# Improving Quality and Efficiency in Home Health Care with Constraint Logic Programming

Massimiliano Cattafi  Marco Gavanelli
Maddalena Nonato
Università di Ferrara

Rosa Herrero  Juan José Ramos
Universitat Autònoma de Barcelona

Federico Malucelli
Politecnico di Milano

# Outline

1. Home Health Care scenario
2. Modeling
   - The first prototype
   - A set variables based model
   - A 'bin-packing' based model
3. Integrating OR based TSP in CLP
   - Travelling Salesman Problem
   - TSP constraint
   - Solving TSP with CLP
   - Integrating in CLP
   - Solving TSP with Lagrangian Metaheuristic
4. Search in the set based model
   - Symmetry Breaking Labeling
   - Threshold Heuristic
   - Clique Partitioning

# Outline

# Home Health Care



- Some categories of patients can stay at home instead of being hospitalized full time
  - e.g. when only frequent blood tests are required
- Nurses visit patients at their homes and provide treatments on place
- More space at hospital for those who actually have to stay there
- Reduced expense for healthcare provider
- Increased comfort for patients

# Challenges in HHC



- Decentralization introduces new factors
- Nurses need to travel (routing)
- Nurses are scattered around (rostering complexity increases)
- Requirements change (new treatments or new patients added, old ones removed), sometimes suddenly (sick leaves of personnel)

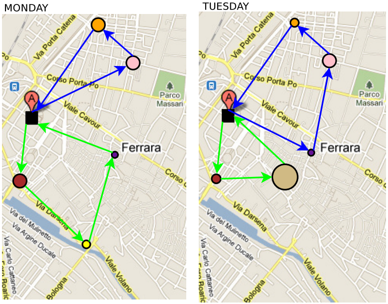
# Ferrara ASL and HHC



- 12 nurses
- Around 60 patients and 150 services per day
- Services lasting from 15 to 60 minutes
- Often more than one service per patient per day
- Nurses currently assigned 'by hand' according to zone subdivision of served area

# Goal

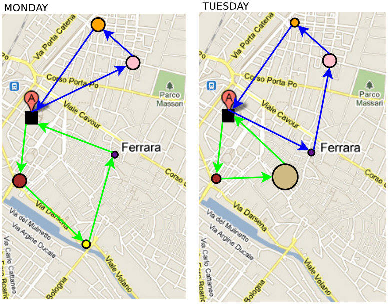
- Build a full optimized scheduling and routing plan, i.e.:
- For each day of the week:
  - Assign daily service schedule to each nurse
  - Suggest an order visiting patients which minimizes travel time (Traveling Salesman Problem)
- Some analogies with [Period Capacitated] Vehicle Routing Problem [Toth & Vigo, 2001; Christofides & Beasley, 2006]



# Requirements



- Constraints:
  - Therapeutic schedule must be followed
  - Day workload (including travel time) limited by contract
- Optimization:
  - Balanced week workloads
  - ‘Fixed’ assignment (as much as possible) of nurses to patients (‘loyalty’)



# Potential impact

*LAPS CARE, a system that was developed in 2002, uses operations research modeling to eliminate the manual planning of home care unit assignments. More than 200 units/organizations in Swedish municipalities use LAPS CARE each day to plan staff scheduling and routing for 4,000 home care workers. The system has increased operational efficiency by 10-15 percent; this corresponds to an annual savings of 20-30 million euros. In addition, the quality of home care for elderly citizens has improved.*

[Eveborn & al., 2009]

# Outline

# Input



- Services, terms whose structure is:

```
service(treatment,patient,duration,delta).
```

- For example:

```
service(28,1,52,2).
service(28,1,52,2).
...
service(30,3,22,1).
```

- Additional parameters

```
days_per_week(DaysPerWeek).
n_nurses(NNurses).
minutes_per_day(MinutesPerDay).
```

# Decision variables

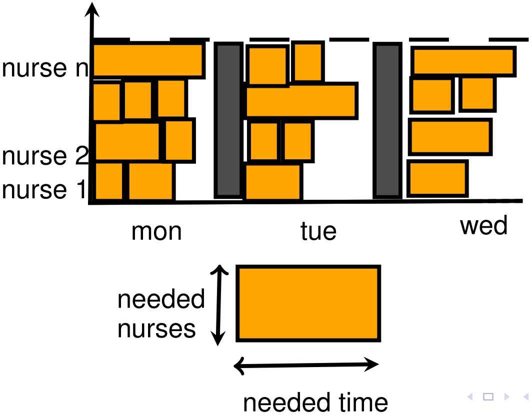
- List of assignments, terms whose structure is:

```
assignment(
  treatment,patient,duration,delta,nurse,day
).
```

