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 scheduling of operations in a large hospital is done jointly by several groups of people, each with its own objectives and 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 and for each day. The final schedule must satisfy all hard inter-ward constraints, such as the allocation of anesthetists, nurses, and equipment to operations occurring in parallel, and should address soft constraints such as accounting for the urgency and complexity of operations.

In addition to assigning operation requests and surgeons to time slots in the operating rooms, the final daily schedule assigns the shared resources of the surgical wards, such as anesthetists, nurses, and the required equipment.

The present work contributes to the ongoing effort to adapt multi-agent optimization models and algorithms to real-world applications by modeling problems in two phases as distributed constraint optimization problems with different properties. The first phase includes partially cooperative agents, which represent wards, allocating operating rooms among themselves for daily use. In the second phase, agents representing the wards interact with agents representing the constraining elements to generate daily operating schedules for each operating room, thus forming a unique bipartite constraint graph. On one side of the graph are the ward agents, and on the other side are the agents of the constraining elements. Each agent has a nontrivial local problem to resolve whose solution serves as a proposed assignment in the distributed algorithm.

We discuss the properties required by the algorithms to solve the two phases, adjust the existing distributed partial cooperative and local search algorithms to solve these problems, and compare algorithms that implement different approaches. The results in both phases emphasize that successful collaboration requires that agents have consistent information about their peers' states and that the degree of exploration that the algorithm implements must be restricted 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 study and development of multi-agent optimization models and algorithms considered abstract problems such as random uniform constraint graphs, graph coloring, and scale-free networks. Over the past two decades, researchers in this community emphasized the importance of identifying practical applications that required the application of these models and algorithms. Today, the emergence of research related to the Internet of Things has produced a set of relevant applications in which devices interact, thereby requiring distributed models and algorithms to solve them [12, 41]. However, much of the motivation to solve distributed problems stems from human interests (e.g., privacy, cooperation), as put forward by the system's agents. Thus, practical applications that include representative human agents remain uncommon, so the introduction of such applications is a significant relevant challenge that merits attention.

Many real-world scheduling problems include conflicting interests between interdependent entities. An intuitive example is the scheduling of activities that must call upon a limited set of resources. Solving such problems requires the interaction of autonomous entities (agents), each with its own objectives and constraints. The goal is that the final schedule satisfies not only the hard inter-group constraints but also the soft constraints. Examples of such problems include the scheduling of dentist appointments, automobile repair, job interviews, and the scheduling of hospital operating rooms (ORs). All these problems have a similar combination of properties that include a level of urgency, the requirement of experts and equipment, and a specially equipped room or venue in which the scheduled process will occur. Such problems have a natural distributed structure in which each autonomous entity participating in the scheduled task has its own requirements for optimizing its performance and its own private preferences and constraints.

This study focuses on the scheduling of operations in a large hospital. Such hospitals can have several operating theaters, each with ten or more ORs. OR teams are staffed by multiple professionals, each of whom is scheduled. In addition to surgeons, these teams include nurses, anesthetists, technicians, and more. Furthermore, operations lasting over a single day are performed by multiple doctors that belong to different wards and who operate on patients that must be properly prepared for the given operation [26]. Since operating theaters, their teams, the operating doctors, and the patients are involved in a crucial and very costly part of a hospital's activity,efficient, computerized management can contribute significantly to a successful operation [1]. However, the procedure to schedule operations is not only complex and time-consuming but also cumbersome and discouraging for all stakeholders.

Several groups of professionals are involved in developing and managing the scheduling (time, date, and room) of operations. Although all such groups share a common overall goal, each has different constraints and preferences and even different sub-goals and objectives that they hope to satisfy. Additionally, multiple wards are available, each managing its operating schedule in the rooms as a function of the allocated date. Each ward has preferences regarding the days and the rooms in which its operations are scheduled [27]. After being allocated ORs on specific dates, the ward assigns its patients' operations to one of the ORs allocated to the ward on the given date and determines the daily schedule in each OR. Daily schedules must be coordinated with the schedules of other ORs such that inter-constraints (i.e., the availability of specialty equipment, skilled anesthetists, etc.) are satisfied [51].

We separate the discussion into two phases, both of which must be done for operations to go forward [15]. The first phase involves the daily allocation of ORs to wards. Different wards in the hospital have different needs for ORs. For each day, each room is allocated to a single ward. Constraints define which rooms support which type of operation, the level of concurrency at which wards can perform operations, the preferences of each ward regarding the ORs and the days of the week on which they can be allocated, and the cardinal needs of the ORs of each ward [3].

The second phase involves generating the daily operating schedules for each ward. As mentioned above, the daily schedule for each OR must consider the availability of the resources required for the given operations (e.g., nurses, anesthetists, equipment). These aspects produce a natural multi-agent problem in which the agents representing the ward must coordinate their decisions with the agents representing the constraining and shared elements. Each ward determines the schedule for a given date for each OR allocated to it on that date. The assignments of all elements involved in the operations are considered, and the resulting multi-agent optimization problem takes the form of a bipartite graph. On one side of the graph are the ward agents, each representing a hospital ward, and on the other side are the agents representing the *constraining elements* (CEs), such as the head nurse, the anesthetist, and the equipment (others may be involved such as technicians or unlicensed assistive-personnel representatives). Thus, agents making these assignments must consider both the internal ward and the medical constraints, such as the urgency of the operation and the availability of surgeons. In addition, all inter-ward constraints and management preferences must be considered, such as the availability of the required equipment and personnel [6].

The final daily schedule assigns the operation requests, surgeons, and required equipment to time-slots and ORs. Although the participants are all autonomous entities, they all belong to the same hospital, so they share common goals, such as the success of the operation, the reputation of the hospital, and its financial success.

Multi-agent optimization scenarios are commonly represented as distributed constraint optimization problems (DCOPs) [32, 36, 39, 48, 53, 8]. When agents place different values on the possible outcomes of such multi-agent problems (i.e., have different constraints), the appropriate model to represent the problems is the *asymmetric distributed constraint optimization problem* (ADCOP) [20, 7]. The problems considered herein are indeed asymmetric: For the problem of allocating rooms each day, different wards have different needs and thus place different values on different allocations. Moreover, because all wards are part of the same hospital, each ward desires the other wards to succeed as well, so agents representing wards are partially cooperative [22, 49]. For the problem of making the daily inter-ward schedule, the agents representing the constraining elements (e.g., nurses, anesthetics, technicians) 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 NP-hard, so the enormous size of the problem at hand rules out complete ADCOP algorithms. Consequently, this study proposes ADCOP-based models to represent the problems and distributed incomplete local search algorithms to solve them. More specifically, we make the following contributions to the research on realistic implementations of multi-agent optimization models and algorithms:

1. We propose an extension of the socially motivated partial cooperative model (introduced in Ref. [49]), which targets periodic indivisible resource allocation and applies to the daily allocation of ORs to wards in large hospitals.
2. We propose a bipartite distributed constraint optimization model, which is an extension of the ADCOP and represents problems that include agents that attempt to schedule elements constrained by the availability of resources, which themselves are controlled by other agents. All agents seek a schedule that does not violate hard constraints and that 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 ORs allocated to wards on specific dates.

