Multi Agent Operating Room Scheduling

Noam Gaon, Yuval Gabai, Roie Zivan
Industrial Engineering and Management Department, Ben-Gurion University of the Negev, Beer Sheva, Israel
{*noamga,gabaiyuv,zivanr*}@*bgu.ac.il*

Multi Agent Operating Room Scheduling

Noam Gaon, Yuval Gabai, Roie Zivan
Industrial Engineering and Management Department, Ben-Gurion University of the Negev, Beer Sheva, Israel
{*noamga,gabaiyuv,zivanr*}*@bgu.ac.il*

Abstract

The schedule of the operations in a large hospital is performed jointly by several groups of people, each with its objective and its constraints. It is a two-phase process, starting with the allocation of operating rooms to wards, followed by the scheduling of operations in each operating room of the hospital at each day. The final schedule must satisfy all inter ward hard constraints, such as the allocation of anesthetists, nurses, and equipment to operations that are taking place in parallel, and preferably address soft constraints such as taking urgency and complexity of operations under consideration.

Besides the assignment of the operation request and surgeons to a time-slot in an operation room, the final daily schedule involves the assignment of anesthetists, nurses, and required equipment. All of these are shared resources among all the surgical wards.

We contribute to the ongoing effort of adapting multi-agent optimization models and algorithms to real world applications by modelling the problems in both phases as distributed constraint optimization problems (DCOPs), with different properties. The first, includes partial cooperative agents representing wards, allocating operating rooms for daily usage among themselves. In the second, ward representing agents interact with agents representing constraining elements, in order to generate daily operation schedules for each operating room, thus forming a unique bipartite constraint graph. On one side are the wards representatives, while on the other are the constraining resource representative agents. Each agent has a non-trivial local problem to resolve, and its solution serves as the proposed assignment in the distributed algorithm.

We discuss the properties required from the solving algorithms of the two phases, adjust existing distributed partial cooperative and local search algorithms to solve these problems, and compare algorithms implementing different approaches. Our results in both phases emphasize that successful collaboration requires that agents hold consistent information regarding their peers' states and that the degree of exploration the algorithm implements must be restricted in order to produce high quality solutions.

Keywords: Distributed Constraint Optimization, Multi-Agent System, Multi-Agent Applications, Operating-Room Scheduling, Distributed Local Search Algorithms

1. Introduction

For many years, the ongoing study and development of multi agent optimization models and algorithms considered abstract random problems such as random uniform constraint graphs, graph coloring and scale-free networks. Researchers in this community emphasized the importance of identifying practical applications in which it is essential to apply such models and algorithms many times in the last two decades. The emerge of IoT-related research has produced a set of relevant applications in which devices interact, and therefore, distributed models and algorithms are adequate for solving them [12, 41]. Yet, much of the distributed problem-solving motivation has been the human interests, represented by the system's agents, such as privacy and cooperation intentions. Thus, practical applications that include human representing agents are still uncommon, and the introduction of such applications, is a significant relevant challenge. Many real-world scheduling problems include conflicting interests between interdependent entities. An intuitive example is the scheduling of activities that require a limited set of resources. Solving such problems requires the interaction of autonomous entities (agents), each with its objective and its constraints. The goal is that the final schedule will satisfy the hard inter-group constraints and take into consideration soft constraints as well. Some examples are the scheduling of dentist appointments, automobile fixing, job interviews, and hospital operating room scheduling. It's notable that all of these have a similar combination of properties that include a level of urgency, the requirement of experts and equipment, and a room or venue for the scheduled process to take place. Such applications have a natural distributed structure, in which each autonomous entity participating in the schedule has its requirements for optimizing its performance and private preferences and constraints.

The last scenario mentioned above, the scheduling of operations in a large hospital, will be the focus of this study. A large hospital can have several operating theatres, each with ten or more operating rooms. Teams of operating rooms have multiple professionals, each of which is scheduled. In addition to surgeons, teams include nurses, anesthetists, technicians, and more. Furthermore, operations over a single day are performed by multiple doctors belonging to numerous wards and performed on patients who need to be prepared correctly for the operation at hand [26]. Since operating theatres, their teams, the operating doctors, and the patients are involved in a crucial and very costly part of a hospital's activity, computerized and efficient management can have a significant impact [1]. However, the procedure of scheduling the time of all operations performed is a complicated and time-consuming process, not to mention cumbersome (and discouraging) for all team members involved.

Several separate groups of professionals are involved in the construction and management of the assignment of time and date and room to all operation requests. While all of these groups aim to achieve a common global goal, each has different constraints and preferences, and even different sub-goals and objectives that they aim to accomplish. Additionally, there are multiple wards, each managing its schedule of operations in the rooms per date allocated to it. Each ward has preferences regarding the days and the rooms in which its operations will take place [27]. After receiving the allocation of operation rooms at specific dates, a ward needs to assign the operation requests of its patients to the operating rooms allocated to it in specific dates and determine the daily schedule in each operating room. Daily schedules must be coordinated with schedules of other rooms such that inter constraints (i.e., the availability of specially required equipment, skilled anesthetists, etc.) are satisfied [51].

We separate the discussion to the two phases which must be performed in order for operations to be executed [15]. The first is the allocation of operating rooms to wards per day. Different wards in the hospital have different needs for operating rooms. Each room at each day, is allocated solely to a single ward. Constraints define which room can be used for which type of operation, the level of concurrency in which wards can perform operations, the preferences of each ward regarding the rooms to be allocated and days of the week, and the cardinal needs for operation rooms for each ward [3].

The second is the generation of daily schedules of operations by wards, to each room per date that was allocated to them. As mentioned above, the daily schedule of each operation room must take into consideration the available resources required for the operations performed, such as nurses, anesthetists, equipment, etc. These aspects produce a natural multi-agent problem in which the ward representing agents need to coordinate their decisions with the representatives of the constraining and shared elements. Each ward determines the daily schedule for a specific date in each operating room allocated to it on that date. The assignments of all elements that participate in the operations are considered. The resulting multi-agent optimization problem takes the form of a bipartite graph. On one side of the graph are the ward agents (WR), each representing a hospital ward. On the other side are the agents representing the constraining elements (CE), the head nurse, the anesthetist's ward, and the equipment agent (there can be others such as technicians or unlicensed assistive personnel representatives as well). Hence, agents performing this process must consider both the internal ward and medical constraints, such as the urgency of the operation and the availability of surgeons. In addition, all inter ward constraints and management preferences need to be considered, such as the required equipment and personal availability [6].

The final daily schedule involves the assignment of operation requests, surgeons and required critical equipment to time-slots and operating rooms. Even though the participants are all autonomous entities, they all belong to the same hospital. Thus, they share common goals, such as the reputation and financial success of the hospital.

Multi-agent optimization scenarios are commonly represented as distributed constraint optimization problems (DCOP) [32, 36, 39, 48, 53, 8]. When agents value the possible outcomes differently for these multi agent problems (have different constraints), the adequate model for representing the problems is the *asymmetric distributed constraint optimization problem (ADCOP)* [20, 7]. The problems at the focus of this study are indeed asymmetric. In the room per day allocation problem, different wards have different needs and obviously have different valuations on different allocations. Moreover, since all wards are part of the same hospital, they have incentives for other wards to succeed as well, thus, in this scenario agents representing wards are partial cooperative [22, 49]. In the inter ward daily schedule problem, the agents representing the constraining elements (e.g., nurses, anesthetics, technicians, etc.) and the underlying communication and constraint graph structure are unique. Different wards have different preferences, and the representatives of the constraining elements have their own interests.

ADCOPs are known to be NP-hard, and thus, the enormous size of the problem at hand rules out complete ADCOP algorithms. Consequently, the study proposes ADCOP-based models for representing the problems and distributed incomplete local search algorithms for solving them. More specifically, in this paper we advance the research on realistic implementations of multi-agent optimization models and algorithms by:

1. Proposing an extension of the socially motivated partial cooperative model (proposed in [49]), specific for periodic indivisible resource allocation, which applies to the allocation of operating rooms per date to wards in a large hospital.
2. Proposing a bipartite distributed constraint optimization model. The model is an extension of ADCOP for representing problems that include agents that attempt to schedule elements constrained by the availability of resources. These resources are in control of other agents.

All agents seek to reach a schedule that does not violate hard constraints and maximizes the utility expressed by soft constraints. Each scheduled operation includes the assignment of all elements involved. This model applies to the problem of scheduling operations in operation rooms allocated to wards on specific dates.

3. Proposing adjustments of distributed local search algorithms for solving the two models that represent the resource allocation and the scheduling problems, in which agents solve their local problem using a centralized heuristic (e.g., Simulated Annealing) and exchange assignments with their neighbors to resolve inter constraints.
4. Present experimental results that demonstrate the compatibility of the proposed algorithms for solving the real-world problem described above.

Our empirical results demonstrate the importance of shared preferences, when agents are partially cooperative. Ignorance may lead to altruistic decisions, which hurt the altruist agents more than they benefit their neighbors. On the other hand, exchanged indications regarding the preferences of agents on their neighbors' actions trigger high-quality solutions. Furthermore, the results on the daily operation scheduling indicate the importance of using means for stability and incremental improvement of interim solutions. They reveal that a limited level of exploration is required in order to achieve solutions with high quality. This exploration level can be achieved by a limited number of revisions in each algorithm iteration or by penalizing schedule revisions, such that only modifications with high benefits are performed.

The rest of this work is organized as follows: Section 2 presents the background regarding operating room planning and scheduling, distributed artificial intelligence, multi-agent systems, distributed constraint optimization problems (DCOPs), asymmetric DCOPs, algorithms for solving DCOP, and centralized local search algorithms. Next, models for representing the problems in the two phases are formalized in Section 3. Then, the algorithms description is presented in Section 4. Finally, Section 5 presents our experimental evaluations, followed by our conclusion in Section 6.

2. Background

In this section we provide background on operation scheduling, on distributed optimization problems and on local search algorithms for solving them.

2.1. Operating Room Planning and Scheduling

The directorial view of supplying health services to patients in hospitals is becoming progressively more crucial. Hospitals want to decrease costs and improve the utilization level on one hand, while maximizing the level of patient contentment on the other. The operating theatre is the hospital's main cost and revenue center [10] and has a significant influence on the hospital's performance. However, operating theatre management is challenging due to the lack of expensive resources and the participants' conflicting priorities and preferences. These emphasize the need for productivity and require the development of sufficient planning and scheduling procedures.

Advanced scheduling is the method of setting up a surgery date for a patient. Allocation scheduling on the other hand, defines the operating room and the starting time of the surgery on the day that the surgery was assigned to [28]. [37] arranges the literature on operating room scheduling based on broad areas of interest, such as cost control or the scheduling of specific resources. Two main patient groups are considered in the literature with respect to operating room scheduling:

elective and non-elective patients. The first group includes patients whose surgery can be scheduled in advance, and the second, patients that must be urgently and unexpectedly operated [6].

Former studies on surgery scheduling problems can be separated into two branches. First, the single operating room (OR) scheduling problems aim to define the start times for a set of surgeries in an OR on a given day [9, 46, 47]. Second, multiple ORs scheduling problems (including this study) [2, 24], that consider the scheduling of parallel surgeries in various ORs.

