Auton Agent Multi-Agent Syst (2015) 29:896–942
DOI 10.1007/s10458-014-9271-3

NB^3: a multilateral negotiation algorithm for large, non-linear agreement spaces with limited time

Dave de Jonge · Carles Sierra

Published online: 21 August 2014

Abstract Existing work on automated negotiations has mainly focused on bilateral negotiations with linear utility functions. It is often assumed that all possible agreements and their utility values are given beforehand. Most real-world negotiations however are much more complex. We introduce a new family of negotiation algorithms that is applicable to domains with many agents, an intractably large space of possible agreements, non-linear utility functions and limited time so an exhaustive search for the best proposals is not feasible. We assume that agents are selfish and cannot be blindly trusted, so the algorithm does not rely on any mediator. This family of algorithms is called NB^3 and applies heuristic Branch & Bound search to find good proposals. Search and negotiation happen simultaneously and therefore strongly influence each other. It applies a new time-based negotiation strategy that considers two utility aspiration levels: one for the agent itself and one for its opponents. Also, we introduce a negotiation protocol that imposes almost no restrictions and is therefore better applicable to negotiations with humans. We present the Negotiating Salesmen Problem (NSP): a variant of the Traveling Salesman Problem with multiple negotiating agents, as a test case. We describe an implementation of NB^3 designed for the NSP and present the results of experiments with this implementation. We conclude that the algorithm is able to decrease the costs of the agents significantly, that the heuristic search is efficient and that the algorithm scales well with increasing complexity of the problem.

Keywords Multilateral · Negotiation · Search · Non-linear utility · Negotiating Salesmen Problem

D. de Jonge (✉) · C. Sierra
Artificial Intelligence Research Institute, IIIA-CSIC, Campus de la Universitat Autònoma de Barcelona, Bellaterra, Catalonia, Spain
e-mail: davedejonge@iiia.csic.es

C. Sierra
e-mail: sierra@iiia.csic.es

1 Introduction

In multiagent systems (MAS) the outcome of the actions of an agent usually depends on the actions of other agents. These agents may have conflicting goals, and, since the other agents may be unknown and may not be benevolent, an agent generally cannot assume that other agents are willing to help without getting anything in return. If each agent would simply take those actions that are individually best, the result will often be sub-optimal for each of them, like in the well known prisoner's dilemma [38]. Therefore, agents in a MAS need to negotiate on what actions each will take. This is exactly what the field of automated negotiations deals with. If a Nash equilibrium [32] is not Pareto optimal, then negotiations allow the agents to reach a more efficient solution, with the commitment from each agent not to deviate from it.

In automated negotiations it is assumed that there exists a set of agreements that the agents can make with each other. We call this set the *agreement space*. Agents can propose agreements from this space to each other and can accept or reject proposals made by others. If a proposal is accepted, it means that each agent involved in the deal has committed itself to execute a certain set of actions. The execution of these actions then yields a certain amount of utility for each agent involved. Although each agent is only interested in optimizing its own utility, it may require the cooperation of the other agents to obtain this and therefore needs to make sure the other agents also receive enough utility to ensure their cooperation. We stress the fact however, that a good negotiation algorithm tries to exploit its opponents as much as possible and has no interest in reaching a social optimum.

1.1 Relation to other fields

Maximizing a utility function for a set of independent agents is also the goal of distributed constraint optimization problems (DCOP), but these problems are fundamentally different from negotiation problems, because DCOPs assume there is only one global function to be optimized and the agents cooperate with the joint goal of finding the solution that maximizes this global utility function [30]. Therefore, DCOP algorithms cannot be applied in cases where each agent is selfish and has its own utility function.