3. We propose adjustments of distributed local search algorithms for solving the two models representing the resource allocation problem and the scheduling problem, in which agents solve their local problem by using a centralized heuristic (e.g., simulated annealing) and exchange assignments with their neighbors to resolve inter-constraints.
4. We present experimental results that demonstrate the ability of the proposed algorithms to solve 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 altruistic agents more than they benefit their neighbors. Conversely, exchanged indications regarding agent preferences of their neighbors' actions lead to high-quality solutions. Furthermore, the results for the daily operating schedule reveal the importance of applying stable and incremental improvements of interim solutions. They also reveal that a limited level of exploration is required to obtain high-quality solutions. This level of exploration can be achieved by applying a limited number of revisions in each algorithm iteration or by penalizing schedule revisions so that only highly beneficial modifications are performed.

The remainder of this work is organized as follows: Section 2 presents the background of operating-room planning and scheduling, distributed artificial intelligence, multi-agent systems, DCOPs, asymmetric DCOPs, algorithms for solving DCOPs, and centralized local search algorithms. Next, models for representing the problems in the two phases are explained in Section 3, and the algorithms are described in Section 4. Section 5 evaluates the experimental results, and the conclusion is made in Section 6.

2. Background

This section provides the requisite background of operation scheduling and distributed optimization problems and of the local search algorithms used to solve them.

2.1. *Operating Room Planning and Scheduling*

The directorial view of supplying health services to hospital patients is becoming progressively more crucial. Hospitals want to decrease costs and increase their level of use, all while maximizing patient contentment. The operating theater is the hospital's main cost and revenue center [10] and significantly affects the hospital's performance. However, managing an operating theater is challenging because of a lack of expensive resources and the stakeholders' conflicting priorities and preferences. This situation emphasizes the need to improve productivity and requires the development of efficient planning and scheduling procedures.

Advanced scheduling is the method by which surgery dates are set for patients. Conversely, allocation scheduling determines the OR and the starting time of the surgery on the day that the surgery is scheduled [28]. Reference [37] arranges the literature on OR scheduling based on broad areas of interest, such as cost control or the scheduling of specific resources. Two main groups of patients are considered in the literature on OR scheduling: elective and non-elective patients. The former group includes patients whose surgery can be scheduled in advance, whereas the latter group of patients must be urgently and unexpectedly operated [6].

Previous studies on scheduling surgery can be separated into two branches: First, the single-OR-scheduling problem defines the start times for a set of surgeries in an OR on a given day [9, 46, 47]. Second, multi-OR scheduling problems (including the present study) [2, 24] consider the scheduling of parallel surgeries in various ORs.

These surgery-scheduling problems have been solved by diverse methods that can be divided into four categories: queuing methods, simulation methods, optimization methods, and heuristic methods [6, 11, 14]. Queuing methods are typically used to solve single-OR-scheduling problems. Simulations can be used to evaluate several scheduling heuristics and are adjustable, which allows researchers to model ambiguity in surgery scheduling. Optimization methods mostly involve the use of deterministic or stochastic integer or mixed-integer programming models and algorithms. Aside from exact methods, some heuristics have been applied, such as simulated annealing, taboo searching, and genetic algorithms. In addition, centralized constraint-programming approaches have been proposed for solving the surgical-scheduling problem [51]. In contrast with our appraoch, all of the above approaches require that problems constraints and preferences be constrained to a single entity that solves the problem.

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 how one entity affects the decisions and actions of the other entity [4]. Such an entity is commonly called an agent, which can be a physical or virtual autonomous entity that can act, perceive its environment (sometimes in a partial way), communicate with others, and achieve its goals [16, 40]. Distributed artificial intelligence focuses mainly on the coordination between multiple autonomous agents, which is described by the interaction between behavior, knowledge, goals, skills, and the programs of agents [4].

A particular environment in which a number of agents interact to pursue some set of goals or perform some set of tasks is known as a “multi-agent system,” which 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 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 the agents of other systems. Systems in which several agents interact to maximize utility and jointly solve tasks are called cooperative multi-agent systems [4, 45].

2.3. Distributed Constraint Optimization

A DCOP is a framework used that characterizes 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 because of, for example, autonomous decisions by the agents, user privacy, or the infeasibility of centralization. They usually involve many interdependent agents, can be represented by a graphical model, and are solved by using message-passing algorithms. DCOPs have a broad range of applications in multi-agent systems [33] and present a scientific challenge because they require the cooperation of various agents (who are 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 or constraints. Values assigned to the variables held by agents are taken from finite, discrete domains, and the agents interact via messages to coordinate the selection of values for their variables, with the aim being to optimize a given global function, which is commonly to minimize or maximize the total costs or utilities of the set of constraints between variables. Constraints between variables that may be held by distinct agents define the costs incurred or the utility derived from the given combinations of variable values.

The following formal description of a DCOP is consistent with the definitions used in numerous DCOP studies (see, e.g., Ref. [32]). A DCOP is a tuple $\langle \mathcal{A}, \mathcal{X}, \mathcal{D}, \mathcal{R} \rangle$, where $\mathcal{A}$ is a finite set of agents $\{A_1, A_2, \ldots, A_n\}$, and $\mathcal{X}$ is a finite set of variables $\{x_1, x_2, \ldots, x_m\}$. A common assumption used to treat DCOPs is that every variable is held by a single agent. $\mathcal{D}$ is a set of domains $\{D_1, D_2, \ldots, D_m\}$, where each domain D_i contains the finite set of values that can be assigned to the variable x_i. Assigning the 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 (i.e., constraints), where each constraint $C \in R$ defines a non-negative cost for every possible combination of values from a set of variables and is of the form $C : D_{i_1} \times D_{i_2} \times \cdots \times D_{i_k} \to \mathbb{R}_+ \cup \{0\}$. A binary constraint refers to precisely two variables and is of the form $C_{i_j} : D_i \times D_j \to \mathbb{R}_+ \cup \{0\}$. A binary DCOP has only binary constraints. A partial assignment (PA) is a set of value assignments to variables in which each variable appears only once. $vars(PA)$ is the set of all variables that appear in PA. A constraint $\mathcal{C} \in \mathcal{R}$ of the form $C : D_{i_1} \times D_{i_2} \times \cdots \times D_{i_k} \to \mathbb{R}_+ \cup \{0\}$ 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.

A DCOP can be used to characterize a wide variety of multi-agent systems in which agents need to cooperate to attain a common goal. Examples of these applications include automated meeting scheduling by intelligent calendars, mobile sensor nets, Internet-of-Things applications such as an operating schedule for electronic devices in smart homes, and resource allocations [17, 20, 30].

2.4. *Asymmetric Distributed Constraint Optimization Problem*

In a DCOP, all agents involved in each constraint incur equal costs. Therefore, the DCOP definition does not correctly characterize real-life problems in which agents assign different values to decision outcomes [20]. For instance, in meeting-scheduling problems, agents may assign different valuations to the meetings to which they are summoned, a scenario that cannot be captured by the standard DCOP model [54].

ADCOP was introduced in Ref. [20]; the new expanded framework allows each agent to determine its own valuation cost for each constraint it endures. ADCOPs generalize DCOPs by explicitly defining for each combination of assignments of constrained agents the exact cost for each participant in the constraint [20]. Combinations of value assignments are mapped to a tuple of costs, one tuple element 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 as for the DCOP. Each constraint $C \in R$ of an ADCOP defines a set of non-negative costs for every possible value combination of a set of variables and takes the form $C : D_{i_1} \times D_{i_2} \times \cdots \times D_{i_k} \to \mathbb{R}_+^k \cup \{0\}$. 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 $C_j, C_j : D_{i_1} \times D_{i_2} \times \cdots \times \ldots D_{i_k} \to \mathbb{R}_+ \cup \{0\}$ such that its privacy is maintained. As in the DCOP, an optimal solution to an ADCOP is a complete assignment of values to all variables with a minimal sum of agent costs.