Distinct methods have been used to solve the surgery scheduling problems. These methods can be divided into four categories: queuing models, simulation methods, optimization methods, and heuristic methods [6, 11, 14]. Queuing models are typically used to solve single OR scheduling problems. Simulations can be used to evaluate several scheduling heuristics and are adjustable to model ambiguity during surgery scheduling. Concerning the optimization methods, most researchers used deterministic/stochastic integer programming/mixed-integer programming models and algorithms. Aside from exact methods, some heuristics have been applied, such as Simulated Annealing, Tabu search, and genetic algorithms. In addition, some centralized Constraint Programming (CP) approaches have been proposed for solving the surgical schedule problem [51]. All of the above require the centralization of all the problems constraints and preferences to a single entity that solves the problem, in contrast to our approach.

2.2. Distributed Artificial Intelligence and Multi-Agent Systems

Many real-world problems are distributed by nature. An interaction between autonomous entities is commonly defined by the influence of an entity on other entities' decisions and actions [4]. Such an entity is commonly referred to as an agent. An agent can be a physical or virtual autonomous entity that can act, perceive its environment (sometimes in a partial way), communicate with others, and has skills to achieve its goal and tendencies [16, 40]. A significant focus in distributed artificial intelligence (DAI) is given to the coordination between multiple autonomous agents. This coordination is described by the interaction between behaviors, knowledge, goals, skills, and programs of agents [4].

A particular environment in which a number of agents interact in order to pursue some set of goals or perform some set of tasks is also known as a Multi-agent System (MAS). MAS contain an environment, objects, agents (the agents being the only ones to act), and interactions between all entities. The agent is an autonomous entity, virtual or real, that perceives the environment. Each agent has a set of skills that allow it to execute actions in order to accomplish its goals [16]. From the point of view of a system's agent, the environment is dynamic; it changes according to the activity of other systems' agents. Systems in which several agents use interaction to maximize utility and jointly solve tasks are called Cooperative MAS [4, 45].

2.3. Distributed Constraint Optimization

Distributed Constraint Optimization Problem (DCOP) is a framework used to characterize combinatorial optimization problems that are distributed by nature and include constraints. DCOPs can represent real-life problems that cannot be resolved in a centralized way for reasons such as autonomous decisions of the agents, user's privacy, and infeasibility of centralization. They usually involve many interdependent agents, can be represented by a graphical model, and solved using message passing algorithms. DCOPs have a broad range of applications in MAS [33]. They constitute a scientific challenge because they require the cooperation of various agents (only aware of a minor component of the problem) to obtain global solutions [21].

A DCOP includes a set of agents, each holding at least one variable and a set of functions/constraints. Values assigned to the variables that agents hold are taken from finite, discrete domains. The agents interact via messages to coordinate the selection of values for their variables aiming to optimize a given global function, which is commonly to minimize/maximize the sum of costs/utilities of the set of constraints between variables. Constraints among variables that are possibly held by distinct agents, define the costs incurred or utilities derived from combinations of value assignments.

The following formal description of a DCOP is consistent with the definitions in many DCOP studies, e.g., [32]. A DCOP is a tuple $\langle \mathcal{A}, \mathcal{X}, \mathcal{D}, \mathcal{R} \rangle$. $\mathcal{A}$ is a finite set of agents $\{A_1, A_2, \ldots, A_n\}$. $\mathcal{X}$ is a finite set of variables $\{x_1, x_2, \ldots, x_m\}$. A common assumption in DCOP is that every variable is held by a single agent. ,$\mathcal{D}$ is a set of domains $\{D_1, D_2, \ldots, D_m\}$. Each domain D_i contains the finite set of values that can be assigned to the variable x_i. An assignment of value $d \in D_i$ to x_i is denoted by an ordered pair $\langle x_i, d \rangle$. $\mathcal{R}$ is a set of relations (constraints). Each constraint $C \in R$ defines a nonnegative cost for every possible value combination of a set of variables and is of the form $C : D_{i_1} \times D_{i_2} \times \ldots \times D_{i_k} \rightarrow \mathbb{R}_+ \cup \{0\}$. A binary constraint refers to precisely two variables and is of the form $C_{i_j} : D_i \times D_j \rightarrow \mathbb{R}_+ \cup \{0\}$. A binary DCOP is a DCOP in which all constraints are binary. A partial assignment (PA) is a set of value assignments to variables, in which each variable appears at most once. $vars(PA)$ is the set of all variables that appear in PA. A constraint $\mathcal{C} \in \mathcal{R}$ of the form $C : D_{i_1} \times D_{i_2} \times \ldots \times D_{i_k} \rightarrow \mathbb{R}_+ \cup \{0\}$ applies to PA if $x_{i_1}, x_{i_2}, \ldots, x_{i_k} \in vars(PA)$. The cost of a PA is the sum of all applicable constraints to PA over the assignments in PA. A complete assignment (or a solution) is a partial assignment that includes all the DCOP's variables $(vars(PA) = X)$. An optimal solution is a complete assignment with minimal cost.

DCOP can be used to characterize a wide variety of multiagent systems in which agents need to cooperate to attain a common goal. Some examples of these applications are Automatic meeting scheduling by intelligent calendars, Mobile sensor nets, IoT applications such as electronic device operation scheduling in smart homes, and resource allocations [17, 20, 30].

2.4. Asymmetric DCOP

In a DCOP, the costs incurred by all of the agents involved in each constraint are equal. Therefore, the DCOP definition cannot correctly characterize real life problems in which agents value the outcomes of decisions differently [20]. For instance, in meeting scheduling problems, agents might have separate valuations for the meeting they were summoned to, a scenario that cannot be captured by the standard DCOP model [54].

Asymmetric DCOP (ADCOP) was introduced by [20]; the new expanded framework allows each agent to hold its own valuated cost for each constraint it is involved in. ADCOPs generalize DCOPs by explicitly defining for each combination of assignments of constrained agents the exact cost for each participant in the constraint [20]. Combination of value assignments are mapped to a tuple of costs, one for each constrained agent, and each agent holds only its part of the constraint.

Formally, an ADCOP is defined by a tuple $\langle \mathcal{A}, \mathcal{X}, \mathcal{D}, \mathcal{R} \rangle$. Where $\mathcal{A}, \mathcal{X}$, and $\mathcal{D}$ are defined the same as in DCOP. Each constraint $C \in R$ of an asymmetric DCOP defines a set on nonnegative costs for every possible value combination of a set of variables and takes the following form $C : D_{i_1} \times D_{i_2} \times \ldots \times D_{i_k} \rightarrow \mathbb{R}^k_+ \cup \{0\}$. R^k_+ is a vector that contains for each agent A_j; $1 \leqslant j \leqslant k$ its cost for each combination of value assignments. This way, each agent A_j; $1 \leqslant j \leqslant k$ holds its part of the constraint $C_j, C_j : D_{i_1} \times D_{i_2} \times \ldots \times \ldots D_{i_k} \rightarrow \mathbb{R}_+ \cup \{0\}$ such that its privacy is maintained.

As in DCOP, an optimal solution to an ADCOP is a complete assignment to all variables with a minimal sum of all agent costs.

2.5. Algorithms for Solving DCOPs

In multiagent optimization algorithms, intelligent agents interact with one another by exchanging messages and sharing information concerning the problem to achieve a particular goal. While centralized solutions (in which all agents send their information to one agent, which computes the results and returns a solution) may be more efficient, problem requirements might include the necessity to be solved in a distributed way. Many of these problems are distributed by nature, imply physical restrictions, or have privacy requirements that prevent centralized problem solving [17, 19, 53].

2.5.1. Complete and Incomplete Algorithms

Solving methods can be roughly divided to two sets. The first is complete methods, which are guaranteed to find an optimal solution if one exists. In practice, since DCOPs are NP-hard [32], complete algorithms can solve many exciting problems within a reasonable amount of time, despite their discouraging worst-case guarantees. A significant obstacle for these algorithms is that it can be challenging to anticipate the time required for such algorithms to solve novel problem instances [43].

Researchers propose several complete algorithms for solving DCOPs. However, as mentioned above, DCOPs are known to be NP-Hard (with exponential worst time complexity in the number of variables). Therefore, these algorithms have restricted effective use, especially with practical applications which primarily involve significant problems with a large set of variables. Thus, there is a great incentive in developing incomplete algorithms [13, 34, 50].

Incomplete methods are not guaranteed to find optimal solutions but run quickly and efficiently enough to be applied to the mentioned realistic problems [43]. These algorithms can be divided into two major groups: inference algorithms and distributed search algorithms.

Inference algorithms (DPOP, Max-Sum, etc.) are based on the propagation of information provided by agents from the entire system. The given information is the foundation for beliefs maintained by the agents about the best cost that can be achieved. Belief propagation implies calculating beliefs based on each new data's influence concerning the constraints' costs. The beliefs are propagated through the graph via message-passing between neighboring variables [35].

Search algorithms (DSA, MGM, etc.) traverse the solution space by generating a complete assignment and performing local assignment replacements to improve it. First, agents select value assignments and share them with each other. Then, the solution is improved iteratively, applying search strategies to discover (hopefully) better assignments [21, 31, 50, 53]. The general design of most state-of-the-art local search algorithms for DCOPs is synchronous [23]. In each algorithm step (or iteration), an agent sends its assignment to all its constraint graph neighbors and receives all the neighbors' assignments. They vary in the approach agents use to determine whether to change their current value assignments to their variables [53].

Anytime algorithms hold the best assignment that is found throughout the search. In these algorithms, the anytime feature ensures that the solution's value remains the same or improves if more steps of the algorithm are performed. This property cannot be assured easily in distributed environments. In these environments, agents are only aware of the cost of their own assignment (and maybe that of their neighbors, too), but no one knows when an excellent global solution is obtained. A general framework that enhances distributed local search algorithms for DCOPs with the anytime property was proposed by [53]. The suggested framework uses a BFS-tree to accumulate the costs of the system's state during the algorithm's iterative performance and to propagate the detection of the best new state when found. The proposed framework does not involve additional network load.

2.5.2. Non-Concurrent Atomic Operations

Evaluating the runtime performance of distributed algorithms is not trivial. Agents may be performing on machines with different hardware, the implementation quality may affect the runtime as well and some of the agents' actions can be performed concurrently, while others cannot. Therefore, there is a need to establish which of the operations performed by agents could not have been performed concurrently in order to detect the sequence of implementation independent actions performed by agents. This sequence constitutes an implementation independent performance measure of distributed algorithms in a distributed environment. Thus, the run-time performance of the algorithm is the most extended non-concurrent sequence of operations that the algorithm performs. [52] suggested a uniform method for measuring and comparing their performance. The performance measure allows an evaluation of the different DCOP algorithms on a consistent scale. The straightforward concept is to measure the most extended sequence of non-concurrent atomic operations (NCLO) [33] (e.g., constraint check). This approach is the one we adopt in this study since we evaluate the quality of the solutions of the algorithms as a function of the asynchronous advancement of the algorithm when agents perform computation concurrently.

2.5.3. Distributed Stochastic Algorithm

The Distributes Stochastic Algorithm (DSA) is a simple distributed local search algorithm in which following a primary step in which agents choose a starting value for their variable (randomly), agents perform a series of steps (loops iteratively) until some termination condition is met. In every step, an agent sends its value assignment to its neighbors in the constraint graph and collects the value assignments of its neighbors. Once collecting the value assignments of all its neighbors, an agent decides whether to keep its value assignment or to modify it. This decision has a significant effect on the performance of the algorithm. If an agent in DSA cannot upgrade its current state by substituting its present value, it does not replace it. On the other hand, if it can improve (or keep the exact cost, depending on the version used), it determines whether to replace the values using a stochastic strategy.

