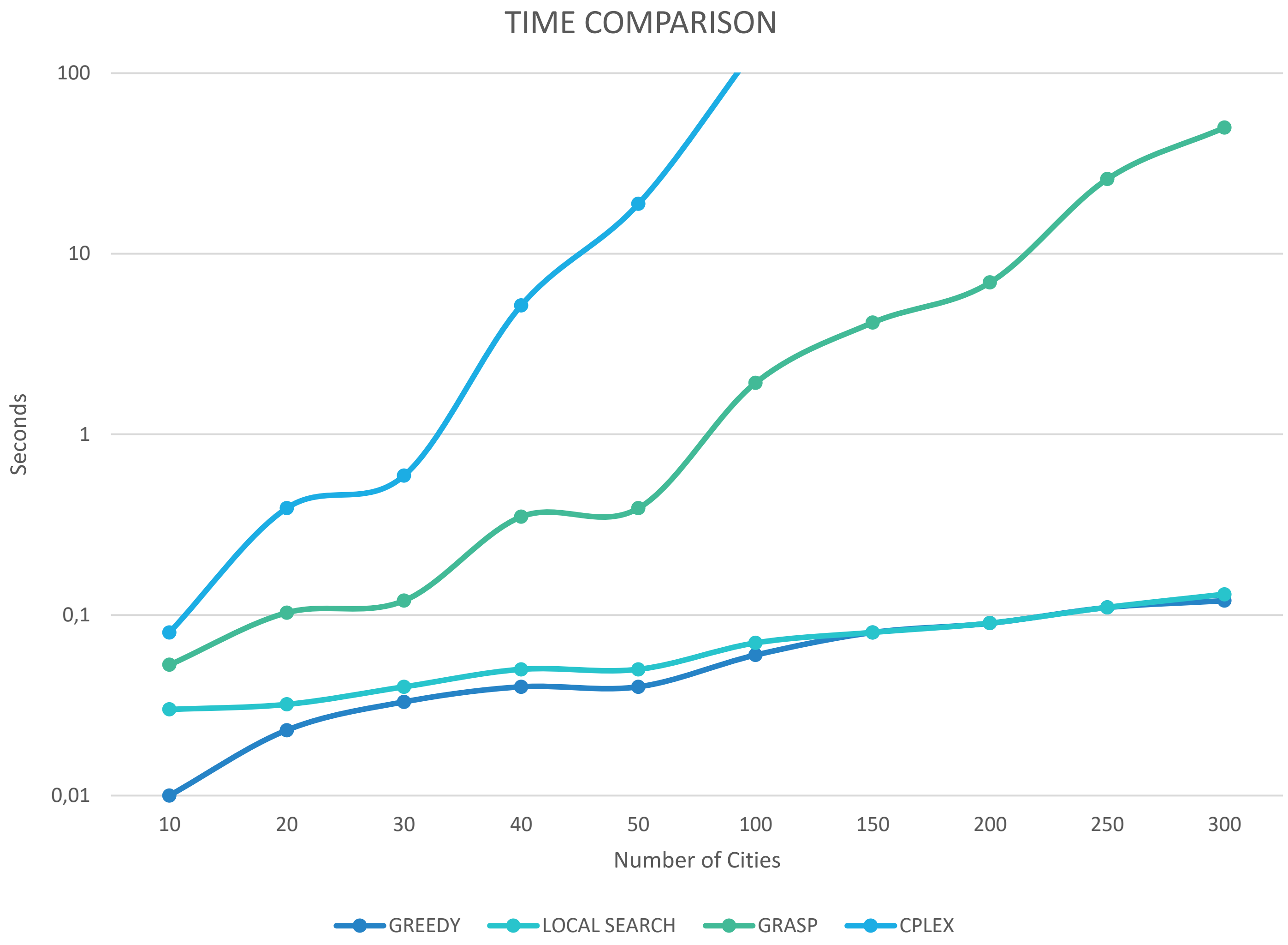
```haskell
-- | Heuristic Algorithms
runAlgorithm' :: Problem -> Algorithm -> IO (Maybe Solution)
runAlgorithm' problem algorithm = do
  gen <- R.createSystemRandom
  t1 <- getSystemTime
  evalStateT loop (GRASPState Nothing t1 gen 0)
  where
    loop :: GRASPMonad (Maybe Solution)
    loop = do
      r <- liftIO . run computation =<< use gsGen
      updated <- updateBest r
      let f = if updated then const 0 else (+ 1)
      it <- (gsIterations <<%= f)
      let hasImprovedRecently = it < maxIterationsWithoutImprovement
      timeLimitReached <- checkTimeLimit
      let stop = not hasImprovedRecently || timeLimitReached
      if stop
        then (fmap fst) <$> use gsBest
        else loop
```

```haskell
computation :: RTS m => m ()
computation = forM_ sortedCities $ \c -> do
  primary <- assignCandidate Primary c locations
  void $ assignCandidate Secondary c (filter (/= primary) locations)

run problem gen = do
  let config = Config minDistLoc opts alpha gen
  r <- try @Infeasible $ runReaderT (execStateT problem def) config
  case r of
    -- Infeasible solution
    Left _ ->
      return Nothing
    -- Feasible solution
    Right pa -> do
      case algorithm of
        Greedy Nothing ->
          return . Just $ toSolution pa
        Greedy (Just strategy) ->
          return . Just . toSolution . runLocalSearch strategy opts $ pa
        GRASP _ _ ->
          return . Just . toSolution . runLocalSearch FirstImprovement opts $ pa
```