2.5. Algorithms for Solving Distributed Constraint Optimization Problems

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

Researchers have proposed 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, the use of these algorithms is effectively restricted, especially for practical applications that primarily involve significant problems with a large set of variables. Thus, the incentive is strong to develop incomplete algorithms [13, 34, 50].

Although incomplete methods run sufficiently quickly and efficiently to be applied to the realistic problems mentioned above, they are not guaranteed to find optimal solutions [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 the influence of each new datum concerning the constraint costs. The beliefs propagate through the graph via message-passing between neighboring variables [35].

Search algorithms (distributed stochastic algorithm, max gain Messages, 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. Next, 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 step (or iteration) of the algorithm, an agent sends its assignment to all its constraint-graph neighbors and receives the assignments of all neighbors. They vary in the approach agents use to determine whether to change their current value assignments for their variables [53].

“Anytime” algorithms hold the best assignment of the search. In these algorithms, the anytime feature ensures that the solution’s value remains the same or improves if the algorithm is further iterated. This property is not easily assured in distributed environments. In these environments, agents are only aware of the cost of their own assignment (and maybe that of their neighbors), but none 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 Ref. [53].

The suggested framework uses a BFS tree to accumulate the costs of the system's state during the algorithm's iteration and to propagate the detection of the best new state when found. The proposed framework does not incur additional network load.

2.5.2. Non-Concurrent Atomic Operations

Evaluating the runtime performance of distributed algorithms is not trivial. Agents may be executing on machines with different hardware, the implementation quality may affect the runtime, and some of the actions of the agents can be done concurrently, whereas others cannot. Therefore, to detect the sequence of implementation-independent actions agents should perform, we must determine which of the operations performed by agents cannot be performed concurrently. This sequence constitutes an implementation-independent performance measure of distributed algorithms in a distributed environment. Thus, the runtime performance of the algorithm is the most extended non-concurrent sequence of operations that the algorithm performs. Reference [52] suggests a uniform method to measure and compare their performance; this method allows the different DCOP algorithms to be evaluated on a consistent scale. The straightforward concept is to measure the most extended sequence of non-concurrent atomic operations [33] (e.g., constraint checks). We adopt this approach herein because we evaluate the quality of the solutions of the algorithms as a function of the asynchronous advancement of the algorithm when agents perform concurrent computations.

2.5.3. Distributed Stochastic Algorithm

The distributed 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 (iterative loops) until some termination condition is met. In each step, the agent sends its value assignment to its neighbors in the constraint graph and collects the value assignments of its neighbors. Once the value assignments of all neighbors are collected, the 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 a DSA cannot upgrade its current state by substituting its present value, it does not replace it. Conversely, if the agent can the cost (or maintain the exact cost, depending on the version used), it determines whether to replace the values by applying a stochastic strategy.

2.6. Partial Cooperation

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

Definition 1. *Denote by μ_i the* **baseline 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.*

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 $N(i)$;
```

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 from the set of all possible outcomes O that satisfies the condition*

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

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, while constantly seeking globally improved solutions, the personal cost of an agent does not exceed the predefined cooperative intention limit. Agents executing this algorithm exploit possible improvements until the solutions converge to some local optimum that cannot be further improved without breaching the cooperation bound of one of the agents. Before replacing a value assignment, agents request their neighbors' approval, which is accorded only if the value-assignment replacement does not breach the cooperative bound for any of the neighbors. Only if all neighbors approve does the agent replace her value assignment.

The pseudo-code for the AGC algorithm is given in Algorithm 1 and emphasizes the three phases that constitute each step of the algorithm. The algorithm begins after agents compute a baseline assignment by performing a simple non-cooperative interaction between them. Thus, the

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 $N(i)$;
```

agent can select her baseline value assignment and can 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 (i.e., an assignment replacement) that maximizes her gain and sends to her neighbors a suggestion to perform it along with the gain expected from this action. Next, agents receiving the suggested actions of their neighbors approve one that they deem acceptable and send Neg! messages to the rest. In the third phase, agents that did not get a Neg! message from their neighbors perform their proposed assignment replacement.

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. To allow agents to exploit the cooperative intentions of their neighbors and thereby improve the solution's quality, Ref. [49] proposed an approach that involves a partial cooperative local search in which agents take an extra step in the interaction process before selecting an assignment. 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 change their current value assignment). After exchanging this information, agents attempt to find an alternative value assignment, taking into consideration their own preferences and the indications received from their neighbors. This approach was combined with the AGC algorithm to form the socially motivated AGC (SM_AGC) [49].

Algorithm 2 gives the pseudo-code for the SM_AGC algorithm. Like the original AGC algorithm, the SM_AGC algorithm begins after agents compute a baseline assignment and use the baseline cost as a reference point. Like the AGC algorithm, after exchanging their value assignments, the agents loop over the four phases of the algorithm until satisfying a termination condition.

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

The DSA algorithm is both uniform (i.e., it does not use agents' identities) and synchronous: in each iteration, the agents send messages to all their neighbors and wait to receive the return messages before advancing to the next iteration [50].

2.8.1. Simulated Annealing

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

SA is a state-of-the-art metaheuristic for assessing global optimization in a sizable domain [44]. Although the algorithm accepts occasional transitions that lead to more pricey solutions, it avoids getting trapped in local optima. In the algorithm, the interpretation of the gradual cooling of metal takes the form of a gradual decrease in the probability of accepting worse solutions as it explores the solution space. Obtaining worse solutions is an essential property of metaheuristics because it allows for a broader search for the optimal solution (i.e., it explores the solution space).

At the beginning of the search, the algorithm widely explores the search space, so the probability of accepting a negative transition is high. As the search continues, the changes are 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. Problem Formalization

This section presents a model that represents each of the problems at hand, followed by their implementation as an ADCOP.

[1] Ω is used to assess the weights of the preferences of neighbors. For more details, see Ref. [49]

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 $W = \{W_1, W_2, \ldots, W_n\}$ and a set of m pairs of the form $\langle room, date\rangle$, $RD = \{RD_1, RD_2, \ldots, RD_m\}$. In this problem, the atomic time unit in which a resource can be allocated is a day, and the number of days in which we allocate rooms (i.e., 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, the allocation to a ward of a room at some date creates the 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.

Each ward W_i has two bounds and a cardinal constraint CC_i that defines the utility it derives with respect to the number of RDs it receives in the specified time interval. The lower bound defines the minimal amount of RDs required in the time interval (LB_i), and the upper bound defines the maximal number of RDs that 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, which can be a fixed cost or a cost related to the amount of RDs allocated. The upper bound (UB_i) defines the number of RDs allocated to ward W_i such that the allocation of an additional RD to W_i does not increment its utility.

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

ORDA as an ADCOP. 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 these variables. Furthermore, the utility calculation must be decomposed into asymmetric constraints that agents (representing wards) can compute and aggregate. Agent A_i representing ward W_i holds variables $v_{i_1}, v_{i_2}, \ldots, v_{i_k}$, where k is the maximal number of resources that may be allocated. The domains include all 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 $q \geq 1$ assignments, and the utility the agent derives from this constraint (i.e., $c = [\langle A_{i_1}, RD_{j_1}\rangle, \ldots, \langle A_{i_q}, RD_{j_q}\rangle, u_i]$). 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 with which it is involved.