2.6. Partial Cooperation

In contrast to early studies of ADCOPs, which assumed full cooperation by the agents [5, 21], partial cooperation models represent agents that cooperate only under some conditions. The level of cooperation (which is represented by λ) determines the reference point according to which agents intentions are modeled. In order to allow the agents to consider solutions with high global quality, which may reduce their personal utility, the parameter λ bounds the losses that an agent is willing to undertake in order to contribute to the global objective, i.e., agents perform actions only if they do not result in a cost that exceeds the maximum cost they are willing to endure. Formally, the following parameters are used by the model:

Algorithm 1 AGC

input: $baseLineAssignment_i$, $baseLineCost_i$, λ_i and

```
$value \leftarrow baseLineAssignment_i$;
$\mu_i \leftarrow baseLineCost_i$;
$localView \leftarrow null$;
send($value$) to $N(i)$;
while stop condition not met do
  PHASE 1:
    Collect all $value$ messages and update $localView$
    $\langle val_i, gain_i\rangle \leftarrow improvingAssignment()$;
    send($\langle val_i, gain_i\rangle$) to $N(i)$;
  PHASE 2:
    Collect all $\langle val_j, gain_j\rangle$ messages;
    $a_j \leftarrow$ agent in $N(i) \cup A_i$ with maximal socialGain s.t.
        $c_i(localView$ including received $val_j) \leq \mu_i \cdot (1 + \lambda_i)$;
    send($Neg$!) to $N(i) \setminus a_j$;
  PHASE 3:
    Collect $Neg$! messages;
    if did not receive $Neg$! & can improve then
      $value \leftarrow val_i$;
      send($value$) to $N(i)$;
```

Definition 1. *Denote by μ_i the* **base-line cost** *of agent A_i (i.e., the cost for agent A_i that she assumes she will pay if she acts selfishly).*

Definition 2. *The* **cooperation intention parameter** $\lambda_i \geq 0$ *defines the maximal* increase *in the value of μ_i that is acceptable by agent A_i.*

These cooperation bounds can significantly decrease the number of feasible outcomes for a distributed incomplete algorithm, as can be seen in the next definition.

Definition 3. *A* **feasible outcome** *for a distributed algorithm is defined to be any outcome (solution) o in the set of all possible outcomes O, that satisfies the following condition.*

$$O^{feasible} = \{o \in O \mid \forall A_i \in \mathcal{A}, \quad c_i(o) \leq \mu_i \cdot (1 + \lambda_i)\}$$

Where $c_i(o)$ is the cost for agent A_i in outcome o

2.7. Partial Cooperative Local Search

The Asymmetric Gain Coordination (AGC) algorithm, guarantees that the personal cost of an agent does not exceed the predefined cooperative intention limit, while constantly seeking globally improving solutions. Agents executing this algorithm exploit possible improvements until they converge to some local optimum, which cannot be further improved without breaching the cooperation bound of one of the agents. Before replacing a value assignment, an agent requests her

Algorithm 2 SM_AGC

input: $baseLineAssignment_i$, $baseLineCost_i$, λ_i and Ω_i

```
$value \leftarrow baseLineAssignment_i$;
$\mu_i \leftarrow baseLineCost_i$;
$localView \leftarrow null$;
send($value$) to $N(i)$;
while stop condition not met do
 PHASE 1:
  Collect all $value$ messages and update $localView$
  for each $A_j \in N(i)$ do
    $\pi_{i,j} \leftarrow preferences(A_j)$;
    send($\pi_{i,j}$) to $A_j$;
 PHASE 2:
  Collect all $\pi$ messages;
  $\Pi_i \leftarrow \pi_{j \in N(i)} \cup preferences(A_i)$;
  $alterVal_i \leftarrow socialImprovingAssignment(\Pi_i, \Omega_i)$;
  send($alterVal_i, socialGain_i$) to $N(i)$;
 PHASE 3:
  Collect all $\langle alterVal_j, socialGain_j \rangle$ messages;
  $a_j \leftarrow$ agent in $N(i) \cup A_i$ with maximal socialGain s.t.
        $c_i(v_j \leftarrow alterVal_j | S_t) \leq \mu_i \cdot (1 + \lambda_i)$;
  send($Neg$!) to $N(i) \setminus a_j$;
 PHASE 4:
  Collect $Neg$! messages;
  if did not receive $Neg$! & can improve then
    $value \leftarrow alterVal_i$;
   send($value$) to $N(i)$;
```

neighbors' approval, which is given only if this value assignment replacement does not cause a breach of the cooperative bound for the neighbor. Only if all neighbors approve, the agent replaces her value assignment.

The pseudo-code for the AGC algorithm is presented in Algorithm 1. It emphasizes the three phases that constitute each step of the algorithm. The algorithm begins after agents computed a baseline assignment by performing a simple non cooperative interaction between them. Thus, the agent can select her baseline value assignment and use the baseline cost as a reference point. After exchanging their value assignments the agents loop over the three phases of the algorithm until a termination condition is met, e.g., a predefined number of iterations. In the first phase, each agent selects an action (an assignment replacement) that maximizes her gain, and sends a suggestion to perform it, along with its expected gain from this action, to her neighbors. In the second, agents receiving the suggested actions of their neighbors, approve one of them that does not cause a damage that they are not willing to endure, and send Neg! messages to the rest. In the third phase, agents that did not get a Neg! message from their neighbors, perform the assignment replacement they have proposed.

2.8. Socially-Motivated Local Search

In the AGC algorithm described above, agents cooperate by approving or rejecting assignment replacements suggested by their neighbors, and thus, preserve a level of personal utility that is acceptable to them. In order to allow agents to exploit the cooperative intentions of their neighboring agents, and so to improve the solution's quality, an approach towards partial cooperative local search, in which agents take an extra step in the interaction process before selecting an assignment, was proposed in [49]. In this additional stage, agents share with their neighbors some information regarding their preferences over their assignment selection, i.e., an indication of the anticipated benefits (or costs) should the neighbors decide to change their current value assignment. After exchanging this information, agents attempt to find an alternative value assignment, taking into consideration their own preferences as well as the indications received from their neighbors. This approach was combined with the AGC algorithm, resulting in the Socially Motivated (SM) AGC [49].

Algorithm 2 includes the pseudo code of SM_AGC. Like the original AGC version, the algorithm begins after agents computed a baseline assignment and use the baseline cost as a reference point. Similar to AGC, after exchanging their value assignments the agents loop over the four phases of the algorithm until a termination condition is met.

In Phase 1, each agent, after receiving the value assignments from her neighbors, sends to each of them an indication regarding her preferences on their value assignment selection. In Phase 2, after receiving preferences indications, each agent attempts to find an alternative *social improving value assignment*, i.e., selects a value while taking into consideration her own preferences and the indications regarding the neighbors' preferences [1]. After selecting the alternative value, the agent sends it to her neighbors along with the the calculated expected social gain. Phases 3 and 4 are identical to phases 2 and 3 in AGC.

The DSA algorithm is uniform, i.e., it does not use agents' identities. It is also synchronous; in each iteration of the algorithm agents send messages to all their neighbors and wait to receive the messages sent to them before advancing to the next iteration [50].

2.8.1. Simulated Annealing

Simulated Annealing (SA) is a local search algorithm that uses an analogy to statistical mechanics in order to balance between exploration and exploitation. In metallurgy, annealing is the procedure used to soften harden metals by heating them to a high temperature and then gradually cooling them, allowing the material to arrive at a low energy crystalline state. SA exploits the idea of annealing to find the bottom cost solutions for combinatorial optimization problems [25].

SA is a state-of-the-art meta-heuristic for assessing global optimization in a sizable domain [44]. The algorithm accepts occasional transitions leading to more pricey solutions, but it avoids getting trapped in local optima. The interpretation of the gradual cooling of metal is translated in the algorithm as a gradual decrease in the probability of accepting worse solutions as it explores the solution space. Obtaining worse solutions is an essential property of meta-heuristics because it allows for a broader search for the optimal solution (explores the solution space).

At the beginning of the search, the algorithm explores the search space widely; therefore, the probability for accepting a negative transition is high. As the search continues, the changes are

[1] Ω is used for assessing the weights of the preferences of neighbors. For more details see [49]

restricted to local improvements and optimizations. The cooling schedule can be regulated by the initial system temperature, the temperature decrement function, and the number of iterations in between [44].

3. Problems Formalization

In this section we present a model for representing each of the problems at hand, followed by their implementation as an ADCOP.

3.1. Operating Room per Date Allocation to Wards

An operating room per date allocation (ORDA) problem is composed of: A set of n wards $W = \{W_1, W_2, ..., W_n\}$ and a set of m pairs of the form $\langle room, date\rangle$, $RD = \{RD_1, RD_2, ..., RD_m\}$. The atomic time unit in which a resource can be allocated in this problem is a day and the number of days in which we allocate rooms (the time horizon H) is finite. Each room date pair RD_j is assigned solely to one of the wards $W_i \in W$. Thus, an allocation of a room at some date to a ward is a pair $\langle W_i, RD_j\rangle$. A *complete allocation* CA is a set of exactly m allocation pairs, such that, each of the room date pairs RD_j ($1 \leq j \leq m$) is included exactly once in these pairs.

Each ward W_i has a cardinal constraint CC_i that defines the utility it derives with respect to the number of RDs it received in the specified time interval, and two bounds. A lower bound defines the minimal amount of RDs required in the time interval (LB_i), and an upper bound that defines the maximal number of RDs the ward can use (UB_i). These bounds define a different utility/cost scheme. An allocation that does not satisfy the lower bound incurs a high cost. It can be a fixed cost or related to the amount of RDs allocated. The upper bound (UB_i) defines the number of RDs allocated to ward W_i such that if it is allocated an additional RD, there is no increment to its utility.

The utility that a ward W_i derives from a complete allocation CA is denoted by $U_i(CA)$. The global utility of CA is the sum of the personal utilities of the wards, $U(CA) = \sum_{i=1}^{n} U_i(CA)$.

ORDA as an ADCOP: In order to represent an ORDA as an asymmetric DCOP, we define the possible allocations of RDs to Wards in terms of variables held by agents and domains of values that can be assigned to them. Furthermore, the utility calculation needs to be decomposed into asymmetric constraints that agents (representing wards) can compute and aggregate. Agent A_i representing ward W_i holds variables $v_{i_1}, v_{i_2}, ...v_{i_k}$, where k is the maximal number of resources that it may be allocated. The domains include all the relevant RDs.

The utility that an agent derives from an allocation, is defined by its personal constraints. We denote by C_i the set of constraints of agent A_i. A constraint $c \in C_i$ includes a set of q assignments, $q \geq 1$ and the utility the agent derives from this constraint, i.e., $c = [\langle A_{i_1}, RD_{j_1}\rangle, ..., \langle A_{i_q}, RD_{j_q}\rangle, u_i]$. Personal preferences are represented by unary constraints. *Cardinal constraints* are also unary constraints, which include all the resources allocated to a single agent. The utility that agent A_i derives from an allocation, U_i, is the sum of the utilities it derives from all the constraints it is involved in.

3.2. Operation Day Scheduling

The operation day scheduling (ODS) is a multi-agent optimization problem where each agent has a complex local problem and includes inter agent constraints. The natural structure of this problem has agents representing wards (WRs) that need to schedule the operations in operation rooms that were assigned to them on specific days, on one side, and agents representing coordinators of *constraining elements* (CEs) on the other. The resulting structure is a bipartite graph.