- Each service has to be assigned to a specific nurse (field nurse) in a specific day (field day)
- In a week: *NServices* services, *NNurses* nurses available, *DaysPerWeek* days
- List *Nurses* of length *NServices*, variables with domain 1..*NNurses*
- List *Days* of length *NServices*, variables with domain 1..*DaysPerWeek*

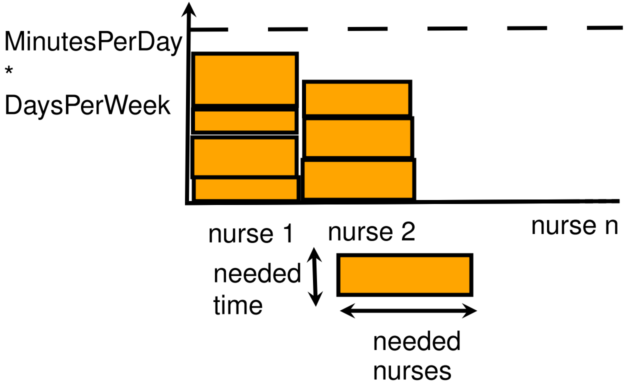
# Cumulative (1/3): all the nurses on all the weekdays

- *NNurses* on resources axis, *MinutesPerDay* $*$ *DaysPerWeek* on time one
- Dummy activities to avoid dayworkload overlap
- Not considering travel time (same applies to following kinds)



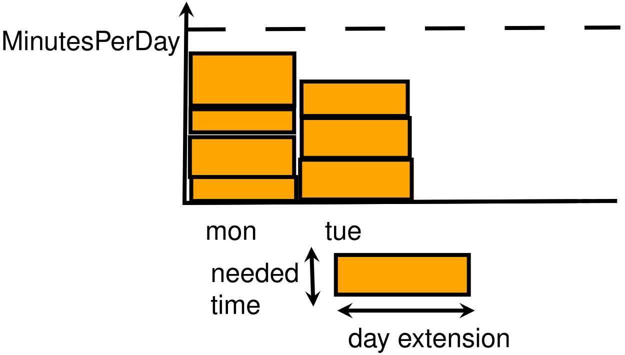
# Cumulative (2/3): all the nurses on the whole week

- Analogously:



# Cumulative (3/3): weekly schedule for each nurse

- For each nurse, *MinutesPerDay* resources every day, *DaysPerWeek* days on time axis
- We need to already know which services are assigned to the respective nurses' weekly schedules



# Spacing in therapeutic scheduling (1/2)

- Some services need to be provided more than once per week with a certain minimum of days (delta) between them

```
spacing([],[]).
spacing([Service|Services],[_|Days]):-
    Service = service(_,_,_,Delta),
    var(Delta),!,
    % this service doesn't need spacing
    spacing(Services,Days).
spacing([Service|Services],[Day|Days]):-
    spacing_loop(Service,Services,Day,Days),
    spacing(Services,Days).
```

# Spacing in therapeutic scheduling (2/2)

```
spacing_loop(_,[],_,[]).
% Same service => impose distance Delta
spacing_loop(Service,[Service|Services],
    Day1,[Day2|Days]):-!,
    Service = service(_,_,_,Delta),
    Day1+Delta #=< Day2,
    spacing_loop(Service,Services,Day1,Days).
% Otherwise, recurse
spacing_loop(Service,[_|Services],
    Day1,[_|Days]):-
    spacing_loop(Service,Services,Day1,Days).
```

# Loyalty (1/2)



- Loyalty as a penalization function
- For each patient, count the number of distinct nurses providing them a service (total function is the sum)

```
loyalty([],[],[0]).
loyalty([Service|Services], [Nurse|Nurses],
    [Count|Loyalties]) :-
    Service = service(_,P,_,_),
    same_index_del(Services,2,P,_,ServiceRest,
      Nurses,NursesPatient,NursesRest),
    count_different([Nurse|NursesPatient],Count),
    loyalty(ServiceRest,NursesRest,Loyalties).
```

# Loyalty (2/2)



- Each set has all the Nurses as upper bound
- # is cardinality constraint

```
count_different([_],1):-!.
count_different(L,Count):-
    n_nurses(NNurses),
    % we actually use intset/3
    Set :: [] ..[1,2,...,NNurses],
    all_members(L,Set),
    #(Set,Count).

all_members([],_).
all_members([H|T],Set):-
    H in Set,
    all_members(T,Set).
```