3.2. Operation-Day Scheduling

The operation-day scheduling 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, on one side, agents representing wards (WRs) that need to schedule operations in operation rooms assigned to them on specific days, and, on the other side, agents representing coordinators of CEs. The resulting structure is a bipartite graph.

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

C is the set of constraints and 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 the operation, etc. The constraints also define the utility derived from the combined assignment of the surgeon and the operation request. For example, if the assigned surgeon cannot perform the requisite type of surgery, the utility derived is $-\infty$, whereas, if the surgeon can perform the surgery, the utility is positive.

The CE agents solve a standard constraint optimization 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 because the variables represent requirements in ordered operation slots for all the ORs in the hospital. Thus, X is a table of n over k over r, where n is the number of ORs in the hospital, k is the maximal number of operations that can be performed in an OR 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 $x_{i,o,r}$ in the table represents an assignment of element r in operation o in OR 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, whereas 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 is standard (ADCOP), is the sum of utilities of all agents.

4. Local Search Algorithms

This section presents distributed incomplete local search algorithms for solving the problems modeled above. Although we use distributed local searching to solve the problems, the two models require the design of algorithms that implement different solution approaches. In the first algorithm, agents must balance between the requirements of the wards they represent and the global good of their hospital. Thus, partial cooperation algorithms are appropriate [22, 49]. The generation of daily room schedules is constrained by inter-ward resource constraints exist (i.e., limited resources required for performing operations) in addition to each ward's internal constraints. These are represented 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: on one side are wards representing agents and on the other side are agents representing the constraining elements.

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

Algorithm 3 AGC_ORDA

```
input: $baseLineAllocation_i$, $baseLineCost_i$, $\lambda_i$,

$alloc \leftarrow baseLineAllocation_i$;
$\mu_i \leftarrow baseLineCost_i$;
$localView \leftarrow null$;
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 $N(i)$;
```

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

To solve the ADCOPs representing ORDA problems, we adjust 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 expect to derive from the RDs that are released for their use or the increment in utility as a result of an exchange.

In more detail, the AGC_ORDA version of AGC (depicted in Algorithm 3) includes three synchronous phases (iterations) in each step of the algorithm. In the first phase, 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 reduce the agent's utility beyond her limitations, and sends an accept message to the proposer and Neg! messages to all other neighbors. In the third phase, requests that were not met with Neg! messages are performed regardless of whether they are transfers or RD exchanges. Notice that, in contrast with the standard version of AGC, here, only the agents involved in a request (i.e., the agent sending the request and the agent receiving it) must approve it for the request to take place.

A similar adjustment is required to use the SM_AGC algorithm in ORDA scenarios (see pseudo-

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 $N(i)$;
```

code 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 for the release of a RD held by the agent herself and selects the choice with the highest social gain (i.e., the mutual gain for her and the other agent involved). Notice that only the agents involved in the exchange of a resource affect the gain, so for each request it receives, an agent need only consider its own preferences and those of the sender of the request. After comparing the expected social gains of all possible requests she can send, the agent selects the request with the highest social gain and sends it to the relevant neighbor along with the social gain. The subsequent actions in the third and fourth phases of the algorithm are similar to those of the second and third phases of the AGC_ORDA algorithm described above.

Consider the example depicted in Figure 1, which deals with the hospital OR scheduling problem described above. The example includes three wards and two ORs that are allocated per day. Ward$_1$ can only use 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 given on its left (lower and upper bounds).

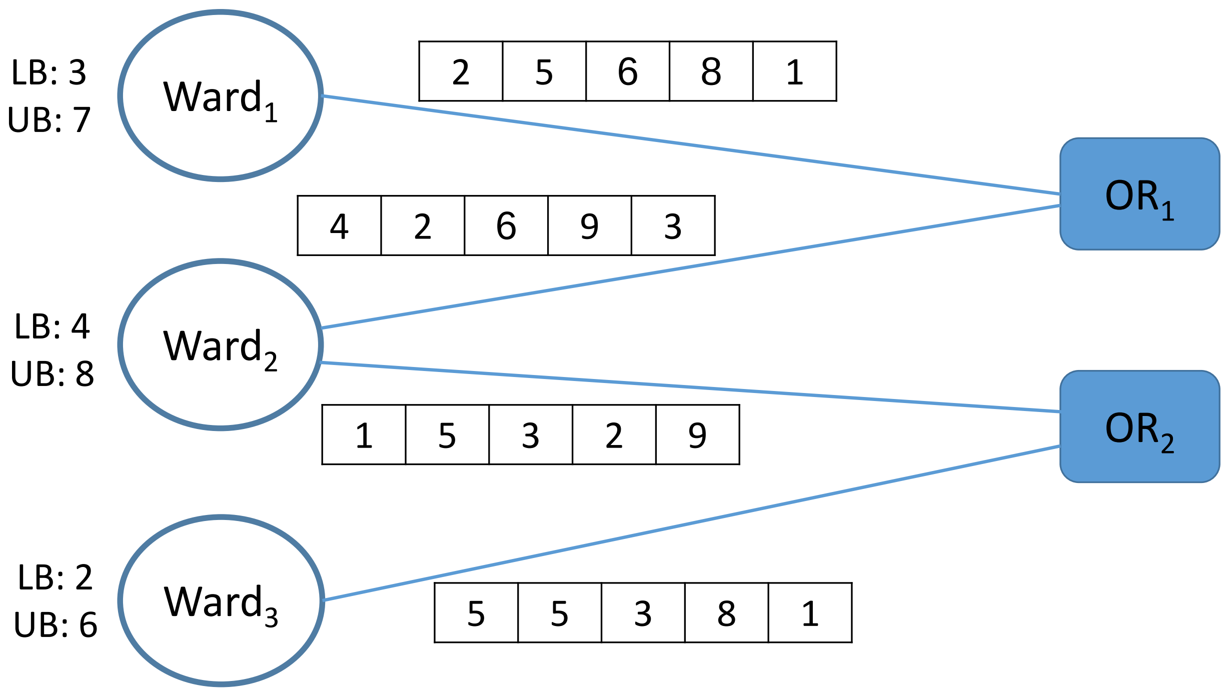


Figure 1: Hospital operating rooms example

We assume that a ward that does not satisfy its lower bound incurs a cost of 100. The allocation that the agents seek to schedule is for a five-day work 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 zero and nine. The preferences for day 1 are presented in the left entry of each array, followed by the preferences for day 2 and so forth.

Consider a situation in which the current allocation is as follows:

- OR_1 is allocated to $Ward_1$ o the first two days of the week.
- OR_1 is allocated to $Ward_2$ for the rest of the week (days 3–5).
- OR_2 is allocated to $Ward_2$ for the first three days of the week.
- OR_2 is allocated to $Ward_3$ for the last two days of the week.

If the agents are following 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 (because this ward is currently not satisfying its 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. The results are the same if the agents follow the SM_AGC algorithm because the preferences of $Ward_3$ are not relevant for OR_1 and the gain for $Ward_1$ is much greater than the loss for $Ward_2$. However, in the next iteration, if the agents are following AGC, $Ward_1$ can ask for OR_1 on day 3 as well and will get it for similar reasons. Conversely, if the agents are following SM_AGC, $Ward_1$ would not make this request because the social gain is negative.

At the same time, $Ward_3$ asks $Ward_2$ to exchange the allocations of OR_2 for days 1 and 5. $Ward_2$ has already agreed to release OR_1 on day 4 and therefore it sends a Neg! message to Ward 3. In the next step of the algorithm, Ward$_2$ agrees to exchange the days of OR$_2$ with Ward$_3$. The resulting allocation is OR$_1$ to Ward$_1$ on days 1, 2, and 4. OR$_2$ to Ward$_3$ on days 1 and 4, and the remaining allocations go 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 that represents 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 SA [38, 42] to generate the first solution to their local problem. In some versions, SA was used at each iteration of the distributed search.

We examine the two main approaches for designing distributed local search algorithms. In the first, inspired by the DSA [18, 50], an agent generates a solution to its local problem and sends it to its neighbors in the bipartite graph. Next, in each iteration, the agent searches for an improving assignment and, if found, replaces its current assignment with probability p (in our experiments we used $p = 0.7$). Notice that, in contrast with the standard DSA [50], the graph's structure is bipartite, so 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. We thus propose a query-response protocol in which the wards suggest schedules and the CEs react to these suggestions, specifying to which of the scheduled operations they could allocate the required element.

The pseudo-code for the proposed query response daily schedule algorithm (QRDSA) is presented in Algorithm 5. The main difference between the QRDSA and the standard DSA is the query response structure. Thus, the pseudo-code for the WR agents starts by selecting an assignment for their local problem using SA. Next, the WR agent w_i sends its selected schedule to its CE neighbors (in set CE_i) and waits for their responses. Once these are received, w_i updates its local information and revises its local assignment before resending it. Conversely, the CE agent ce_i waits for the assignments of its WR neighbors (WR_i) to arrive before it performs its computation.

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 $wr_j$
```