Formally the operation day schedule problem (ODSP) includes two sets of agents: WR, the agents representing wards, and CE, agents representing constraining elements. The problem solved by each $wr \in WR$ is a tuple $\langle S, RTG, R, X_s, X_\sigma, C\rangle$. $S = \{S_1, S_2, ..., S_n\}$ is the set of the ward's surgeons. $RTG = \{\sigma_1, \sigma_2, ..., \sigma_m\}$ is the set of surgery requests that can be scheduled. R is a table defining the availability of operating rooms to the ward, in the relevant dates. For example, $r_{i,j}$ represents the allocation of room j to the ward on day i. X_s and X_σ are two sets of variables. X_s includes variables that represent the assignment of surgeons to operations, e.g. the assignment of S_i to an operation o. The domain of a variable $x \in X_s$ includes all surgeons that are available at the specific day. X_σ consists of variables representing the allocation of an operation request (OR) to an operation o that will take place at a room at that day. The value of $0 < o \leq k$ is the position of this operation in the order of the k operations that are scheduled to be performed in that room that day. If $o = 1$ then the operation is the first to be held in that room that day. The domain of variable $x \in X_\sigma$ includes all the ward's $RTGs$.

C is the set of constraints. It includes hard constraints, e.g., a constraint that prevents the same surgeon from being allocated to two different operations simultaneously, and soft constraints that represent surgeons' preferences, the urgency of operation requests etc. The constraints also define the utility derived from the assignment combination of a surgeon and the assignment of an operation request. For example, if the surgeon cannot perform this type of surgery, the utility derived is $-\infty$ and for valid capabilities the utility is positive.

The CE agents solve a standard COP problem, i.e., a problem that is a tuple $\langle X, D, C\rangle$, where X is the set of variables, D is a set of domains for these variables and C is a set of constraints. However, X has a unique structure, since the variables represent requirements in ordered operation slots for all the operation rooms in the hospital. Thus, X is a n over k over r table, where n is the hospital number of operating rooms, k is the maximal number of operations that can be performed in a room in a single day, and r is the maximal number of units of the relevant element that can be required in an operation. An entry in the table $x_{i,o,r}$ represents an assignment of the r'th element in the o'th operation in room i on that day. The domains include all the available elements on a given day, e.g., nurses or X-ray machines. For this assignment problem, hard constraints prevent invalid assignments while soft constraints define the degree of suitability of the elements to the surgery taking place and the preferences. In addition, constraints also represent priorities between wards and among types of surgeries.[2]

The global utility for a complete assignment to this distributed allocation problem, as in standard (ADCOP), is the sum of utilities of all agents.

4. Local Search Algorithms

In this section we present distributed incomplete local search algorithms for solving the problems we model above. While we use distributed local search to solve the problems, the two models require the design of algorithms that implement different solution approaches. In the first, agents need to balance between the requirements of the wards they represent and the global good of the hospital they are a part of. Thus, partial cooperation algorithms are appropriate [22, 49]. For the generation of daily room schedules, besides each ward's internal constraints, inter ward resource constraints exist, i.e., limited resources required for performing operations. These are represented

[2]A full problem formalization is included in the appendix

by agents that manage their assignment to operations. Thus, these types of problems include two unique features:

1. The local problem that each of the agents must solve is on its own a complex multi variable problem.
2. The constraint graph is bipartite, where on one side, there are wards representing agents and on the other side are the agents representing the constraining elements.

4.1. Partial Cooperative Algorithms for the Operating Rooms and Dates Allocation Problem

In order to solve the ADCOPs representing ORDA problems we adjusted partial cooperative local search algorithms (including socially motivated partial cooperative algorithms) such that they will be compatible with ORDA problems [22, 49]. The main difference between the existing general partial cooperative algorithms and the algorithms adjusted for ORDA, is that the actions in ORDA algorithms are specific requests for the release or exchange of RDs. The expected benefits that agents exchange are either the utility that they are expected to derive from the RDs that are released for their use or the increment in utility as a result of an exchange.

In more details, the AGC_ORDA version of AGC (depicted in Algorithm 3), includes three synchronous phases (iterations) in each step of the algorithm. In the first, agents select one of their neighbors and send a request for a release of a RD or an exchange, including their expected gain from this action. In the second phase, each agent selects the offer with the highest reported gain (including its own), which does not cause a reduction in utility, beyond her limitations, sends an accept message to the proposer and NEG! messages to all its other neighbors. In the third phase, requests that were not replied by NEG! messages are performed whether they are transfers or exchanges of RDs. Notice that in contrast to the standard version of AGC, here only the agents involved in a request (the agent sending the request and the one receiving it) must approve it in order for it to take place.

A similar adjustment is required in order to use the SM_AGC algorithm in ORDA scenarios (pseudo code depicted in Algorithm 4). In the first phase, agents exchange preferences regarding the RDs they would like to receive from their neighbors. In the second phase, each agent calculates the social gain for each request from a neighbor for an exchange or release of a RD it holds, and selects the one with the highest social gain (the mutual gain for her and the other agent involved). Notice that here, only the agents involved in the exchange of a resource affect the gain, thus, for each request it receives, an agent only needs to take into consideration the preferences of the sender of the request, and its own. After comparing the expected social gains of all possible requests it can send, it selects the request with the highest social gain and sends it to the relevant neighbor along with the social gain. The following actions in the third and fourth phases of the algorithm, are similar to the second and third phase of the AGC_ORDA algorithm described above.

Consider the example depicted in Figure 1. It is an example of the hospital operating room scheduling problem we described above. The example includes three wards and two operating rooms that are allocated per day. Ward$_1$ can only use operating room OR$_1$, Ward$_3$ can only use OR$_2$ and Ward$_2$ can use both. For each ward the minimal and maximal number of allocations they require is depicted on its left (lower and upper bound). We will assume that a ward that does not satisfy its lower bound endures a cost of 100. The allocation that the agents seek to schedule is for a five day working week. For each room that can be allocated to a ward, next to the line connecting them, the personal preferences of the ward are specified as an array of natural numbers between

Algorithm 3 AGC_ORDA

input: $baseLineAllocation_i$, $baseLineCost_i$, λ_i,

```
$alloc \leftarrow baseLineAllocation_i$;
$\mu_i \leftarrow baseLineCost_i$;
$localView \leftarrow null$;
send($alloc$) to $N(i)$;
while stop condition not met do
 PHASE 1:
  Collect all $alloc$ messages and update $localView$
  $\langle r_{iq}, gain_i \rangle \leftarrow improvingRequest()$;
   send($\langle r_{iq}, gain_i \rangle$) to $A_q$;
 PHASE 2:
  Collect all $\langle r_{ji}, gain_j \rangle$ messages;
  $a_j \leftarrow$ agent in $N(i) \cup A_i$ with maximal socialGain s.t.
        $c_i(localView$ after performing $r_{ji}) \leq \mu_i \cdot (1 + \lambda_i)$;
  send($Neg$!) to $N(i) \setminus a_j$;
 PHASE 3:
  Collect $Neg$! messages;
  if did not receive $Neg$! from $A_q$ and from $A_i$ & can improve
    then perform $r_{iq}$;
  else if did not receive $Neg$! from $A_j$
    then perform $r_{iq}$;
  send($alloc$) to $N(i)$;
```

zero and nine. The preferences for day 1 are presented in the left entry of each array, next, the preferences for day 2 and so forth.

Consider a situation in which the current allocation has:

- OR$_1$ allocated to Ward$_1$ in the first two days of the week.
- OR$_1$ is allocated to Ward$_2$ in the rest of the week (days 3, 4 and 5).
- OR$_2$ is allocated to Ward$_2$ in the first three days of the week.
- OR$_2$ is allocated to Ward$_3$ in the last two days of the week.

If the agents are performing AGC, then in phase 1, Ward$_1$ sends to Ward$_2$ a request to transfer OR$_1$ to them on day 4 with a gain of 108 (since currently this ward is not satisfying it's lower bound). The preferences for this day are high for Ward$_2$. However, if λ is large enough, they would agree to release OR$_1$ on that day to Ward$_1$, since they will remain above the lower bound. If the agents would perform SM_AGC, the results would be the same because the preferences of Ward$_3$ are not relevant for OR$_1$ and the gain for Ward$_1$ is much larger than the loss for Ward$_2$. However, in the next iteration, if the agents are performing AGC, Ward$_1$ can ask for OR$_1$ at day 3 as well and it will get it for similar reasons. On the other hand, if agents are performing SM_AGC, Ward$_1$ would not make this request because the social gain is negative.

Algorithm 4 SM_AGC_ORDA

input: $baseLineAlloc_i$, $baseLineCost_i$, λ_i and Ω_i

```
$alloc \leftarrow baseLineAlloc_i$;
$\mu_i \leftarrow baseLineCost_i$;
$localView \leftarrow null$;
send($alloc$) to $N(i)$;
while stop condition not met do
  PHASE 1:
    Collect all $alloc$ messages and update $localView$
    for each $A_j \in N(i)$ do
      $\pi_{i,j} \leftarrow preferences(A_j)$;
      send($\pi_{i,j}$) to $A_j$;
  PHASE 2:
    Collect all $\pi$ messages;
    $\Pi_i \leftarrow \pi_{j \in N(i)} \cup preferences(A_i)$;
    $r_{iq} \leftarrow socialImprovingRequest(\Pi_i, \Omega_i)$;
    send($r_{iq}, socialGain_i$) to $A_q$;
  PHASE 3:
    Collect all $\langle r_{ji}, socialGain_j \rangle$ messages;
    $a_j \leftarrow$ agent in $N(i) \cup A_i$ with maximal socialGain s.t.
          $c_i$($localView$ after performing $r_{ji}$) $\leq \mu_i \cdot (1 + \lambda_i)$;
    send($Neg$!) to $N(i) \setminus a_j$;
  PHASE 4:
    Collect $Neg$! messages;
    if did not receive $Neg$! from $A_q$ or from $A_i$ & can improve
      then perform $r_{iq}$;
    else if did not receive $Neg$! from $A_j$
      then perform $r_{ji}$;
    send($alloc$) to $N(i)$;
```

At the same time, Ward$_3$ asks Ward$_2$ to exchange the allocations of OR$_2$ of days 1 and 5. Ward$_2$ has already agreed to release OR$_1$ on day 4 and therefore it sends Neg! to Ward 3. In the next step of the algorithm it will agree to exchange the days of OR$_2$ with Ward$_3$. The resulting allocation will be OR$_1$ to Ward$_1$ on days 1, 2 and 4. OR$_2$ to Ward$_3$ on days 1 and 4 and the rest of the allocations are to Ward$_2$. The utility for each ward is $U_1 = 15$, $U_2 = 25$, $U_3 = 13$ and the global utility is 53.

4.2. Distributed Local Search for Generating Daily Schedules

The model we describe above for representing the daily schedule problem includes a bipartite graph of agents, each of which has its own complex local search problem. Distributed local search algorithms are synchronous algorithms in which agents exchange information and decide whether to replace their local assignments [29, 50]. Thus, to design a distributed local search algorithm, we first specify how agents generate local assignments. In all our algorithm implementations, the agents used simulated annealing (SA) [38, 42] for generating the first solution to their local problem. In some versions, SA was used at each iteration of the distributed search.