# Optimization and search (1/2)



- To obtain balanced weekly workloads, minimize the maximum
- Objective function: sum of loyalty (penalty) and biggest weekly workload

```
maxlist(WorkLoadNurses,MaxWorkLoad),
F #= LoyaltyFun+MaxWorkLoad,
minimize((
  labeling(Nurses),
  dvar_range(LoyaltyFun, LoyaltyFun, _),
  % impose the cumulative 3
  scheduling_per_nurse(NNurses,Assignment,
    MinutesPerDay),
  labeling(Days),
  workload(Services,Days,Nurses,NNurses,
    DaysPerWeek,WorkLoadMat,MinutesPerDay),
  sum_lists(WorkLoadMat,WorkLoadNurses)),
  F).
```

# Optimization and search (2/2)



- Before assigning the services to specific days we can compute a bound on the week workloads
- Lower bound on travel time imposing that all the activities are scheduled in one day (only one TSP instead of multiple ones)

```
...
scheduling_per_nurse(...),

workload(Services,Ones,Nurses,NNurses,
  DaysPerWeek,WorkLoadMatBound,_),
sum_lists(WorkLoadMatBound,WorkLoadNursesBound),
maxlist(WorkLoadNursesBound,LB),
MaxWorkLoad #>= LB,

labeling(Days)
...
```

# Problem: 'split' workload

- Assignment of services to days necessary to calculate travel time
- Week workload depends on both labelings

```
workload(Services,Days,Nurses,NNurses,
  DaysPerWeek,WorkLoadMat,MinutesPerDay):-
  %(for each Nurse,
  %for each WorkloadNurse in WorkloadMat do
  %  (for each Day,
  %    for each WorkLoad in WorkLoadNurse do
% find the services assigned to that nurse in that days
% sum their service times and put the sum in ServiceTime
% put the visited patients (without duplicates) in ListP
      solve_tsp(ListP, TravelTime),
      Workload #= TravelTime + ServiceTime,
      Workload #=< MinutesPerDay
  %  )
  %).
```

# Problem: symmetries



- To our purpose, nurses are indistinguishable, swapping the entire weekly schedule of two nurses we obtain an equivalent solution (permutations: many)

| Nurse | Mon | Tue | ... |
|---|---|---|---|
| 1 | 1, 4, 9 | 26, 45 | ... |
| 2 | 6, 7 | 52, 75, 90 | ... |
| ... | ... | ... | ... |

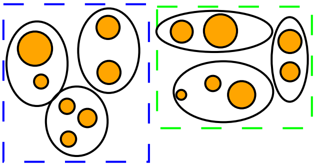
| Nurse | Mon | Tue | ... |
|---|---|---|---|
| 1 | 6, 7 | 52, 75, 90 | ... |
| 2 | 1, 4, 9 | 26, 45 | ... |
| ... | ... | ... | ... |

- Under some conditions (e.g. respecting spacing) the same applies to day swapping
- The assignment of services to nurses and days shouldn't be tightly coupled to the order of the services *in* each day: the order of patients is decided by the TSP, and it's irrelevant in which order services to the same patient in the same day are provided

# Set variables based model



- Partition the set of services into $NNurses * DaysPerWeek$ subsets
- Making sure each subset is feasible (with respect to *MinutesPerDay*), taking *DaysPerWeek* ones for each nurse we surely obtain a feasible (wrt to *MinutesPerWeek*) weekly schedule
- Some symmetries are eliminated (order of services into subsets and, at this level, of subsets into weekly schedules, are not considered)
- $NNurses = 2$, $DaysPerWeek = 3$:



# Subsets and constraints on workload

```
%(for each ElementOfSOS in SubsetsOfServices do
  ElementOfSOS subset SetOfServices,
  weight(ElementOfSOS, DurationArray, ServiceTime),
  TravellingTime #>= 0,
  traveltime(ElementOfSOS, PatientArray, TravellingTime),
  TotalTime #= ServiceTime  + TravellingTime
%)
all_disjoint(SubsetsOfServicesList),
all_union(SubsetsOfServicesList, SetOfServices),
```

# Constraints on therapeutic scheduling (1/2)

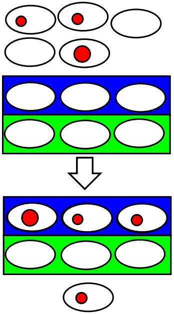


- Services to the same patient which require spacing can't be in the same subset
- Associate each service (which can have spacing issues) with a variable which represents the identifier of the subset in which the service is
- For each group of services which can't be in the same subset, impose that their subset identifiers are different
- Further: impose order (to eliminate some symmetries)