The CE agent updates its local operation schedule and proposes its corresponding assignment of CEs to this schedule. Each of its neighboring WR agents projects its assignment onto the schedule relevant to the neighbor.

To select the revised assignment in each iteration in both algorithms, we used the following three methods:

1. Single change: The variables are ordered and the agent searches for the first variable (operation) that did not receive all the elements required for the operation to go forward (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 value of the following ordered variable.
2. Single change with exploration: A random variable is selected. The agent tries to replace the current variable's assignment with an alternative value to improve its utility. If the variable change does not improve the utility, the variable's previous value is reassigned, and the another random variable is chosen. This procedure continues until a stop condition is met.
3. Simulated annealing: 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 changes in operation assignments (i.e., whenever a fully scheduled operation is moved to a different slot or postponed to an undetermined future date, the agent utility decreases). We examine the sf at two levels: at the first level, unsuccessful attempts are not recorded. At the second level, agents use a dynamic memory structure that stores visited solutions that are "no good" (sf_ng) throughout the algorithm's execution. The no-good visited solutions consist of surgery requests that were scheduled but did not result in a full allocation of all constraining resources in the expanded formulation. Scheduling a surgery request from the no-good structure reduces the agent's utility. Furthermore, the size of the penalty depends on the time interval required for the unsuccessful attempt to schedule the operation.

In all versions of the algorithm we use forward checking (i.e., values that are not consistent with previous assignments performed are removed from the domains). To select value assignments in the single-change versions, we used two methods from among the consistent values in the domains: The first method consisted of selecting a random value, and the second method consisted of selecting the value that seems most promising (the value expected to increment the utility the most). We demonstrate experimentally that the second method requires many more calculations that are not always beneficial.

Figure 2 presents an example of the daily schedule problem described above. It includes two wards and demonstrates the scheduling of a single day in which each ward is allocated a single OR. The problem further consists of a single CE agent that schedules the use of an indivisible resource (e.g., an x-ray machine). The day in this example is two hours long, and each operation lasts 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 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, so revisions must be made for the daily schedule to be feasible.

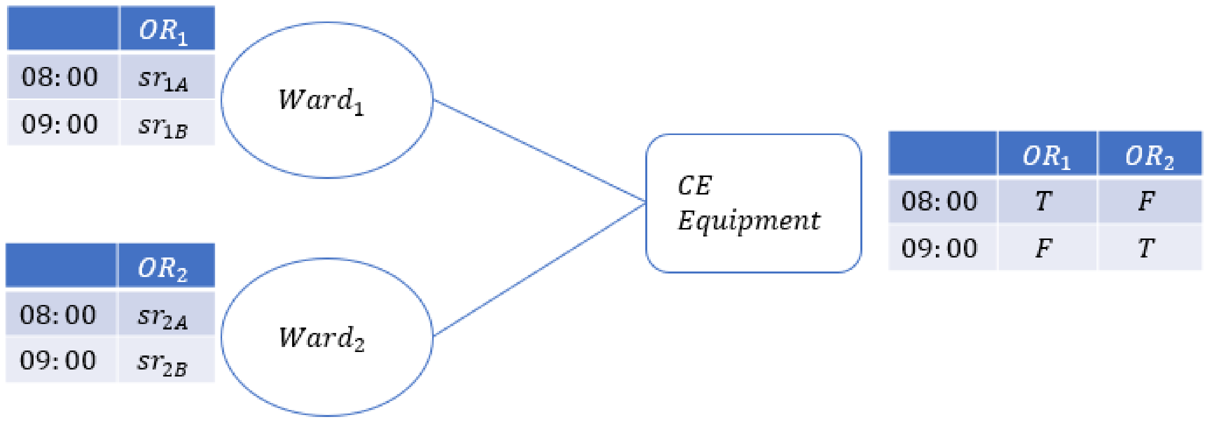


Figure 2: Example of daily schedule generation.

5. Experimental Evaluation

Our experiments included scenarios based on real data for both types of problems. To avoid a breach of privacy, we selected random values for some of the parameters of the problems. However, based on our analytical review of the data and on input from hospital personnel, the distributions from which the parameter values were selected were realistic. In all sets of experiments we used t-tests to examine statistical significance.

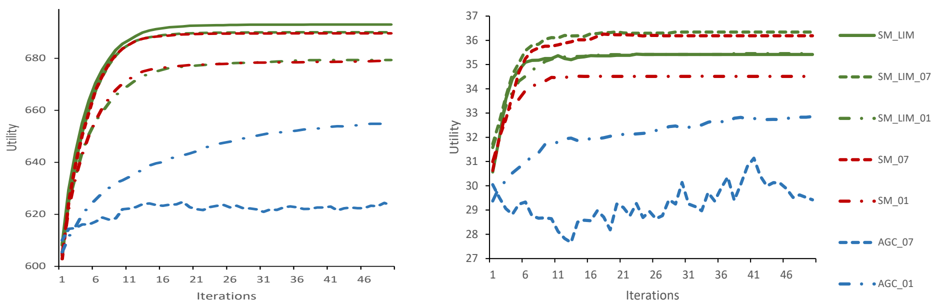
5.1. Evaluation of Room-Date Allocation Algorithms

This set of experiments included different versions of socially motivated local search algorithms, solving the hospital OR date-allocation problem. Agents represent hospital wards with different needs. The resources being allocated are ORs, each with specific properties, making the OR attractive to some of the wards and useless for others. The problem included 10 wards and 15 ORs allocated periodically every day. The allocation was for a five-day work week (i.e., each room was allocated five times). For each problem, the personnel constraints specifying the preferences of wards over the days of the week and the ORs were randomly selected from zero to nine. Of the 15 rooms, seven 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.