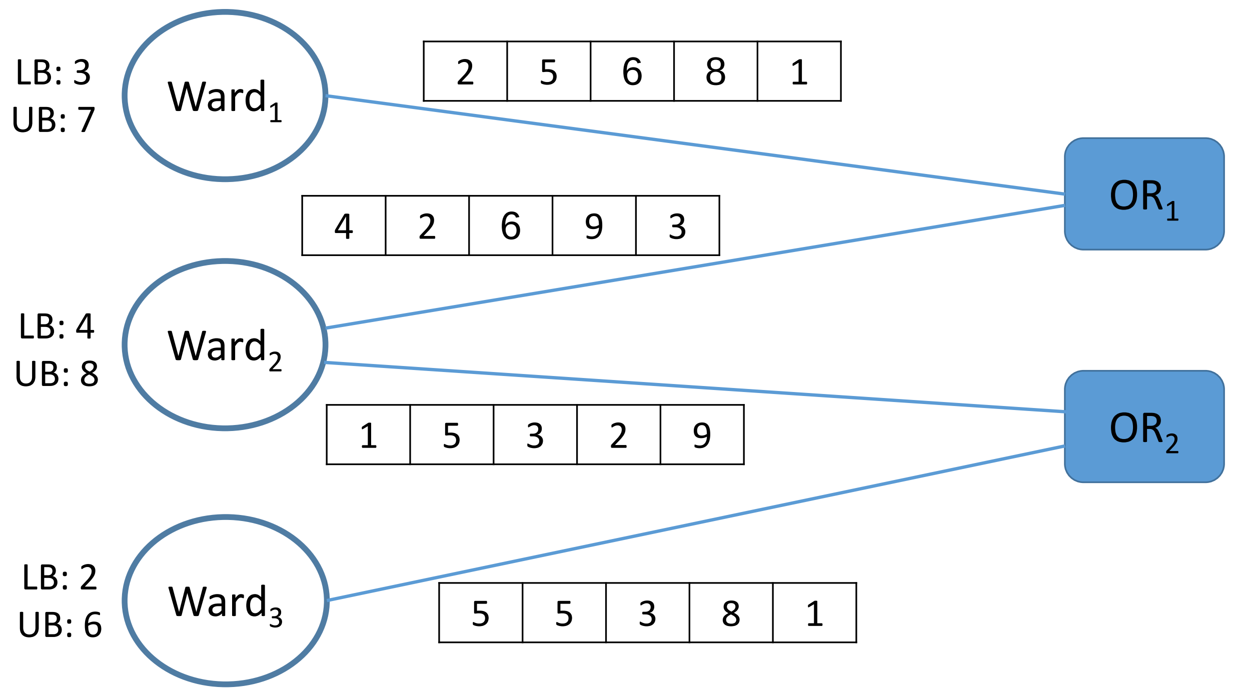


Figure 1: Hospital operating rooms example

.

We examine two main approaches for the design of the distributed local search algorithms. In the first, inspired by the distributed stochastic algorithm (DSA) [18, 50], an agent, generates a solution to its local problem and sends it to its neighbors in the bipartite graph. Then, in each iteration, the agent searches for an improving assignment and, if it finds one, replaces its current assignment with probability p (in our experiments we set $p = 0.7$). Notice that, in contrast to standard DSA [50] the graph's structure is bipartite, thus, agents send their (complex) assignment only to agents of the other type, i.e., WRs to CEs and vice versa.

The second approach we propose, considers the natural role of CEs, which is to provide service to the operating wards. Thus, we propose a query-response protocol in which the wards suggest schedules and the CEs react to these suggestions, specifying which of the scheduled operations they could allocate the required element.

A sketch of the code of the proposed query response daily schedule algorithm (QRDSA) is presented in Algorithm 5. The main difference in QRDSA from standard DSA, is the query response structure. Thus, the pseudo-code for the WR agents starts with selecting an assignment for their local problem using SA. Then, a WR agent w_i, sends its selected schedule to its CE neighbors (in set CE_i) and waits for their response. Once these responses are received, it updates its local information, and revises its local assignment before sending it again. On the other hand, the CE agent ce_i waits for the assignments of its WR neighbors (WR_i) to arrive before it performs its computation. It updates its local operation schedule and proposes its corresponding assignment of constrained elements to this schedule. Each of its neighboring WR agents sends the projection of its assignment on the schedule relevant to the neighbor.

For the selection of the revised assignment in each iteration in both algorithms, we used three different methods:

1. Single change (SC): The variables are ordered. According to this order, the agent searches for the first variable (operation) that did not receive all elements required for the operation to take place (not fully scheduled). The assignment for this variable is replaced. If the agent's utility decreases by this change, the variable's previous value is returned, and the agent attempts to change the following ordered variable's value.
2. Single change with exploration (SC_e): A random variable is selected. The agent tries to replace the current variable's assignment with an alternative value to improve its utility. Suppose the variable change does not improve the utility. In that case, the variable's previous value is reassigned, and the another random variable is chosen, until a stop condition is met.
3. Simulated Annealing (SA): The agent performs a new SA search to select its assignment in every iteration.

We also examined the use of a stability factor (sf) that penalizes a change in the assignment of an operation, i.e., whenever an operation that was fully scheduled was moved to a different slot or postponed to an undetermined future schedule, there was a reduction in the utility the agent derives. We examine the stability factor in two levels; the first that does not record unsuccessful attempts (sf). The second on the other hand uses a dynamic memory structure, which stores no-good visited solutions (sf_ng) throughout the algorithms run. The no-good visited solutions consist of the surgery requests which were scheduled but did not result in a full allocation of all the constraining resources—in the expanded formulation. Scheduling a surgery request from the no-good structure results in a reduction in the agent's utility. Furthermore, the size of the penalty is relative to the time passed since the unsuccessful attempt to schedule the operation request.

In all versions of the algorithm we used forward checking, i.e., values that were not consistent with previous assignments performed were removed from the domains. For the selection of value assignments in the single change versions, among the consistent values in the domains we used two methods. The first was selecting a random value and the second was selecting the value that seemed most promising (the one expected to increment the utility the most). We demonstrate in our experiments that the second method required much more calculations that was not always beneficial.

Figure 2 presents a small example of the daily schedule problem described above. It includes two wards, and it demonstrates the scheduling of a single day in which each ward is allocated a

Algorithm 5 QRDSA

```
WR:
 1: while Not Terminated do
 2:     sched ← assign(localProblem)
 3:     send(sched) to $CE_i$
 4:     receive $resp_j$ from all $ce_j \in CE_i$ and update(localInfo)
CE:
 5: while Not Terminated do
 6:     recieve $sched_j$ from $WR_i$ and update (localInfo)
 7:     sched ← assign(localProblem)
 8:     for all $wr_j \in WR_i$ do
 9:         $resp_j \leftarrow \{sched \downarrow resp_j\}$
10:         send $resp_j$ to $wr_j$
```

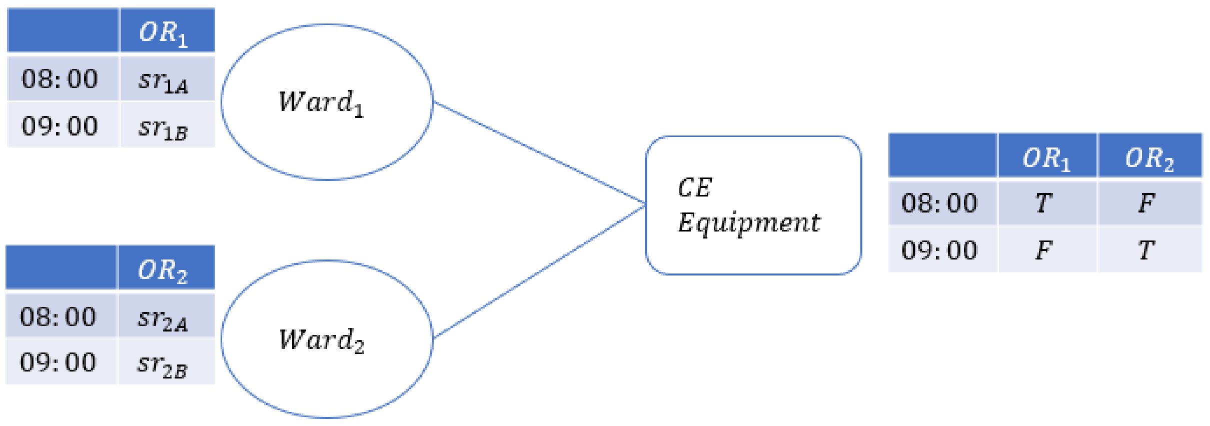


Figure 2: Daily schedule generation example.

single operating room. The problem further consists of a single constraining element (CE) agent that schedules the use of an indivisible resource (e.g., an X-ray machine). The day in this small example is two hours long, and the length of each operation is one hour. We further assume that surgery requests sr_{1A} and sr_{2A} require the equipment unit. Consider the initial local schedules chosen by each one of the agents as presented in the tables included in Figure 2. As depicted in the CE agent table, in its initial schedule, the agent allocates the equipment unit such that each ward has it for one surgery slot. Both surgery requests overlap in the current schedule; revisions must be made for the daily schedule to be feasible.

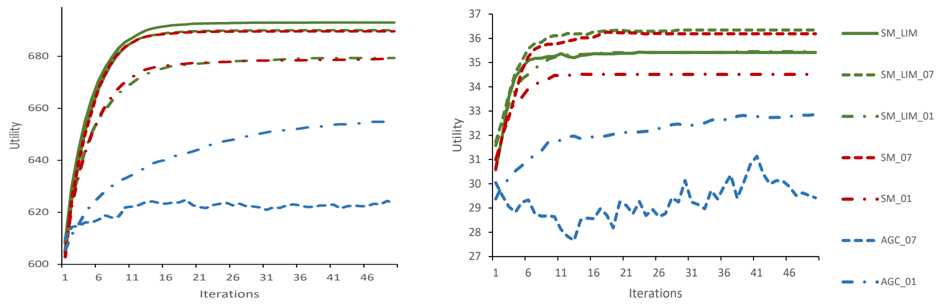
5. Experimental Evaluation

Our experiments included scenarios based on real data for both problem types. In order to avoid privacy breach, some of the parameters of the problems were selected randomly. However, the distributions from which they were selected were realistic according to our analytical review of the data and to the hospital personal input. In all sets of experiments we used t-tests for examining statistical significance.

5.1. Evaluation of RD Allocation Algorithms

This set of experiments included different versions of socially motivated local search algorithms, solving the hospital operating room date allocation problem. Agents represented hospital wards with different needs. The resources being allocated were operating rooms, each with specific properties, that make it attractive to some of the wards, and useless for others. The problem included 10 wards and 15 operating rooms allocated periodically every day. The allocation was for a five working day week , i.e., each room was allocated five times. For each problem, the personal constraints specifying the preferences of wards over days of the week and operating rooms was randomly picked between zero and 9. Among the 15 rooms, 7 could be used by all the wards, one could be used by a single ward, two could be used by two wards, three could be used by three wards and the last two could be used by four wards.

In order to examine the relation between the results and the problem structure, we generated two additional, less realistic, sets of problems: A set of sparse problems, in which we had every



(a) (b)

Figure 3: Average social welfare (a) for the origin problem set (b) for the sparse problem set.

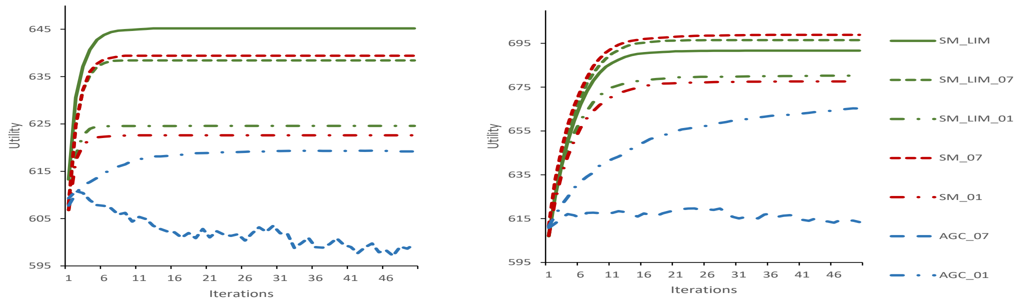
ward interested in exactly two operation rooms, and a dense set, in which we had each ward interested in exactly five operating rooms. We will refer to the three sets as *origin*, *sparse* and *dense*, respectively.