A field closely related to automated negotiations is the field of cooperative game theory [36]. In cooperative game theory one assumes that utility is assigned to coalitions of agents and that agents within such a coalition can freely divide the utility between one another. Such a division of utility is called an 'allocation' and the set of allocations that keeps the coalition stable is called the 'core'. The notion of an allocation in cooperative game theory can be compared to the notion of a deal in automated negotiations, and the notion of a coalition can be compared to a set of agents that together agree on a certain deal. The difference however is that cooperative game theory is mainly concerned with the question of whether the core and other solution concepts exist, while automated negotiations focus more on *how* agents decide to agree on a certain allocation.

Alternatively, negotiations can be modeled as a non-cooperative game, by modeling the proposals and acceptances of deals as the moves of a game. This is for example the approach taken in [1,2,23,32]. One can then try to apply techniques from non-cooperative game theory to find equilibrium strategies. We will however not do this, but take a heuristic approach instead.

1.2 Our contribution

In this paper we introduce a new family of negotiation algorithms, called Negotiation-Based Branch & Bound (NB^3), that applies a heuristic Branch & Bound search to explore the agreement space and determine which of the possible proposals are good enough to propose or accept. Our main motivation for this is that many existing papers make strong assumptions about the environment that we consider unrealistic in real-world negotiations.

Previously proposed negotiation algorithms have mainly focused on the utility values of deals, rather than the underlying deals themselves [12,13,31]. They assume that for a given proposal the utility for the agents is directly given, or can be calculated quickly. They describe a strategy to propose a deal characterized by a pair of utility values (u_1, u_2) at time t, but the deal itself is abstracted away and reduced to nothing more than this pair of utility values. They do not take into account that in reality, when you are negotiating about say, a car, for every offer made to you by the salesman, you first have to evaluate how much that deal is worth to you before you can make a decision.

The negotiation algorithm introduced in this paper negotiates explicitly over deals rather than over utility values. This is important for three reasons. Firstly, the evaluation of a deal may be computationally expensive, and therefore cannot be ignored in real-world situations. Secondly, if an opponent applies a different algorithm he may have different approximations of the utility values than you, so proposing a pair of utility values without specifying the underlying deal has no meaning. Thirdly, it is important not to reveal your valuation of a deal, as this is important strategic information.

Moreover, many papers assume that for any given utility value it is possible to find a proposal that exactly yields that value [24] (they assume the utility functions map surjectively onto an interval of the real numbers). This is often not the case, for example when the agreement space is discrete [44], when there are integrity constraints among the issues [48], or when there is no closed-form expression of the utility function [11].

While game theoretical approaches of negotiations often assume full knowledge, or partial knowledge about utility functions [2,15], most heuristic approaches assume that the utility values for the opponents of the agent are completely unknown [6]. Although it is true that one generally does not know the precise utilities for your opponents, we think that in a real-world negotiation you would at least know the preference order of your opponents, and you would have an approximate idea of your opponent's utility. Suppose for example that you are negotiating the price of a car. As a client you do not know precisely what would be the lowest price the dealer is willing to accept, but at least you know that the price of the car should, for example, be in the range of 10,000–20,000 Euro. Offering any price below this range would probably make negotiations fail, while offering a price above this range would be a very costly mistake. Also, you know that the aim of the car dealer is to make you pay a price as high as possible.

In this paper we model the fact that agents have approximate knowledge about their opponents' utilities by assuming that the *expressions* of the utility functions of the agents are publicly known, but evaluating these expressions to obtain utility *values* is costly in terms of time. Therefore, our agent can only make approximations of the opponents' utility values, and can only do so for a limited set of possible deals.

While most of the previous studies on automated negotiation assume strict negotiation protocols such as the Alternating Offers Protocol [40] to structure the actions of the agents, we assume an unstructured protocol that imposes almost no restrictions; the agents are allowed to say whatever they want and whenever they want, and are never required to reply to any proposal. This high degree of freedom introduces another dimension of complexity to the

scenario, but makes it closer to real-world negotiations and can therefore be applied, for example, in negotiations with humans.