To examine how the results relate to the problem structure, we generated two additional less realistic sets of problems: A set of sparse problems in which every ward was interested in exactly two ORs, and a dense set for which each ward was interested in exactly five ORs. We refer to the three sets as *origin*, *sparse*, and *dense*, respectively.

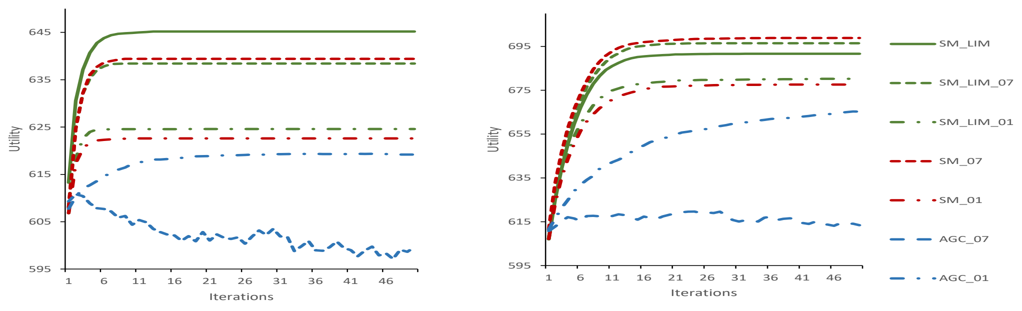
The versions of the partial cooperative local search algorithms we compared include (corresponding notations in brackets)

- 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 $\lambda = 0.7$ ($SM\_0.7$);



(a) (b)

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



(a) (b)

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

- 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 given 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 given above and as a function of the number of iterations preformed. Consistent with the results presented in Ref. [49], the current results demonstrate the clear advantage of the socially motivated algorithms over the standard AGC algorithms. Moreover, the results demonstrate that, to increase social welfare, intentions for cooperation (represented by λ) must be combined with preference sharing between agents. Thus, in all socially motivated versions, the $\lambda = 0.7$ versions outperform the $\lambda = 0.1$ versions. Conversely, the $\lambda = 0.1$ version is more successful for AGC algorithms. Finally, among the socially motivated versions of the algorithm, those using bounds are more successful.

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

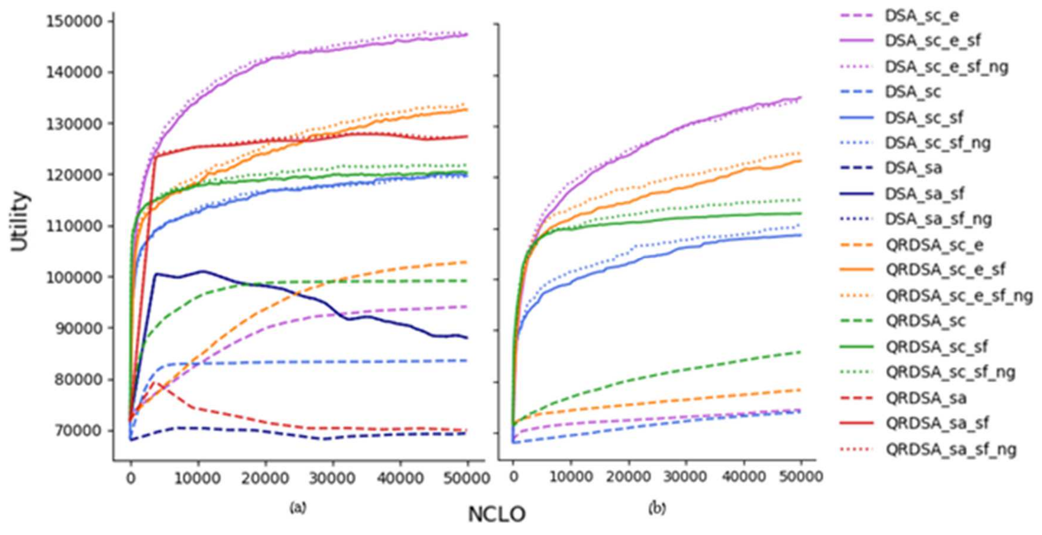


Figure 5: Average global utility for (a) origin set with best selection and (b) origin set without best selection.

Figures 4(a) and 4(b) present the social welfare of the allocations generated by the algorithms for the sparse and the dense synthetic problem sets, respectively. The most apparent difference is that, for the sparse problems, the version that only uses bounds is most successful, whereas, 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 ORs to be allocated to them, only the bounds are relevant, whereas, when more options are available, more refined intentions for cooperation are beneficial.

5.2. *Evaluation of Daily Schedule Algorithms*

This set of experiments includes different versions of local search algorithms to solve the ODSP and was implemented by using a simulator that represents the realistic distributed scheduling problem according to the model described. In all experiments the instance of the bipartite graph described above covers ten WR agents representing ten different surgical wards, and three CE agents representing the nurses, anesthetists, and equipment-allocation surgical coordinators.

All problems include 500 patients awaiting surgery. Each patient has a birth date sampled uniformly between 01/01/1925 and 01/01/2020 and each has at least a single surgery request. Different parameters define each surgery request:

- The type of surgery requested is uniformly selected from all the possible types of surgery offered by the ward.
- The number of cancellations (NC) is the number of times that a given surgery request was scheduled for surgery and then canceled. For every surgery request, the number of cancellations was sampled uniformly between zero and ten: $NC \approx Uniform(0, 10)$.
- The entrance or referral date is when the surgery request enters the hospital's queue of surgery requests awaiting surgery. The date is sampled uniformly from a period of a year before the scheduling day.

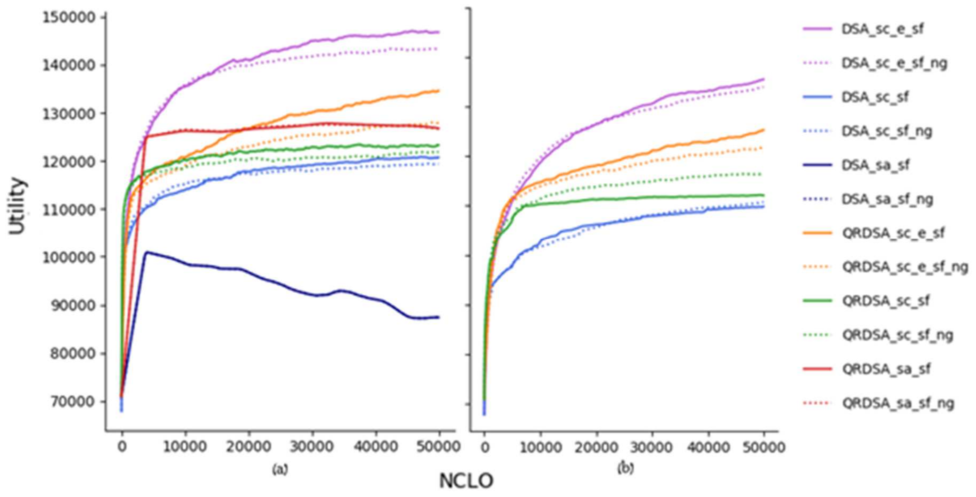


Figure 6: Average global utility for origin set with enlarged no-good (a) with best selection and (b) without best selection.

- A surgery request may or may not be assigned in advance with a specific surgeon. Usually, the surgery requests made in advance with a particular surgeon are unique and complex cases. To simulate this type of behavior for every ward, we used a random surgery type. All surgery requests of this surgery type were then assigned with a specific surgeon randomly selected from all highly graded surgeons qualified for this type of surgery.