The versions of the partial cooperative local search algorithms we compared included (corresponding notations in brackets):

- AGC with $\lambda = 0.1$ ($AGC\_0.1$).
- AGC with $\lambda = 0.7$ ($AGC\_0.7$).
- SM_AGC with $\lambda = 0.1$ ($SM\_0.1$).
- SM_AGC with $\lambda = 0.7$ ($SM\_0.7$).
- SM_AGC with bounds (agents reject any request that may cause a reduction beneath their lower bound and do not require a trade for allocations they give beyond their upper bound) (SM_LIM).
- SM_AGC with bounds and $\lambda = 0.1$ ($SM\_LIM\_0.1$).
- SM_AGC with bounds and $\lambda = 0.7$ ($SM\_LIM\_0.7$).

Figure 3(a) presents the global utility (social welfare) derived from the allocations generated by the versions of the algorithms listed above, as a function of the number of iterations preformed. Consistent with the results presented in [49], the results depicted demonstrate the clear advantage of the socially motivated versions over standard AGC. Moreover, they demonstrate that intentions for cooperation (represented by λ) must be combined with preference sharing among agents, in order to increase social welfare. Thus, in all socially motivated versions the $\lambda = 0.7$ versions outperform the $\lambda = 0.1$ versions. On the other hand, the $\lambda = 0.1$ version is more successful in AGC. Among the socially motivated versions of the algorithm, the ones using bounds are more successful.

One may wonder if the use of partial cooperative methods prevents outcomes in which some of the agents derive very low utility from the allocation. In order to answer this question we present in Figure 3(b) the average on the minimum utility derived by an agent from the allocations produced by the different algorithms. It is clear that socially motivated algorithms with $\lambda = 0.7$ are most successful when considering this egalitarian measure.



(a) (b)

Figure 4: Average social welfare (a) for the sparse problem set (b) for the dense problem set.

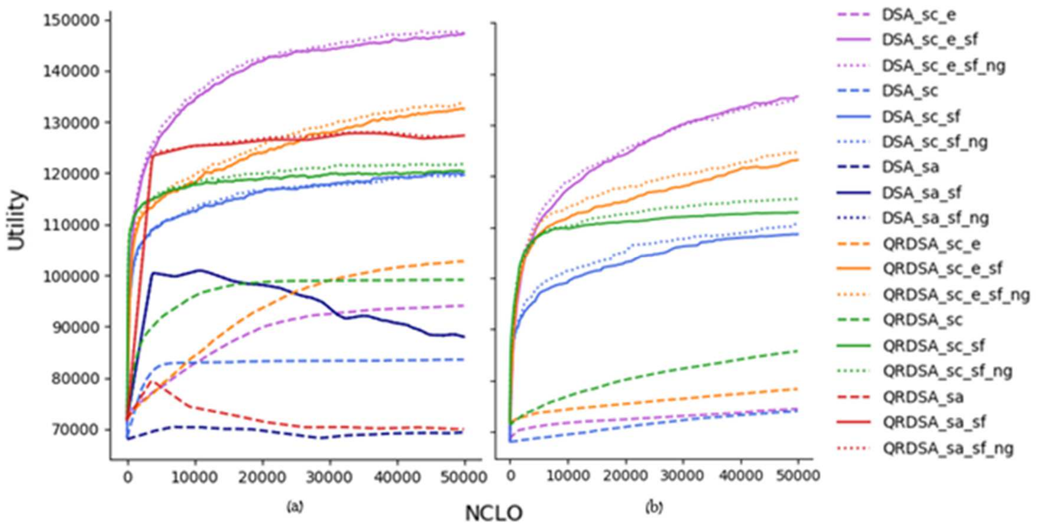


Figure 5: Average global utility (a) for the origin set with best-selection (b) for the origin set

Figure 4 presents the social welfare of the allocations generated by the algorithms for the sparse (a) and the dense (b) synthetic problem sets, respectively. The most apparent difference is that for the sparse problems, the version that only uses bounds is most successful, while for the dense problems the versions that use $\lambda = 0.7$ produce better solutions. It seems that when agents do not have many options for operating rooms to be allocated to them, only the bounds are relevant, while when more options are available, more refined intentions for cooperation are beneficial.

5.2. *Evaluation of Daily Schedule Algorithms*

This set of experiments included different versions of local search algorithms, solving the operations daily schedule problem, implemented in a simulator in which the realistic distributed scheduling problem is represented, according to the model described. In all our experiments the instance of the bipartite graph described above covered 10 WR agents representing 10 different surgical wards, and 3 CE agents representing the nurses, anesthetists and equipment allocation surgical coordinators.

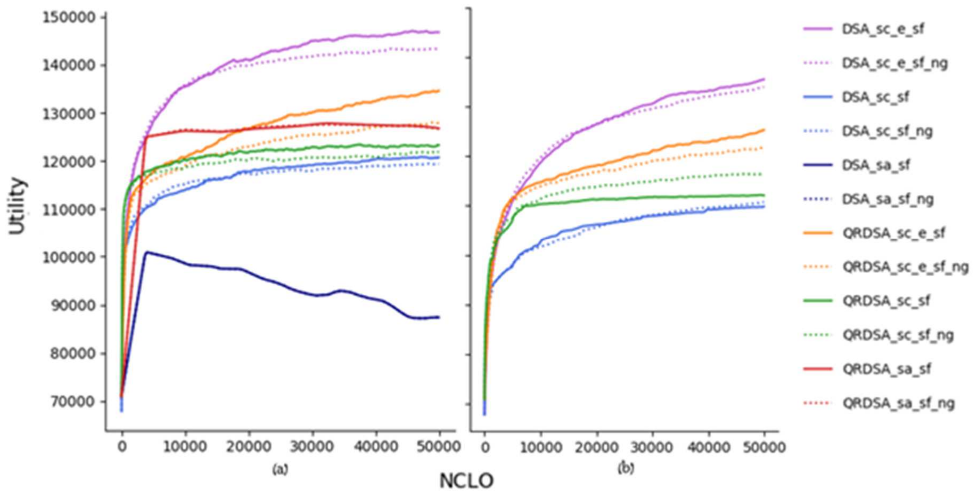


Figure 6: Average global utility for the origin set with enlarged ng (a) with best-selection and enlarged ng (b)

All problems included 500 patients awaiting surgery. Each patient had a birth date sampled uniformly between 01/01/1925 and 01/01/2020. Every patient holds a list of at least a single surgery request. Different parameters define each surgery request:

- The surgery type of the surgery request was uniformly selected from all the ward's possible surgery types.
- The number of cancellations (NC) is the number of times that this surgery request was scheduled for surgery and canceled. For every surgery request, the number of cancellations was sampled uniformly between 0 and 10: $NC \sim Uniform(0, 10)$.
- The entrance/referral date is when the surgery request entered the hospital's system and the queue of surgery requests awaiting surgery. The date was sampled uniformly from a period of a year before the scheduling day.
- A surgery request may or may not be assigned in advance with a specific surgeon. Usually, the surgery requests set in advance with a particular surgeon are unique and complex cases. To simulate this kind of behavior for every ward, a random surgery type was chosen. Then, all surgery requests of this surgery type were assigned with a specific surgeon randomly selected from all the surgeons qualified for this surgery type and highly graded.

50 surgical units were randomly (uniformly) assigned to the ten wards (assuring at least one unit to each ward), and 300 surgery types randomly assigned to the wards. For every surgery type, the following parameters were selected: urgency, complexity, duration, and utility derived by the hospital. The hospital in our hometown refers to six levels of urgency and complexity of surgeries, such that level 1 is the lowest and level 6 is the highest. For every surgery type, the urgency and

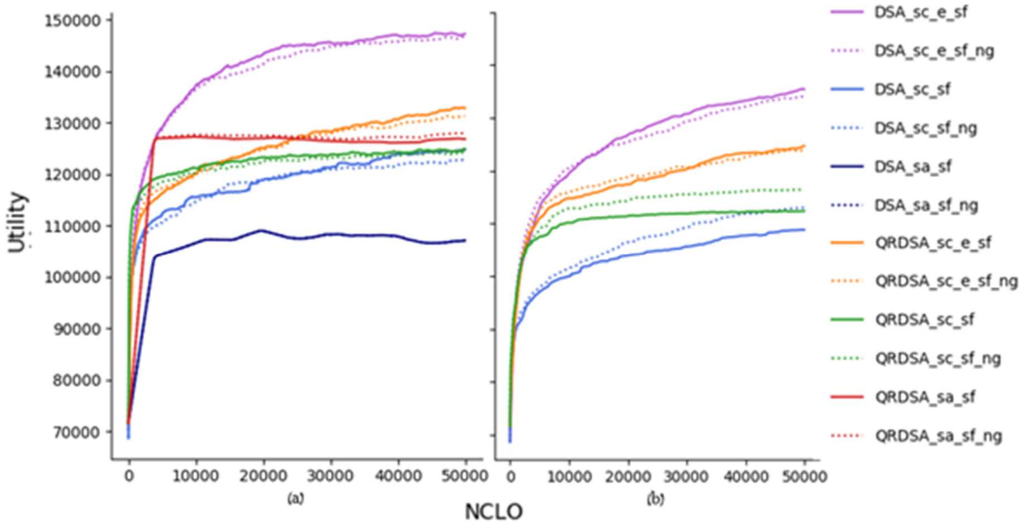


Figure 7: Average global utility for the origin set with enlarged sf and ng(a) with best-selection and enlarged sf and ng(b)

complexity were chosen from the following discrete distribution:

$$Pr(X = x) = \begin{cases} \frac{1}{12}, & \text{if } x = 1, 6 \\ \frac{1}{6}, & \text{if } x = 2, 5 \\ \frac{1}{4}, & \text{if } x = 3, 4 \\ 0, & \text{else} \end{cases}$$

The duration of every surgery type was uniformly sampled from a minimum duration of 30 minutes and a maximum duration of the surgical day length (L): $Duration \sim Uniform(30, L)$.

The number of surgeons for each ward was decided randomly and uniformly between the number of operation rooms allocated to the ward on a day and the number of surgery types of the ward multiplied by 3. Every surgeon had a set of graded skills that inform the different surgeries she is qualified to do and the level of expertise. The set size differed between the surgeons and was decided randomly between 1 and the number of the ward's surgery types. The level of expertise of every surgery type was also randomly selected, assuring that there would be at least a single expert surgeon for every surgery type. The surgeons perform surgeries in different shifts during the day; as learned from the hospital, the generator implies a single expert surgeon for every surgery type.

Each problem included 15 operation rooms, and each of them had a list of compatible surgery types. For every room, we sampled a random number of surgery types between 1 and the number of surgery types of the hospital. Each ward had the rooms that were allocated to it for specific dates. For the experiment at hand for each room, a random ward was chosen uniformly from all the wards compatible to perform surgery in it.