# Constraints on therapeutic scheduling (2/2)



- We don't want a patient to be visited more than once (i.e. by different nurses) in the same day
- Necessary condition is that each patient is contained in *DaysPerWeek* or less subsets
- Example with $NNurses = 2$, $DaysPerWeek = 3$ and same patient in 4 subsets:



# Symmetry breaking constraint

- Swapping two subsets, still same partitioning

```
[[1, 2], [3, 4] ...]
[[3, 4], [1, 2] ...]
```

- Lexicographic order constraint
- Order key is sum of service identifiers

```
[[1, 2], [3, 4]...] -> [3, 7, ...] (ok)
[[3, 4], [1, 2]...] -> [7, 3, ...] (no)

%(for each ElementOfSOS in SubsetsOfServicesList,
% for each ElementOfSum in SumOfServices do
    weight(ElementOfSOS, SetOfServicesArray,ElementOfSum)
%)
ordered(>=,SumOfServices),
```

# Problem: sets not as developed as fd



- Example: propagation in global constraints

```
A::[]..[1,2,3], weight(A, w(1,1,100), 1).

A = A{([] .. [1, 2, 3])
...
Yes

A::[]..[1,2,3], bpweight(A, w(1,1,100), 1).

A = A{([] .. [1, 2])
...
Yes
```

- In ECLiPSe no support for symmetry breaking during search (sbds)
- No equivalent of *search*/6 generic predicate

# A 'bin-packing' based model



- Each subset could be seen as a bin
- Objective, however, is not minimizing number of bins
- *bin_packing* constraint [Shaw, 2004] available in recent versions of ECLiPSe

# Cumulative for bin packing (1/2)

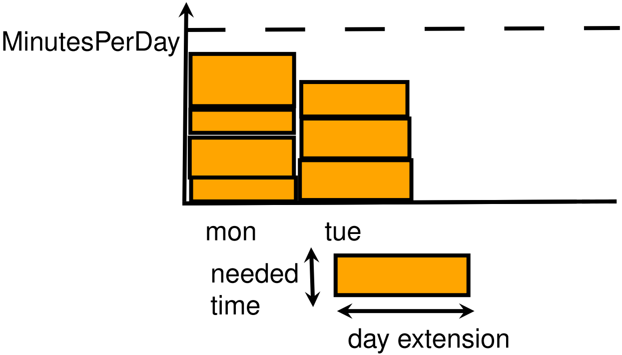


## Cumulative

$cumulative(+StartTimes, +Durations, +Resources, ++ResourceLimit)$
StartTimes: List of start times for tasks (integer variables or integers)
Durations: List of duration for tasks (integer variables or integers)
Resources: List of resource usages by tasks (integer variables or integers) ResourceLimit: Maximum amount of resource available (integer)

# Cumulative for bin packing (2/2)






- No information on each column (bin) resource usage (occupied space)
- Variable item size is allowed

# Bin packing constraint



## Bin packing

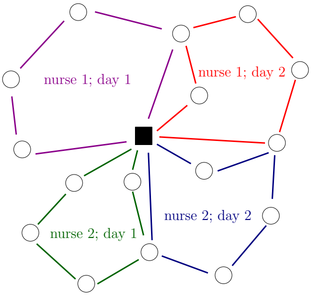
$bin\_packing(+Items, ++ItemSizes, +BinLoads)$
packing M items into N bins, each bin having a load
Items: A collection of M variables or integers (domain/value between 0 and N)
ItemSizes: A collection of M non-negative integers
BinLoads: A collection of N variables or non-negative integers

- Information on load of each bin
- Items' sizes are fixed (can't add travel time as activity of variable duration)
- Constraint still under development

# Outline

# Travelling Salesman Problem (TSP)

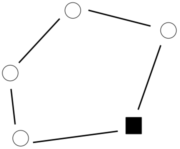
- All patients must be visited once
- Route must start and finish at the hospital
- Minimizing travel time



# Travelling Salesman Problem (TSP)



- All patients must be visited once
- Route must start and finish at the hospital
- Minimizing travel time



# TSP constraint



```
suspend(+Goal, +Prio, +CondList)
```

Suspend the Goal (a callable term) and wake it with priority Prio as soon as one of the conditions in CondList occurs.

```
:- dynamic cached_tsp/2.
cache_tsp(Nodes, Workload) :-
    cached_tsp(Nodes, Workload), !.
cache_tsp(Nodes, Workload) :-
    solve_tsp(Nodes, Workload),
    assertz(cached_tsp(Nodes, Workload)).
```

Read or add specified clause at the end of the dynamic procedure to the database.