Fifty surgical units are randomly (uniformly) assigned to the ten wards (assuring at least one unit per ward), and 300 surgery types are randomly assigned to the wards. For each type of surgery, the following parameters were selected: urgency, complexity, duration, and utility derived by the hospital. The hospital has six levels of urgency and surgery complexity, with level 1 being the lowest urgency and level 6 the highest. For each type of surgery, the urgency and complexity are chosen from the following discrete distributions:

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

The duration of each type of surgery is uniformly sampled from a minimum duration of 30 minutes and a maximum duration of the entire surgical day (L): $Duration \approx Uniform(30, L)$.

A random number of surgeons is assigned to each ward 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 three. Each surgeon has a set of graded skills that determines the surgeries that she is qualified to perform and her level of expertise. The set size differs between the surgeons and is randomly set between one and the number of surgery types supported by the ward. The level of expertise of each surgery type is also selected at random to assure that at least one surgeon is

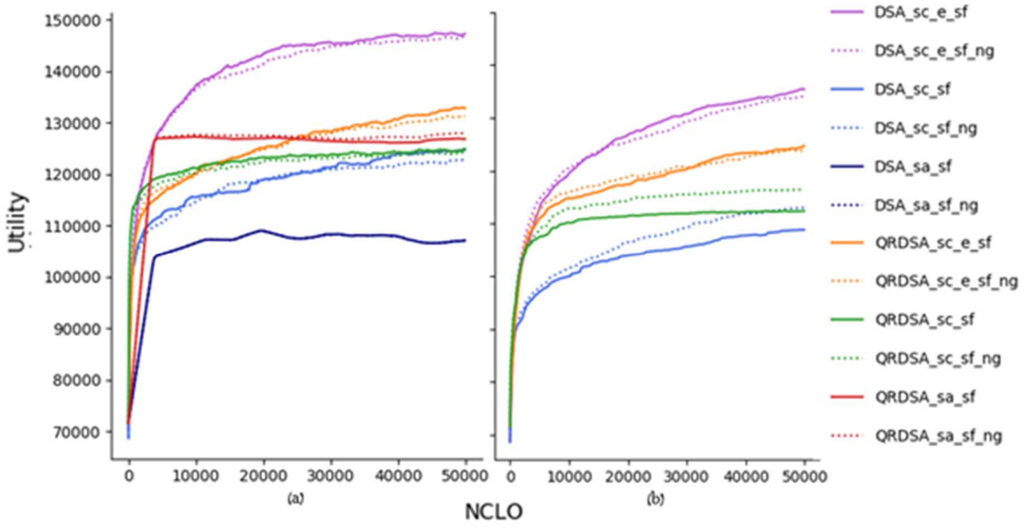


Figure 7: Average global utility for origin set with enlarged stability factor and no-good (a) with best selection and (b) without best selection.

qualified for each type of surgery. Finally, the surgeons perform surgeries in different shifts during the day.

Each problem includes 15 operation rooms, with each supporting a set of surgery types. For each OR, we sample a random number of surgery types between one and the number of surgery types supported by the hospital. Each ward has ORs allocated to it for specific dates. For the experiment at hand for each OR, a random ward is chosen from among all the wards compatible with the given surgery.

In addition, 100 nurses and anesthetists are available for each problem. The nurses have different skills that define the type of surgeries to which they can be allocated. For each type of surgery in the problem, a set of nurses is selected as qualified to perform the given type of surgery. The number of nurses is selected uniformly between 1 and the total number of hospital nurses (100 in this case). In the hospital, nurses are first qualified to be scrubbing nurses and only later become qualified as circulating nurses. Each nurse is eligible to participate as circulating nurse in a subset of the available surgery types. Nurses also perform surgeries in shifts. The number of nurses assigned to a surgical shift is precisely the number of nurses needed (i.e., twice the number of ORs because each surgery requires a circulating nurse and a scrubbing nurse). The nurses for each shift are selected randomly and uniformly from among all the nurses in the problem.

The anesthetists are divided by their experience into three ranks: intern, expert, and senior. Each rank defines a set of roles that the anesthetics are qualified to perform. For each problem, 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. For the remaining anesthetists, their ranks are

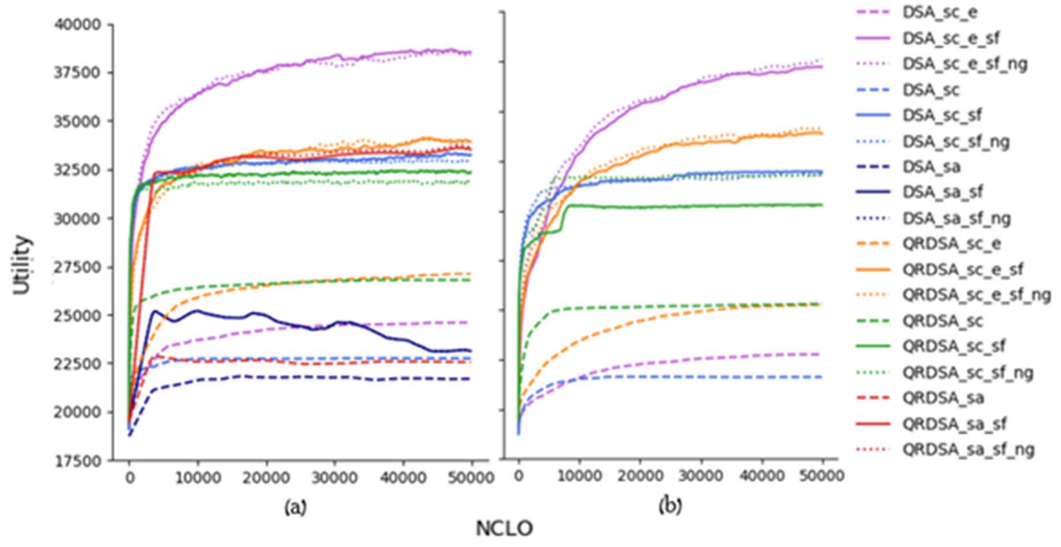


Figure 8: Average global utility for problems with five rooms (a) without best selection and (b) with best-selection.

sampled from the following discrete distribution:

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

The current ward rotation for intern anesthetists is selected uniformly and randomly from among all surgical wards. An intern is certified to perform surgery in the ward in which she did her previous rotations, but not for all types of surgeries. For each intern, a different number of wards are sampled to simulate her previous rotations. The number is determined uniformly between zero and the number of surgical wards in the hospital. For each type of surgery, the number of interns certified to perform the given surgery is sampled uniformly between one and the number of interns who are or were in rotation in its ward. Anesthetists also perform surgeries by shifts. Each shift is staffed by precisely the number of anesthetists needed according to the different ranks required by the various roles during the surgical day.

The problems include three types of equipment that are required for some types of surgeries. First, the units available for each type of equipment are selected randomly and uniformly between 1 and 15 (the number of ORs). Next, the surgery requests that required each piece of equipment is selected at random.