In addition, 100 nurses and anesthetists were available for every problem. The nurses differ with different skills that define the type of surgeries they can be allocated to. For every surgery type in the problem, a set of nurses was selected as qualified to perform this surgery type. The number of

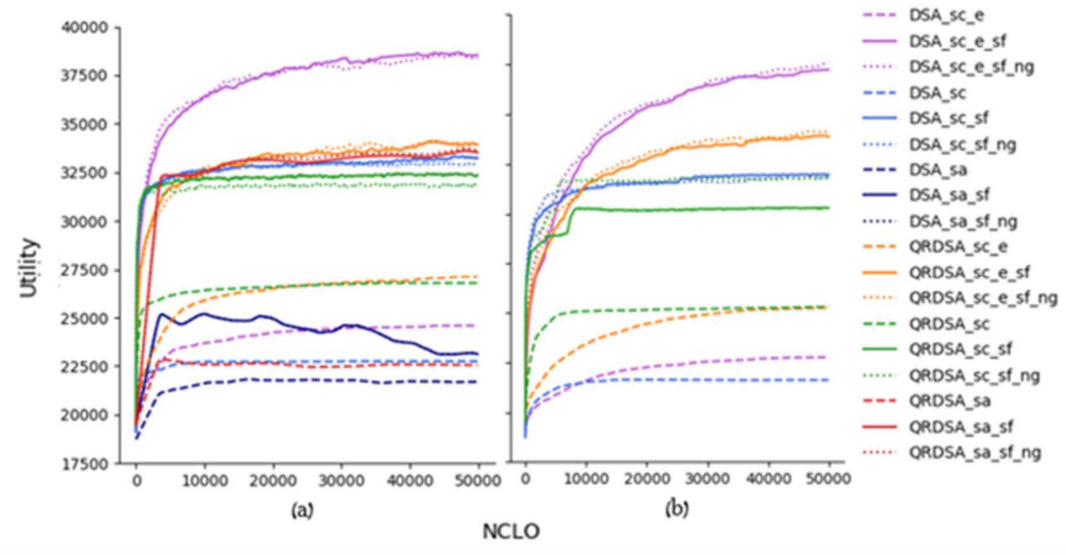


Figure 8: Average global utility for problems with 5 rooms(a) with 5 rooms and best-selection

nurses was drawn uniformly between 1 and the number of hospital nurses. In the hospital, nurses are first qualified to be scrubbing nurses and only later circulating nurses. Every nurse holds along with the set of surgery types she is eligible to do a sub-set of these surgery types to indicate the surgery types she is trained to operate as a circulating nurse. Nurses also perform surgeries by shifts. The number of nurses assigned in a surgical shift will be precisely the number of nurses needed, i.e., the number of surgical rooms multiplied by 2 (for every surgery, there is a need for a circulating nurse and a scrubbing nurse). The nurses for each shift were randomly and uniformly chosen from all the nurses in the problem.

The anesthetists were divided by their experience into three ranks – Intern, Expert, or Senior. Each rank had a set of roles that the anesthetics included in it could perform. For every problem instance, the data generator assures at least a single Senior anesthetist, and for every ward, the generator confirms at least a single Intern and a single Expert. The rest of the anesthetist's ranks are sampled from the subsequent discrete distribution:

$$Pr(X = x) = \begin{cases} 0.45, & \text{if } x = \text{Intern} \\ 0.4, & \text{if } x = \text{Expert} \\ 0.15, & \text{if } x = \text{Senior} \\ 0, & \text{else} \end{cases}$$

Intern anesthetists' current ward rotation is selected uniformly and randomly between all the surgical wards. An Intern is certified to perform surgery in all her past rotation's wards, but not all surgery types. For every Intern, a different number of wards are sampled to simulate her last rotations. The number is determined uniformly between 0 and the number of the hospital's surgical wards. For every surgery type, the number of Interns certified to perform surgery is sampled uniformly between 1 and the number of Interns who are or were in rotation of its ward. Anesthetists also perform surgeries by shifts. Every shift is staffed by precisely the number of anesthetists

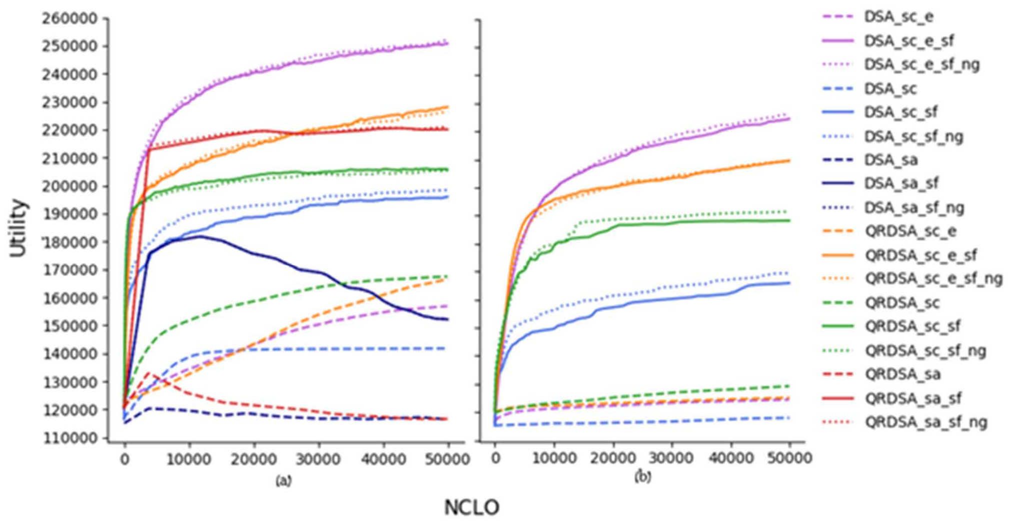


Figure 9: Average global utility for problems with 25 rooms(a) with 25 rooms and best-selection

needed following the different ranks required by the various roles during the surgical day.

Problems included three types of equipment that were required for some types of surgeries. First, the units available for every type of equipment were selected randomly and uniformly between 1 and 15 (the number of operating rooms). Then, the surgery requests that required each one of them was randomly selected.

To examine the dependency of the results on the structure of the problem, we also examined the problem in four less realistic scenarios. We analyzed the effect of two of the problem parameters on the quality of the algorithms, the number of operating rooms in the hospital and the length of the operating day. The number of operating rooms was enlarged to twenty-five in one set of experiments to illustrate a sparser problem. In the second set of experiments, the operating day was prolonged to a ten-hour shift instead of a seven-hour shift. Finally, to demonstrate a dense collection of problems, two additional sets of experiments were conducted. In the first, we decreased the number of operating rooms to five, and in the next, the operating day was shortened to a four-hour shift. In addition, we performed parameter analysis to determine the adequate stability factor. Finally, all the algorithms were assessed without using any stability factor and with a stability factor. Also, an additional set of experiments was performed, enlarging the value of the stability factor by fifty.

5.3. *Experimental Results*

Our results compare algorithms implementing the two approaches, DSA inspires the first, and the second is the QRDSA algorithm described above. For each of the two approaches, we implemented the identical versions of the algorithm, i.e.:

1. sc - single change.
2. sc_e - sinle change with exploration.
3. sa - simulated annealing in each iteration.
4. An addition of sf - an addition of stabilization factor.

5. An addition of ng - an addition of a no-good dynamic memory structure to the stabilization factor.
6. An addition of $best - selection$ - an addition of values selection from domain.

We used non-concurrent logical operations (NCLO) as a time measure [33, 52]. Each algorithm solved fifty random instances. The results present the average over the utility of the solutions produced by the different algorithms. The experiments were performed in Python with Pycharm IDE on a Windows Operating System.

Figure 5 (a) presents the results in the origin set representing the natural setting, i.e., seven-hour shifts and 15 operating rooms. Apparently, the versions using a stability factor have a significant advantage over the algorithms that did not. The second observation is the dominance of the versions of the algorithms using single change with exploration, both in DSA and in QRDSA. All single change versions of DSA did better than the simulated annealing versions. Moreover, the quality of solutions produced by the DSA version that used simulated annealing deteriorated instead of improving during the execution. When considering QRDSA, the simulated annealing version outperformed the single change version. When using single change with exploration, the DSA versions significantly outperformed the QRDSA versions. When using simulated annealing, QRDSA dominated. It is notable that the use of the no-good dynamic memory structure did not contribute significantly.

Figure 5(b) presents the results of the algorithms, solving the origin set, while using the best value selection method (best-selection). Obviously, (Compared to the results in Figure 5(a)) this method is less successful than the random value selection. This result can be explained by the limited view of the agents on the constraints of the distributed problem, e.g., a WR can repeatedly select surgery requests of a particular surgery type, which from its point of view will contribute in the most significant way to increasing its ward's utility, but on this specific day, there are no qualified nurses for this surgery type. It is also worth noting that both single change with exploration versions of both algorithms are still in a climbing trend at the end of the 50,000 NCLO. The moderate rate of increase can be explained by the large number of values assessments for each variable's value change.

In order to assess the maximal influence of the stability factor (sf) on the algorithms, Figure **??** presents the results of the versions of the algorithms that include the sf penalty, after increasing it fifty times compared to the sf used when producing the results presented in Figure 5. Figure 6 illustrates the results of the sf versions of the algorithm with the penalty, resulting from a selected value from the ng structure, fifty times larger. Finally, Figure **??** presents the results of the sf versions of the algorithm with both sf and ng penalties fifty times larger than in the results shown in Figure 5.

The results presented in Figure **??** and Figure 7 indicate that this change mainly affected the simulated annealing versions, especially the DSA version, which did not deteriorate that much when a larger sf was used. Figure 6 indicates that the enlargement of ng needs to be accompanied by the growth of sf to make an impact. Furthermore, the changes when enlarging both sf and ng are not significant to when only broadening sf.

Figure 8 and Figure 9 present results of the algorithms solving problems in which the number of operating rooms available in a day was either smaller (5 in Figure 8) or larger (25 in Figure 9). When the number of rooms was smaller, the problem became tighter. As a result, the advantage of the single change explore versions became more prominent, while the other versions using sf (apart from the simulated annealing DSA version) produced similar results. On the other hand, when solving the less tight version with 25 rooms, the deterioration of the DSA simulated annealing versions was steeper. Also, it is interesting to note that in the tighter scenario, the algorithm versions implemented with best-selection produce similar results to those that don't. It seems that when the problem's constraint are tighter, the effort in selecting the best value is more beneficial, and a random selection has lower probability to be successful. with a relatively more minor gap between them when compared to the rest of the setups.

Figure **??** and Figure **??** present results of the algorithms solving problems in which the operating day length was either shorter (240 minutes in Figure **??**) or longer (600 minutes in Figure **??**). When the day is shorter, the problem becomes tighter, again the problem became tighter. Nevertheless, in this case the advantage of the single change explore version did not become more prominent. However, the results were sparser once the surgical day was prolonged, and the single change explore dominated significantly. In this scenario, the deterioration of the DSA simulated annealing versions was steeper in the tighter setting. Uniquely the addition of the ng memory structure in the 240-minute duration day implemented with best-selection,i.e., with value selection from the domain, impacted and improved the single change versions in both approaches significantly.

The results on all scenarios emphasize the importance of stability in such a distributed environment where agents aim to resolve conflicts among them. If agents make too many changes to their local assignment, their neighbors' decisions are generated while considering obsolete information. Our results indicate that in the simulated annealing versions, an agent made an average of 17.67 operation assignment changes in each iteration in origin set up. In contrast, the single change versions made at most one. Furthermore, when the algorithm explores the different possibilities for this single change, this effort is worthwhile.