# TSP constraint

```
constraint_on_tsp(Set, PatientSet, TravelTime) :-
    cache_tsp(PatientSet, TravelTimeLB),
    (ground(Set) ->
        TravelTime #= TravelTimeLB
    ;
        TravelTime #>= TravelTimeLB,
        suspend(
            constraint_on_tsp(Set, PatientSet, TravelTime),
            10, Set->add)
    ).
```

# Solving TSP with CLP

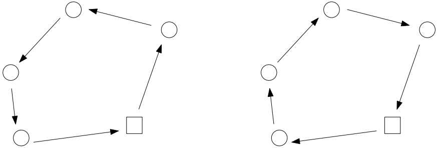
```
solve_tsp( PatientList, TravelTime):-
    length(PatientList,N),
    length(List,N),
    List #:: PatientList,
    alldifferent(List),
    append([0|List], [0], ListHospital),
    compute_travel_time(ListHospital, TravelTime),
    minimize(labeling(List), TravelTime).
```

# Solving TSP with CLP



```
solve_tsp( PatientList, TravelTime):-
    length(PatientList,N),
    length(List,N),
    List #:: PatientList,
    alldifferent(List),
    append([0|List], [0], ListHospital),
    compute_travel_time(ListHospital, TravelTime),
    minimize(labeling(List), TravelTime).
```

Symmetry breaking: from the hospital you can only go to the first half of patients



# Integrating in CLP



```
remote_connect(?Address, ?Peer, ?InitGoal)
remote_disconnect(+Peer)
```

Initiates a remote attachment at address Host/Port, the remote side must return control to the ECLiPSe side. Peer is the name of the control connection.

```
write_exdr(eclipse_to_java, Elements),
flush(eclipse_to_java),
read_exdr(java_to_eclipse, Result),
```

The term Elements is written onto the output stream eclipse_to_java. Buffered output may need to be flushed. Finally, a term Result is read from the input stream java_to_eclipse.

# Solving TSP with Lagrangian Metaheuristic



The proposed mathematical formulation for the symmetric TSP is as follows:

$$\min \sum_{e \in E} c_e x_e$$

subject to

$$\sum_{e \in \delta(i)} x_e = 2, \qquad \forall i \in I$$

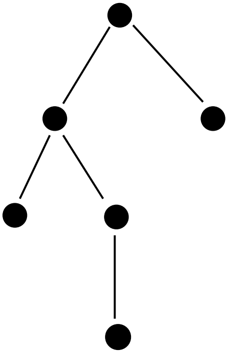
$$\sum_{e \in E(S)} x_e \leq | S | -1, \quad \forall\, S \subset I\,,\ | S | \leq \frac{1}{2} \mid I \mid$$

binary variable

$$x_e = \begin{cases} 1 & \text{if patient } j \text{ is visited immediately after } i \\ 0 & \text{otherwise} \end{cases} \quad , \ \forall e = (i,j) \in E$$

Lagrangian Relaxation [Held & Karp, 1971]

- All subcycle can be avoided constructing the solution $x$ as a 1-tree.
- A 1-tree is a tree on the graph induced by patients $\{i_1, \ldots, i_n\}$ plus two incident edges to the Hospital $i_0$.

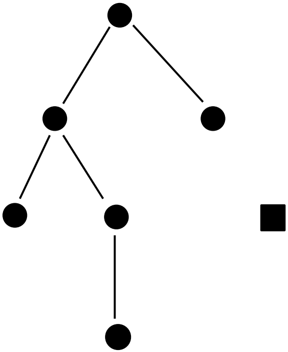


# Solving TSP with Lagrangian Metaheuristic



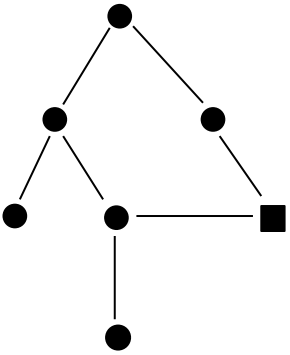
Lagrangian Relaxation [Held & Karp, 1971]

- All subcycle can be avoided constructing the solution $x$ as a 1-tree.
- A 1-tree is a tree on the graph induced by patients $\{i_1, \ldots, i_n\}$ plus two incident edges to the Hospital $i_0$.



Lagrangian Relaxation [Held & Karp, 1971]

- All subcycle can be avoided constructing the solution $x$ as a 1-tree.
- A 1-tree is a tree on the graph induced by patients $\{i_1, \ldots, i_n\}$ plus two incident edges to the Hospital $i_0$.