In order to deal with this unstructured protocol we have developed a new negotiation strategy that not only determines what proposal to make next, but also takes time into account to determine whether the agent should really make a new proposal or rather continue searching for better proposals. This distinguishes our strategy from existing negotiation strategies.

Finally, in order to test our algorithm, we have defined a new negotiation game, which is a variant of the traveling salesman problem in which there are several salesmen that have to negotiate in order to minimize their individual path lengths. The complexity of the traveling salesman problem makes it a non-trivial task to calculate the utility of a given proposal, or to find a proposal with a given utility value, so traditional negotiation algorithms cannot be applied.

In short, we take the following assumptions into account:

- Utility is highly non-linear and calculating it or inverting it is computationally expensive.
- Solutions may involve a large number of agents, possibly including humans.
- The space of solutions is very large, i.e. there is no possibility to exhaustively explore the set of solutions.
- The environment changes during the negotiations due to actions of others.
- Other agents in the system are unknown.
- Decisions have to be made within a limited time frame.

Although many of these assumptions have been made before in existing work, to the best of our knowledge no algorithm exists that takes all of these into account.

Many difficult problems indeed require these assumptions to be made, e.g:

Online shopping When searching online for the best deal it would be convenient to have an automated agent that could negotiate on your behalf with the sellers. Especially if the product is composed of several smaller products such as a fully integrated holiday package this can get complex. In this case utility is difficult to calculate as your agent would need estimate for each deal under consideration how much you would valuate it. Furthermore, your agent should consider multiple sellers to buy from, the number of possible combinations of products can easily become very large, the environment changes since prices may change, new products may enter the market, or may sell out, the sellers may be anonymous and you may need to have the products before a certain deadline.

Time tabling School teachers have individual preferences for their teaching schedules. Once an initial schedule is determined by the school head, they may improve their particular allocations by negotiating local exchanges with fellow teachers. Teachers may for example want to avoid 'holes' in their schedules, or may want their schedules to be compatible with other activities (e.g. to practice sports). Current software solutions [45,46] are centralized and do not permit negotiation among teachers. Again, utility is difficult to calculate because a teacher's agent needs to estimate how much the teacher would valuate a certain schedule. Schedules may involve many teachers, the number of possible schedules that can be constructed is large, the set of feasible schedules changes as other teachers may come to agreements before you, making your preferred schedule impossible, and the final schedule has to be constructed before the start of the year.

Logistics A key issue in logistics is how to optimize multi-truck scheduling of package delivery between companies where every connection between nodes in the delivery network has a cost and every package delivery has a price. Current centralized systems [29] are not reactive enough to dynamic changes and call for a more efficient distributed solution where the negotiation of who transports what is done at the truck's and customer's level.

A simplified version of this problem with postmen exchanging letters was studied in [40]. In this case utility is a complicated function that depends on distances to cover, amount of time and fuel necessary for the delivery and the price payed by the customer. Many package deliverers may be on the road simultaneously, many packages may need to be delivered in a day, along many possible routes, the environment changes as new packages appear throughout the day and traffic jams may occur, deliverers may be from different companies and may therefore be unknown, and clients expect their packages to be delivered on time.

Diplomacy Diplomacy is a classical board game for seven players, without chance moves. This game is designed such that players need to form coalitions and therefore need to negotiate with their opponents in order to play well. An online community dedicated to the application of AI to Diplomacy has been developing software bots for many years [8], but most of those bots do not apply any reasonable negotiation techniques and are thus vulnerable when playing with humans that show great capacity in negotiation [26,34]. In this case, utility is defined by your probabilities to win the game, for which no explicit formula exists (like in chess). Deals are often made between 3 or 4 players. The agreement space of this game is very large: there are 34 units on the board, each of which has around 5 to 10 options for each turn, while a game typically takes more than 20 turns to finish. The environment changes in each turn of the game, the game is often played online with unknown players, and each turn of the game has a fixed time limit.