6. Conclusions

The operation room allocation and scheduling application we present in this paper is a realistic application that requires distributed models and algorithms, in order to preserve the natural structure of the problem and the autonomy and privacy of the involved humans. It includes agents representing wards (or ward directors), each with her own interest, which belong to a large organization (a hospital), and thus, besides their personal objectives there is a global mutual goal.

Each of the two phases of the problem includes unique properties, which trigger the design of non trivial models for representing them. In the room per date allocation problem, partial cooperative agents divide a mutual resource among themselves and maximize a global goal while satisfying their needs. In the daily schedule phase, wards optimize their complex operation schedule, while interacting with agents that represent the hospitals constraining elements.

In future work we intend to explore similar realistic distributed applications and to investigate the compatibility of incomplete inference algorithms for solving these realistic problems.

References

[1] C. A. H. F. M. Association. Achieving operating room efficiency through process integration. *Healthcare financial management: journal of the Healthcare Financial Management Association*, 57(3):1–112, 2003.

[2] S. Batun, B. T. Denton, T. R. Huschka, and A. J. Schaefer. Operating room pooling and parallel surgery processing under uncertainty. *INFORMS journal on Computing*, 23(2):220–237, 2011.

[3] J. T. Blake and J. Donald. Mount sinai hospital uses integer programming to allocate operating room time. *Interfaces*, 32(2):63–73, 2002.

[4] A. H. Bond and L. Gasser. *Readings in distributed artificial intelligence*. Morgan Kaufmann, 2014.

[5] I. Brito, A. Meisels, P. Meseguer, and R. Zivan. Distributed constraint satisfaction with partially known constraints. *Constraints*, 14(2):199–234, 2009.

[6] B. Cardoen, E. Demeulemeester, and J. Beliën. Operating room planning and scheduling: A literature review. *European journal of operational research*, 201(3):921–932, 2010.

[7] D. Chen, Y. Deng, Z. Chen, Z. He, and W. Zhang. A hybrid tree-based algorithm to solve asymmetric distributed constraint optimization problems. *Autonomous Agents and Multi Agent Systems*, 34(2):50, 2020.

[8] Y. Deng, R. Yu, X. Wang, and B. An. Neural regret-matching for distributed constraint optimization problems. In Z. Zhou, editor, *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI 2021, Virtual Event / Montreal, Canada, 19-27 August 2021*, pages 146–153. ijcai.org, 2021.

[9] B. Denton, J. Viapiano, and A. Vogl. Optimization of surgery sequencing and scheduling decisions under uncertainty. *Health care management science*, 10(1):13–24, 2007.

[10] B. T. Denton, A. J. Miller, H. J. Balasubramanian, and T. R. Huschka. Optimal allocation of surgery blocks to operating rooms under uncertainty. *Operations research*, 58(4-part-1):802–816, 2010.

[11] S. A. Erdogan, B. T. Denton, J. Cochran, L. Cox, P. Keskinocak, J. Kharoufeh, and J. Smith. Surgery planning and scheduling. In *Wiley encyclopedia of operations research and management science*. Wiley Hoboken, NJ, 2011.

[12] A. Farinelli, A. Rogers, and N. R. Jennings. Agent-based decentralised coordination for sensor networks using the max-sum algorithm. *Autonomous Agents and Multi Agent Systems*, 28(3):337–380, 2014.

[13] A. Farinelli, A. Rogers, A. Petcu, and N. R. Jennings. Decentralized coordination of low-power embedded devices using the max-sum algorithm. In *AAMAS*, pages 639–646, 2008.

[14] H. Fei, C. Chu, N. Meskens, and A. Artiba. Solving surgical cases assignment problem by a branch-and-price approach. *International journal of production economics*, 112(1):96–108, 2008.

[15] H. Fei, N. Meskens, and C. Chu. An operating theatre planning and scheduling problem in the case of a" block scheduling" strategy. In *2006 International Conference on Service Systems and Service Management*, volume 1, pages 422–428. IEEE, 2006.

[16] J. Ferber and G. Weiss. *Multi-agent systems: an introduction to distributed artificial intelligence*, volume 1. Addison-Wesley Reading, 1999.

[17] F. Fioretto, W. Yeoh, and E. Pontelli. A multiagent system approach to scheduling devices in smart homes. In *Workshops at the Thirty-First AAAI Conference on Artificial Intelligence*, 2017.

[18] S. Fitzpatrick and L. G. L. T. Meertens. An experimental assessment of a stochastic, anytime, decentralized, soft colourer for sparse graphs. In K. Steinhöfel, editor, *Stochastic Algorithms: Foundations and Applications, International Symposium, SAGA 2001 Berlin, Germany, December 13-14, 2001, Proceedings*, volume 2264 of *Lecture Notes in Computer Science*, pages 49–64. Springer, 2001.

[19] R. Greenstadt, J. P. Pearce, E. Bowring, and M. Tambe. Experimental analysis of privacy loss in dcop algorithms. In *Proceedings of the fifth international joint conference on Autonomous agents and multiagent systems*, pages 1424–1426, 2006.

[20] T. Grinshpoun, A. Grubshtein, R. Zivan, A. Netzer, and A. Meisels. Asymmetric distributed constraint optimization problems. *J. Artif. Intell. Res. (JAIR)*, 47:613–647, 2013.

[21] A. Grubshtein, R. Zivan, T. Grinshpon, and A. Meisels. Local search for distributed asymmetric optimization. In *AAMAS 2010*, pages 1015–1022, May 2010.

[22] A. Grubshtein, R. Zivan, and A. Meisels. Partial cooperation in multi-agent local search. In *ECAI 2012 - 20th European Conference on Artificial Intelligence. Including Prestigious Applications of Artificial Intelligence (PAIS-2012) System Demonstrations Track, Montpellier, France, August 27-31 , 2012*, pages 378–383, 2012.

[23] K. Hirayama and M. Yokoo. The distributed breakout algorithms. *Artificial Intelligence*, 161:1-2:89–116, January 2005.

[24] A. Jebali, A. B. H. Alouane, and P. Ladet. Operating rooms scheduling. *International Journal of Production Economics*, 99(1-2):52–62, 2006.

[25] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *science*, 220(4598):671–680, 1983.

[26] L. R. Kroer, K. Foverskov, C. Vilhelmsen, A. S. Hansen, and J. Larsen. Planning and scheduling operating rooms for elective and emergency surgeries with uncertain duration. *Operations research for health care*, 19:107–119, 2018.

[27] P. C. Kuo, R. A. Schroeder, S. Mahaffey, and R. R. Bollinger. Optimization of operating room allocation using linear programming techniques. *Journal of the American College of Surgeons*, 197(6):889–895, 2003.

[28] J. M. Magerlein and J. B. Martin. Surgical demand scheduling: a review. *Health services research*, 13(4):418, 1978.

[29] R. T. Maheswaran, J. P. Pearce, and M. Tambe. Distributed algorithms for dcop: A graphical-game-based approach. In *PDCS)*, pages 432–439, September 2004.

[30] R. T. Maheswaran, J. P. Pearce, M. Tambe, et al. Distributed algorithms for dcop: A graphical-game-based approach. In *ISCA PDCS*, pages 432–439. Citeseer, 2004.

[31] A. Meisels. *Distributed Search by Constrained Agents: algorithms, performance, communication*. Springer Science & Business Media, 2008.

[32] P. J. Modi, W.-M. Shen, M. Tambe, and M. Yokoo. Adopt: asynchronous distributed constraints optimizationwith quality guarantees. *Artificial Intelligence*, 161:1-2:149–180, January 2005.

[33] A. Netzer, A. Grubshtein, and A. Meisels. Concurrent forward bounding for distributed constraint optimization problems. *Artificial Intelligence Journal (AIJ).*, 193:186–216, 2012.

[34] S. Okamoto, R. Zivan, and A. Nahon. Distributed breakout: Beyond satisfaction. In *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence, IJCAI 2016, New York, NY, USA, 9-15 July 2016*, pages 447–453, 2016.

[35] J. Pearl. *Probabilistic reasoning in intelligent systems: networks of plausible inference*. Elsevier, 2014.

[36] A. Petcu and B. Faltings. A scalable method for multiagent constraint optimization. In *IJCAI*, pages 266–271, 2005.

[37] Z. H. Przasnyski. Operating room scheduling: A literature review. *AORN journal*, 44(1):67–82, 1986.

[38] C. R. Reeves, editor. *Modern heuristic techniques for combinatorial problems*. John Wiley & Sons, Inc., New York, NY, USA, 1993.

[39] A. Rogers, A. Farinelli, R. Stranders, and N. R. Jennings. Bounded approximate decentralized coordination via the max-sum algorithm. *Artificial Intelligence*, 175(2):730–759, 2011.

[40] S. Russell and P. Norvig. Artificial intelligence: a modern approach. 2002.

[41] P. Rust, G. Picard, and F. Ramparany. Using message-passing DCOP algorithms to solve energy-efficient smart environment configuration problems. In S. Kambhampati, editor, *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence, IJCAI 2016, New York, NY, USA, 9-15 July 2016*, pages 468–474. IJCAI/AAAI Press, 2016.

[42] A. Schoneveld, J. F. de Ronde, and P. M. A. Sloot. On the complexity of task allocation. *Journal of Complexity*, 3:52–60, 1997.

[43] Y. Shoham and K. Leyton-Brown. *Multiagent systems: Algorithmic, game-theoretic, and logical foundations*. Cambridge University Press, 2008.

[44] S. S. Skiena. *The algorithm design manual*, volume 2. Springer, 1998.

[45] P. Stone. Learning and multiagent reasoning for autonomous agents. In *IJCAI*, volume 7, pages 13–30, 2007.

[46] Y. Sun and X. Li. Optimizing surgery start times for a single operating room via simulation. In *Proceedings of the 2011 Winter Simulation Conference (WSC)*, pages 1306–1313. IEEE, 2011.

[47] P. P. Wang. Static and dynamic scheduling of customer arrivals to a single-server system. *Naval Research Logistics (NRL)*, 40(3):345–360, 1993.

[48] W. Yeoh, A. Felner, and S. Koenig. Bnb-adopt: An asynchronous branch-and-bound dcop algorithm. *Artificial Intelligence Research (JAIR)*, 38:85–133, 2010.

[49] T. Ze'evi, R. Zivan, and O. Lev. Socially motivated partial cooperation in multi-agent local search. In *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems, AAMAS 2018, Stockholm, Sweden, July 10-15, 2018*, pages 2150–2152, 2018.

[50] W. Zhang, Z. Xing, G. Wang, and L. Wittenburg. Distributed stochastic search and distributed breakout: properties, comparishon and applications to constraints optimization problems in sensor networks. *Artificial Intelligence*, 161:1-2:55–88, January 2005.

[51] Z. Zhao and X. Li. Scheduling elective surgeries with sequence-dependent setup times to multiple operating rooms using constraint programming. *Operations Research for health care*, 3(3):160–167, 2014.

[52] R. Zivan and A. Meisels. Message delay and discsp search algorithms. *Annals of Mathematics and Artificial Intelligence*, 46(4):415–439, 2006.

[53] R. Zivan, S. Okamoto, and H. Peled. Explorative anytime local search for distributed constraint optimization. *Artificial Intelligence*, 211, 2014.

[54] R. Zivan, T. Parash, L. Cohen-Lavi, and Y. Naveh. Applying max-sum to asymmetric distributed constraint optimization problems. *Autonomous Agents and Multi-Agent Systems*, 34(1):1–29, 2020.