# Solving TSP with Lagrangian Metaheuristic



The Lagrangean Dual problem obtained from TSP is:

$$\max_{u \in \mathbb{R}^n} L(u)$$

where

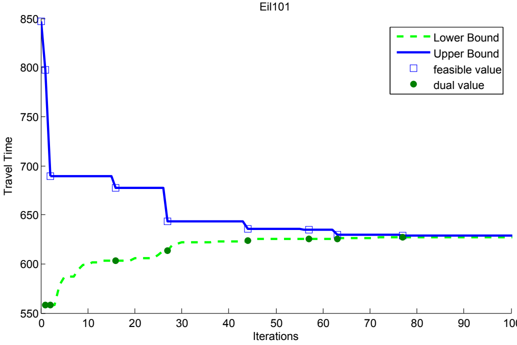
$$L(u) \quad = \quad \min_{x\ 1-tree} \sum_{e \in E} c_e x_e + \sum_{i \in I} u_i (2 - \sum_{e \in \delta(i)} x_e)$$

Its subgradient direction is $\gamma = \left(2 - \sum_{e \in \delta(i)} x_e\right)_i$

- Subgradient Optimization converges to the optimal iteratively.
- Finding a optimal solution of the dual problem is $O(n^2)$.

## Lagrangian Metaheuristic

- A heuristic obtains a feasible solution from the dual variable.
- Improving the upper bound ensures algorithm's convergence.



# Outline

# Symmetry Breaking Labeling

Symmetry Breaking on Sets:

- $Set_1 = [s_1, s_2], Set_2 = [s_3, s_4], Set_3 = [s_5]$.
- $Set_1 = [s_1, s_2], Set_2 = [s_5], Set_3 = [s_3, s_4]$.

How to avoid it within the search labeling?

# Symmetry Breaking Labeling

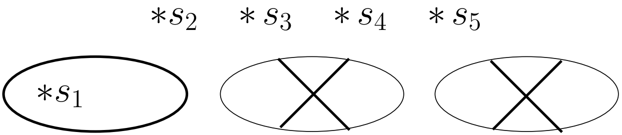


Symmetry Breaking on Sets:

- $Set_1 = [s_1, s_2], Set_2 = [s_3, s_4], Set_3 = [s_5]$.
- $Set_1 = [s_1, s_2], Set_2 = [s_5], Set_3 = [s_3, s_4]$.

How to avoid it within the search labeling?
[Meseguer & Torras, 1999]

- Assign the first service to the first set.
- Assign the next services to the opened sets or open a new one.



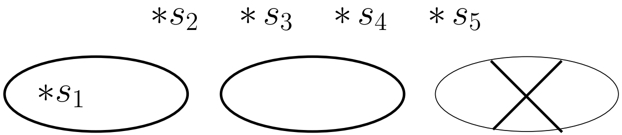
# Symmetry Breaking Labeling

Symmetry Breaking on Sets:

- $Set_1 = [s_1, s_2], Set_2 = [s_3, s_4], Set_3 = [s_5]$.
- $Set_1 = [s_1, s_2], Set_2 = [s_5], Set_3 = [s_3, s_4]$.

How to avoid it within the search labeling?
[Meseguer & Torras, 1999]

- Assign the first service to the first set.
- Assign the next services to the opened sets or open a new one.



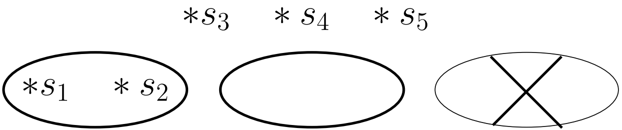
# Symmetry Breaking Labeling

Symmetry Breaking on Sets:

- $Set_1 = [s_1, s_2], Set_2 = [s_3, s_4], Set_3 = [s_5].$
- $Set_1 = [s_1, s_2], Set_2 = [s_5], Set_3 = [s_3, s_4].$

How to avoid it within the search labeling?
[Meseguer & Torras, 1999]

- Assign the first service to the first set.
- Assign the next services to the opened sets or open a new one.



# Symmetry Breaking Labeling

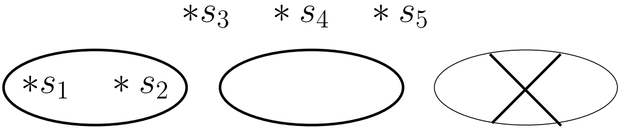


Symmetry Breaking on Sets:

- $Set_1 = [s_1, s_2], Set_2 = [s_3, s_4], Set_3 = [s_5]$.
- $Set_1 = [s_1, s_2], Set_2 = [s_5], Set_3 = [s_3, s_4]$.