The NB^3 algorithm that we present in this paper is capable of doing negotiations under realistic scenarios that satisfy the strong criteria mentioned above. Since it is not feasible to calculate the utility values of each possible deal it applies a heuristic search algorithm to determine which possible deals should be evaluated.

This paper is organized as follows: first, in Sect. 2 we give a brief overview of existing work in the field of automated negotiations. In Sect. 3 we state the assumptions made in this paper and define our goals. In Sect. 4 we give a formal definition of the negotiation problems we aim to solve and in Sect. 5 we define the protocol that we apply to the negotiations. Next, in Sect. 6 we define the Negotiating Salesmen Problem (NSP): a specific example of a negotiation problem as defined earlier, which we use for the experiments. Then, in Sect. 7 we introduce our family of negotiation algorithms called NB^3 and in Sect. 8 we describe the negotiation strategy applied by NB^3. Next, in Sect. 9 we describe an implementation of an agent that uses the NB^3 negotiation algorithm to negotiate in the specific scenario of the NSP. In Sect. 10 we describe the experimental results we obtained with this agent. Finally, in Sect. 11 we summarize our conclusions and discuss future work.

2 Related work

Much work has been done on automated negotiations, which can roughly be divided in two categories: the *Game Theoretical Approach* and the *Heuristic Approach*.

The game theoretical approach focuses on the game theoretical properties of negotiation, such as the existence of equilibrium strategies. One of the best known papers in this area is a paper by Nash [31] in which it is shown that under the assumption of certain axioms the outcome of a bilateral negotiation is the solution that maximizes the product of the players' utilities. Many papers have been written afterwards that generalize or adapt some of these assumptions. Multilateral versions of the bargaining problem have been studied for example in [1,23], while a non-linear generalization has been made in [14]. A general overview of such game theoretical studies is made in [42].

This paper falls into the second category, the Heuristic Approach: work that focuses on implementing algorithms that can negotiate under circumstances where no equilibrium results are known, or where the equilibrium cannot be determined in a reasonable amount of time.

Most studies that have been done in this category however involve scenarios with only two agents, a small agreement space and linear additive utility functions that are explicitly given or can be calculated without much computational cost. For example in the first four editions of the annually held Automated Negotiating Agent Competition (ANAC 2010-2013) [6]. Also, most of these studies assume an alternating offers protocol, which is good for automated agents, but not desirable for negotiations with humans, because with humans there is no guarantee that they will indeed follow the protocol.

The combination of search and negotiation has been studied before, for example in [13]. There, the agent has a fixed aspiration level for its utility and searches for the deal that satisfies this aspiration level and is closest (with respect to some similarity measure) to the deal previously proposed by the opponent. This assumes however that there are only two agents involved in the negotiation. Also, their algorithm does not try to model the opponent's preferences and therefore only considers contracts that are close to contracts previously proposed by the opponent. Moreover, in order to find the next best contract to propose, it assumes that the utility function is linearly additive.

Klein et al. also propose a negotiation scenario with search in [21], but time constraints are not taken into account. A more important difference between their approach and ours is that their algorithm applies a mediator that must be trusted and that limits the control that the agents have over the search, since they can only accept or reject proposals made by the mediator, while this mediator does all the searching. In our work we are assuming circumstances where other agents cannot be trusted, so the use of a mediator is not an option. In the same article they also propose a variant of their algorithm without a mediator, involving a mutually observable 'die' to steer the search, instead. But this still means that the agents should trust the fact that the die is fair. Moreover, the agents need to follow a strict protocol, so this algorithm is only suitable for negotiation between agents that were designed for this particular protocol. In [10] it was suggested that one could use genetic algorithms to explore the agreement space. However, neither an implementation nor concrete results were given.

Negotiations with non-linear utility functions have been studied for example in [24]. The negotiations are however bilateral, the agreement space is continuous and it is assumed the agreements at least have a known, closed-form, expression. Also in [21] the utility functions are strictly spoken non-linear over the issues, but they are still linearly additive over *pairs* of issues.