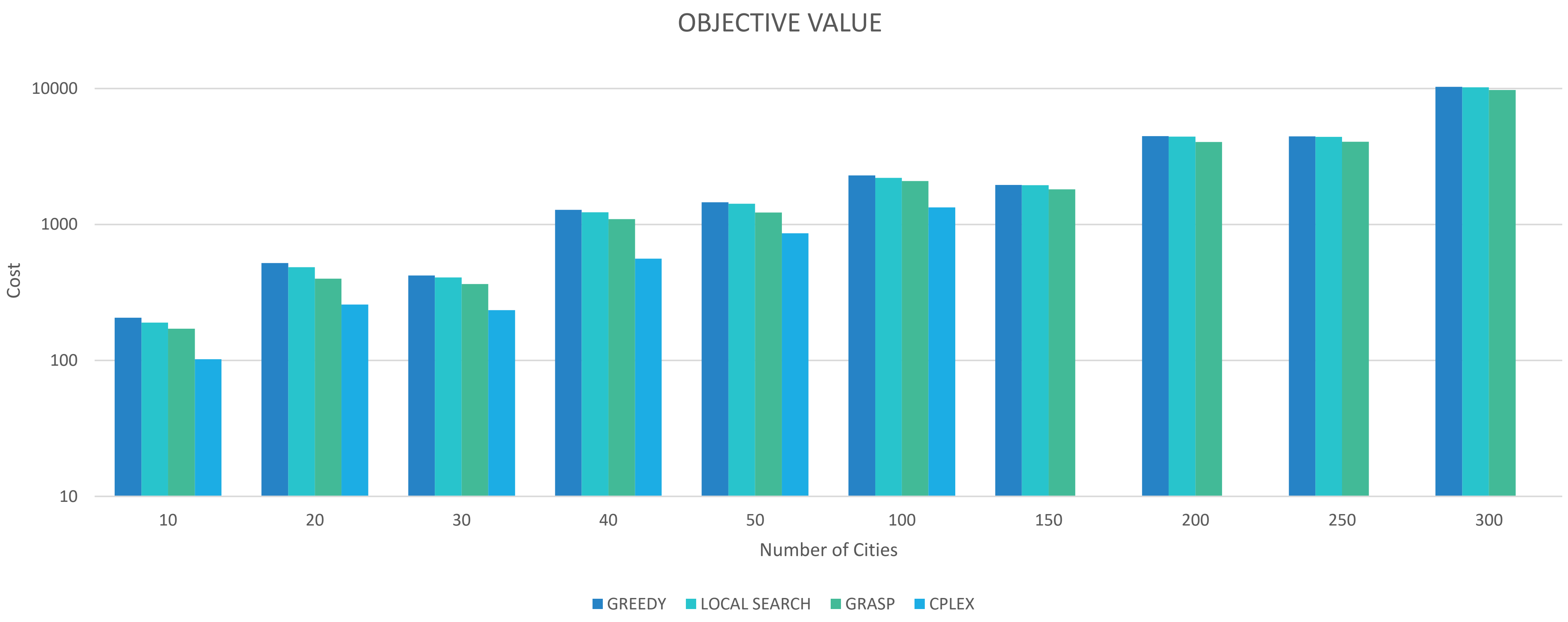
```haskell
-- | Given a city c computes the best assignment wrt the current partial solution.
-- The parameter \( \alpha \) controls the randomization of the  constructive part.
assignCandidate ::
  RTS m =>
  Tier ->
  City ->
  [Location] ->
  m Location
assignCandidate t c locations = do
  w <- get
  (Config minDistLoc ft _ _) <- ask
  let solutions = catMaybes (computeCost t minDistLoc w c ft <$> locations)
  rcl <- computeRCL solutions
  if rcl ^. candidates . to null
    then liftIO $ throwIO Infeasible
    else do
      (_, candidate, updatedAssignment) <- chooseCandidate rcl
      put updatedAssignment
      return candidate
```

Results and Conclusions



Comparative Results

Comparative Results



Possible Future Works

Refine the implemented algorithms

Add new strategies to the local search.

Parallelise the Algorithms.

QUESTIONS?

THANK YOU FOR ATTENDING TO THIS TALK