How to avoid it within the search labeling?
[Meseguer & Torras, 1999]

- Assign the first service to the first set.
- Assign the next services to the opened sets or open a new one.




Advantage: reduce search effort!

# Threshold Heuristic

A service is inserted into the closest set, that is, if distance between their patients is lower than some threshold:

- Try to insert it into the closest sets.
- Try to open new one.
- Try to insert into the other ones.

# Threshold Heuristic

A service is inserted into the closest set, that is, if distance between their patients is lower than some threshold:

- Try to insert it into the closest sets.
- Try to open new one.
- Try to insert into the other ones.

Challenge: Parameter Threshold!

# Threshold Heuristic



Given a list of elements:

```
Key-(Set-Index)
```

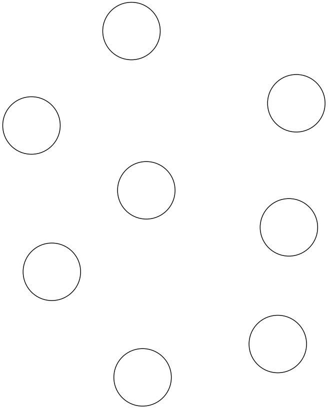
The sort is done according to the value of the Key dividing the list into two list the closest ones and the farthest ones.

```
make_sortable_dist([Set|LOpenSubs],Index,Act,
        Patient,[Key-(Set-Index)|Sortable]):-
    set_range(Set,SetLB,_),
    compute_distance_key(SetLB,Act,Patient,100000,Key),
    Index1 is Index + 1,
    make_sortable_dist(LOpenSubs,Index1,Act,
        Patient,Sortable).
```

# Clique Partitioning

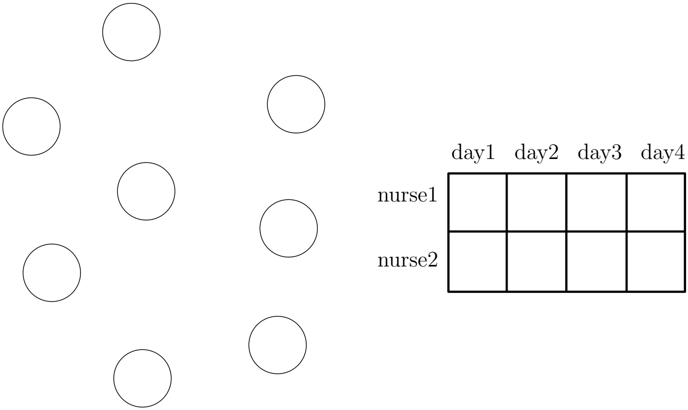


Sets of services:



# Clique Partitioning

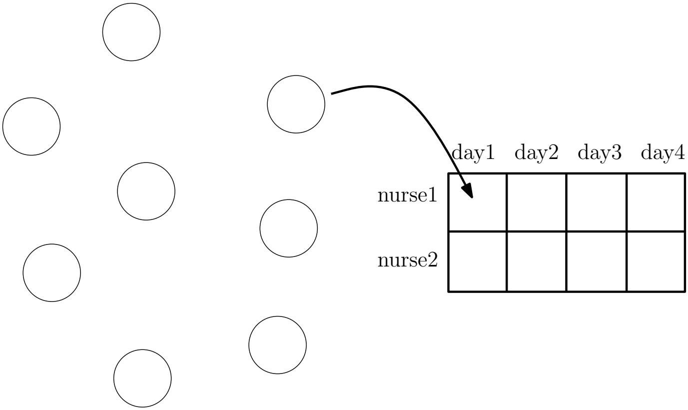
Sets of services:



# Clique Partitioning



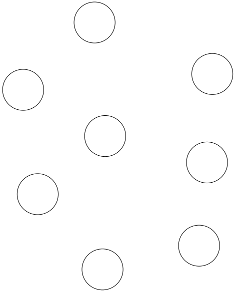
Sets of services:



# Clique Partitioning



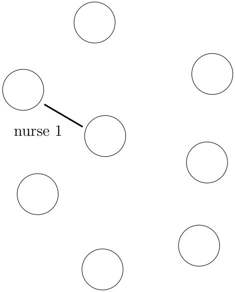
For each nurse, *Ndays* sets should be assigned.