Other research on large agreement spaces with non-linear utility has been done in [19,27,28]. In these cases the non-linearity stems from the fact that some combinations of constraints are incompatible and therefore result in zero utility. Whenever two or more constraints are compatible however, they do assume that the utility is a linearly additive over these constraints, so they treat the problem as a linear additive optimization problem restricted by some constraints. Although in theory any non-linear function can indeed be modeled in this way, in practice it is often not feasible to do so because one needs an explicit expression of the utility function, which one often does not have (e.g. there is no closed-form expression for the utility values of all possible configurations of a chess game). Moreover, the algorithms described in [19,28] again depend largely on a mediator.

This model of non-linear utility functions given as linear combinations of constraints was also adopted in the last edition of the Automated Negotiating Agents Competition (ANAC 2014). We have participated in this competition, but we did not use the NB^3 algorithm, because negotiations were bilateral and the agents in this competition had very little information about

their own utility functions. In order to know the value of any bid they had to request it from an oracle.

Work that comes relatively close to ours is [39], in which non-linear utilities are handled using preference-graphs. They focus however on how to simplify the utility by exploiting knowledge about independence between issues. They assume that utility can indeed be simplified in such a way that the search space is shrunk to a reasonable size and can be explored exhaustively. Moreover, they only consider bilateral negotiations.

Most research on multilateral negotiations that we know of (apart from the game theoretical papers mentioned above) focuses on developing protocols ([9,17]) or on non-selfish negotiations [22]. We do not know of many papers in which multilateral negotiation algorithms for selfish agents are developed, like in this paper.

One case in which such an algorithm was proposed, is [33]. In their study however, a strict separation is made between *buyers* and *sellers*, so a buyer can only come to an agreement with a seller. Our approach is more general, since we do not make this distinction. Indeed, in many real life negotiations one often does not make this distinction either. A retailer, for example, sells its products to consumers, but buys them from a wholesaler, so acts both as buyer and seller. Moreover, they consider the presence of only one buyer, therefore excluding competition between possible buyers, and although multiple sellers are present, they still assume that each agreement is strictly bilateral. Once again utility is linear additive and the alternating offers protocol is assumed.

Also [3] describes multilateral negotiations in which one buyer negotiates with n sellers, but each negotiation thread between the buyer and a seller follows the alternating offers protocol, and they negotiate only about the price of a single item.

As explained, our algorithm applies a Branch & Bound (BB) search algorithm to explore the space of possible agreements. BB has mostly been used as a centralized algorithm. Distributed versions that try and exploit concurrency in the exploration of the tree do also exist [16]. However, not much work has been done on the application of BB algorithms in search problems where the variables are controlled by different agents, as in the asynchronous backtracking method used in Distributed Constraint Satisfaction [48], and where there is not one single function $f(x)$ to optimize but a *set* of functions, one per agent, that are not centrally known.

3 Problem statement

In this section we define the goal of our work, state the assumptions we have made, and motivate the approach we have taken. A formal definition of the problem we aim to tackle is given in Sects. 4 and 5.

3.1 Assumptions

The goal of the research presented in this paper is to design an agent that is able to decrease its cost function, and to do so better than other agents. We have made the following assumptions:

- Negotiations are multilateral.
- Every agent has a finite set of actions it can take to change the current world state (see Sect. 4).
- Each agent has an individual preference relation over world states, defined by a cost function (see Sect. 4).

- Agents are selfish: each agent wants to take those actions that decrease its own cost. The agents have no interest in minimizing other agents' cost functions or reaching a social optimum.
- The definitions of the cost functions are publicly known.
- The cost functions do not have an explicit formula, but are expressed as an NP-hard problem and therefore calculating the cost of a world state or proposal is computationally expensive (see Sect. 6).
- Agents can make binding agreements with each other about the actions each will take. This can improve the efficiency of their actions.
- The number of possible agreements is too large to apply exhaustive search (in Sect. 6.5 we show there can be as many as 20^{200} possible agreements in our experiments).
- The agents negotiate about their plans of action under the Unstructured Communication Protocol (see Sect. 5).
- There is a fixed deadline for the negotiations which is equal for all agents and known to all agents.
- The cost functions do not change over time (e.g. there are no discount factors).
- There is no mediator to help the negotiations.