To examine how the results depend on the structure of the problem, we also examine four less realistic scenarios. We analyze how two of the problem's parameters affect the quality of the algorithms, and how the number of ORs in the hospital and the length of the operating day affect the outcome. The number of ORs is enlarged to 25 in one set of experiments to illustrate a sparser problem. In a second set of experiments, the operating day is prolonged to a ten-hour shift instead of a seven-hour shift. Finally, to demonstrate dense problems, two additional sets of experiments

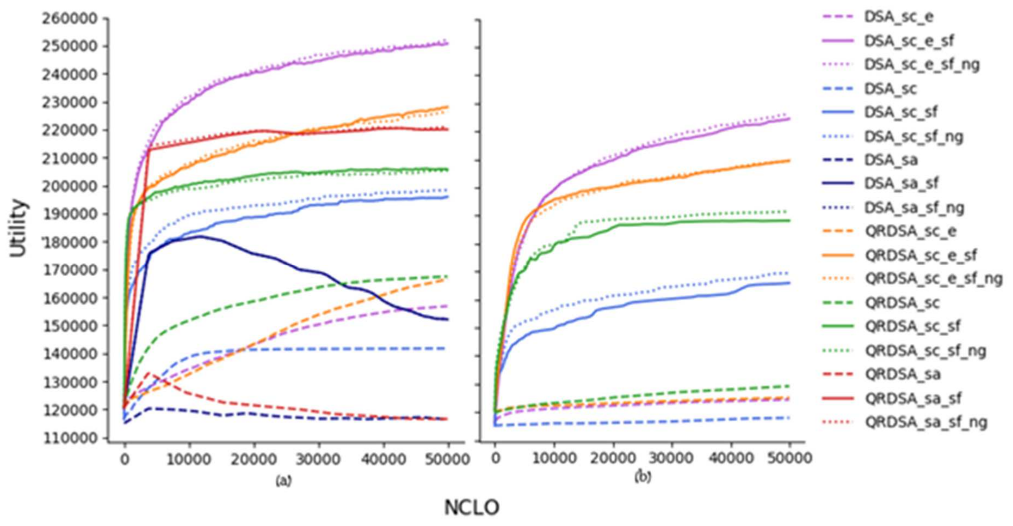


Figure 9: Average global utility for problems with 25 rooms (a) without best selection and (b) with best selection.

were conducted where we (i) decreased the number of ORs to five, and (ii) shortened the operating day to a four-hour shift. In addition, we analyzed the parameters to determine the adequate sf. Finally, all algorithms were assessed both with and without a sf. In addition, an additional set of experiments are done with the sf increased fifty-fold.

5.3. *Experimental Results*

The results compare algorithms implementing the two approaches: the DSA inspires the first, and the QRDSA inspires the second. For each approach, we implement identical versions of the algorithm as follows:

1. sc: single change;
2. sc_e: single change with exploration;
3. sa: SA in each iteration;
4. Addition of sf: addition of a sf;
5. Addition of ng: addition of a no-good dynamic memory structure to the sf;
6. addition of $best$-$selection$: addition of values selection from the domain.

We used non-concurrent logical operations as a time measure [33, 52], and each algorithm solved 50 random instances. The results present the average over the utility of the solutions produced by the different algorithms. The experiments were implemented using Python in the Pycharm IDE on a Windows Operating System.

Figure 5(a) shows the results for the original set representing the natural setting (i.e., seven-hour shifts and 15 ORs). Apparently, the algorithms that use a sf have a significant advantage over those that do not. A second observation is that the algorithms using single change with exploration produce better solutions for both the DSA and in QRDSA. All single-change versions of the DSA produce better solutions than the SA versions. Moreover, the quality of the solutions produced by the SA DSA deteriorate instead of improving during execution. For the QRDSA, the SA version outperforms the single-change version. When using single change with exploration, the DSA versions significantly outperform the QRDSA versions. However, when using SA, the QRDSA dominates. Note that the use of the no-good dynamic memory structure does not significantly modify the results.

Figure 5(b) shows the results of the algorithms, solving the origin set by using the best-value-selection method (best selection). Compared with the results shown in Figure 5(a), this method is clearly less successful than the method of random-value selection, which we attribute to the limited view of the agents on the constraints of the distributed problem (e.g., a WR can repeatedly request a particular surgery type, which from its point of view will contribute significantly to increasing its utility, but no qualified nurses are available on the given day for the given type of surgery. In addition, both the DSA and QRDSA in the single-change version and with exploration output an increasing trend after 50 000 non-concurrent logical operations. The moderate rate of increase can be explained by the large number of assessments for each change of value of a variable.

To assess the maximal influence of the sf on the algorithms, Figure **??** presents the results of the algorithms that include the sf penalty after increasing it fifty-fold compared with the sf used to produce the results presented in Figure 5. Figure 6 shows the results of the 50$\times$ sf versions of the algorithm with the penalty, resulting from a selected value from the no-good structure. Finally, Figure **??** presents the results of the sf versions of the algorithm with both sf and no-good penalties fifty-fold larger than those used to produce the results shown in Figure 5.

The results presented in Figures **??** and 7 indicate that this change mainly affects the SA versions, especially for the DSA, which does not deteriorate significantly upon using a larger sf. Figure 6 indicates that the no-good enlargement must be accompanied by an increased sf to make an impact. Furthermore, compared with broadening sf, the results do not change significantly upon increasing both sf and no-good.

Figures 8 and 9 show the solutions produced by the algorithms when the number of ORs available each day was either small (five for Figure 8) or large (25 for Figure 9). For few rooms, the problem becomes tighter. As a result, the advantage of the single-change exploration versions becomes more prominent, whereas the other algorithms using the sf (apart from the SA DSA) produce similar results. Conversely, when solving the looser version with 25 rooms, the deterioration of the SA DSA is steeper. Also, in the tighter scenario, the DSA implemented with best selection produces results similar to those of the DSA without best selection. It seems that, when the constraints of a problem tighten, the effort expended to select the best value is more beneficial, whereas a random selection has a lower probability to succeed, with a relatively minor gap between them when compared with the remaining setups.

Figures **??** and **??** present the results of the algorithms with the operating day either shorter (240 minutes in Figure **??**) or longer (600 minutes in Figure **??**). When the operating day is shorter, the problem becomes tighter; nevertheless, the advantage of the single-change explore version does not become more prominent. However, the results become sparser once the surgical day is prolonged, and the single-change explore algorithm produces the best solutions. In this scenario, the deterioration of the SA DSA is steeper in the tighter setting. Uniquely, the addition of the no-good memory structure in the 240-minute day implemented with best selection (i.e., with value selected from the domain), significantly improved the single-change versions in both approaches.

The results of all scenarios emphasize the importance of stability in a distributed environment where agents aim to resolve conflicts between themselves. If agents make too many changes to their local assignment, their neighbors make decisions based on obsolete information. Our results indicate that, in the SA algorithms, an agent changed an average of 17.67 operation assignments in each iteration in the original setup. In contrast, the single-change algorithms made at most a single change. Furthermore, when the algorithm explores the different possibilities for this single change, the effort becomes worthwhile.

6. Conclusions

The operating room allocation and scheduling application we present herein is a realistic application that requires distributed models and algorithms to preserve the natural structure of the problem and the autonomy and privacy of the patients involved. It includes agents representing wards (or ward directors), each with her own interest and who belong to a large organization (a hospital). Thus, a global mutual goal exists in addition to the personal goals of the agents.

Each of the two phases of the problem have unique properties that require the design of non-trivial models to be treated. In the operating room per date allocation problem, partially cooperative agents divide a mutual resource among themselves and maximize a global goal while satisfying their personal needs. In the daily schedule phase, wards optimize their complex operating schedule while interacting with agents that represent the hospital's constraining elements.

In future work we plan to explore similar realistic distributed applications and to investigate whether incomplete inference algorithms can solve 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.