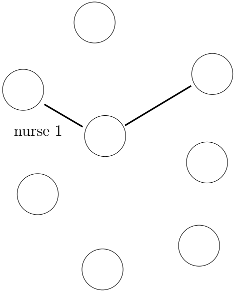
# Clique Partitioning



For each nurse, *Ndays* sets should be assigned.



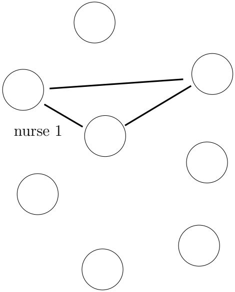
For each nurse, *Ndays* sets should be assigned.



# Clique Partitioning



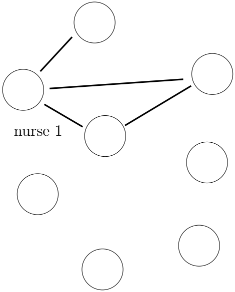
For each nurse, *Ndays* sets should be assigned.



# Clique Partitioning

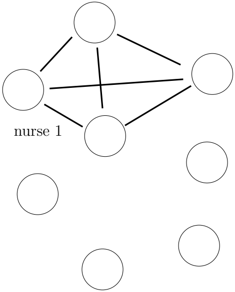


For each nurse, *Ndays* sets should be assigned.



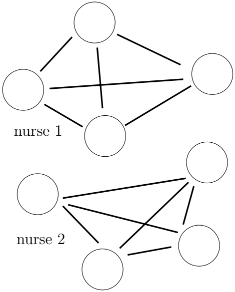
# Clique Partitioning

For each nurse, *Ndays* sets should be assigned.



# Clique Partitioning

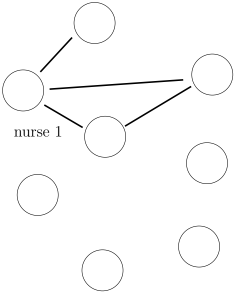
For each nurse, *Ndays* sets should be assigned.




This assignment can be seen as a variation of Clique Partitioning, [Grötschel & Wakabayashi, 1990].

# Clique Partitioning

Variable: *Arc* is a boolean symmetric matrix having zero on the diagonal.
Arc[i,j]=1 arguments that the sets *i* and *j* are assigned to the same nurse.

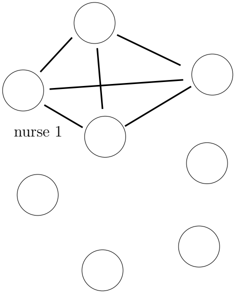



```
ArcIJ + ArcIH - ArcJH #=< 1,
```

# Clique Partitioning



Variable: *Arc* is a boolean symmetric matrix having zero on the diagonal.
Arc[i,j]=1 arguments that the sets *i* and *j* are assigned to the same nurse.

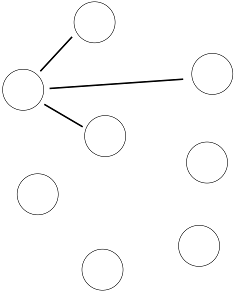



```
ArcIJ + ArcIH - ArcJH #=< 1,
```

# Clique Partitioning



Variable: *Arc* is a boolean symmetric matrix having zero on the diagonal.
Arc[i,j]=1 arguments that the sets *i* and *j* are assigned to the same nurse.

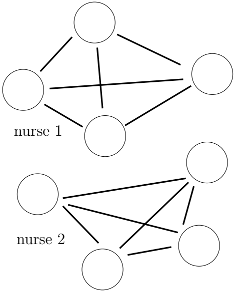


```
occurrences(1, Arc[i, 1..Nsets],  Ndays-1),
```

# Clique Partitioning



Variable: *Arc* is a boolean symmetric matrix having zero on the diagonal.
Arc[i,j]=1 arguments that the sets *i* and *j* are assigned to the same nurse.




```
occurrences(1, Arc, (Ndays-1)*Nsets),
```

# Clique Partitioning

```
suspend(
    loyalty_cell(ArcIJ, PatSetI, PatSetJ, LoyaltyIJ) ,
    5, ArcIJ->inst ),


loyalty_cell(ArcIJ, PatSetI, PatSetJ, LoyaltyIJ) :-
    (ArcIJ==0 ->
        #(PatSetI /\ PatSetJ, LoyaltyIJ)
    ;
        LoyaltyIJ #= 0
    ).
```

# Improving Quality and Efficiency in Home Health Care with Constraint Logic Programming

Massimiliano Cattafi Marco Gavanelli Maddalena Nonato
Università di Ferrara
Rosa Herrero Juan José Ramos
Universitat Autònoma de Barcelona
Federico Malucelli
Politecnico di Milano