Furthermore, we have made the following assumptions in this paper, purely to keep the discussion and the notation simple. Our algorithm would work equally well without these assumptions.

- For any agent i a joint plan in which i does not participate does not influence the cost of i.
- The order of execution of actions is irrelevant for the outcome of those actions.

Finally, we mention some important properties we do not take into account, although we think a realistic algorithm should take them into account. We leave them for future work.

- Non-numerical preferences: when negotiating with real people it is often not possible to express preferences as numerical values. Therefore, it would should be better to use preference relations, rather than real-valued cost functions.
- Modeling opponent cost functions: in this paper we assume that an explicit expression for the opponents' cost functions is given. In real-world scenarios one may need to make a model of the opponents' costs.
- Modeling opponent strategy: we do not make any attempt to model the concession strategy of the opponent. Our agent just uses a generic, fixed strategy that does not adapt to the opponents' strategies.
- In this paper we assume agents always fulfill their commitments. In a real negotiation setting there should be some system that enforces agents to fulfill their commitments, otherwise making a commitment has no real meaning. We don't use such a system. Instead we have simply implemented all agents such that they do fulfill their commitments.

3.2 Complete information

Although formally speaking the agents in this model have complete information in the sense that the cost functions are publicly known, we feel it is important to stress that in practice the information they have is far from complete. This is because the agents only know the *definitions* of the cost functions. In order to know the *values* of the cost functions however, they need to perform heavy, time consuming calculations. Given that the domains under consideration are very large, it is absolutely impossible for any agent to know all the cost values

of all possible deals for all agents. Therefore, an agent will usually only make *approximations* of the cost values and can only do so for a *very small subset* of the agreement space.

3.3 Approach

The approach that we take is purely heuristic. We do not try to find any equilibrium strategies because we do not think calculating an equilibrium strategy in the real world is a feasible thing to do. Also, even in the scenario we treat in this paper we cannot think of any way to find formal game theoretical results without simplifying our assumptions so much that they become unrealistic in real-world applications. Let us state some arguments to support this:

- We do not make common assumptions such as the existence of a discount factor, which are often needed to obtain non-trivial results, because we don't think in real negotiations you would ever explicitly have such a discount factor (or know its value).
- Any result that provides hard mathematical guarantees would probably only refer to the test case under consideration (the NSP, see Sect. 6), while our goal is to tackle negotiation problems in general.
- The number of possible deals the agents can make is very large: typically of the order 10^{100} in our experiments, and there are no clear symmetries to reduce this considerably. Analyzing all possible options of a player is impossible.
- Since the players cannot calculate the utilities of all 10^{100} deals, they need to apply a heuristic exploration of the space of possible deals to determine which ones to calculate. This exploration takes place continuously, meaning that the knowledge the agents have about the world changes continuously, and since many solution concepts depend on the knowledge of the agents, such results would also change continuously.
- Even if you find an optimal strategy that tells you to propose a deal with a given target utility, there is no guarantee that you can actually find a deal that indeed yields that utility.
- Since each player explores the space of possible deals independently, each player discovers different possible deals. Therefore, a player does not know which proposals the other players have discovered so far, so there is lack of information about the opponents' options.
- Players do not only accept or propose deals, but also need to decide how long to search for good deals before making a proposal. How would one assign utility to such a decision? Of course one could define some kind of utility function for that, but the results would be dependent on that choice, therefore lose all generality, and therefore not satisfy our goals.

A good example that exemplifies these statements is the game of Diplomacy [8,11,34]. This game has been played by many players worldwide for more than 50 years. If people would have been able to find an optimal negotiation strategy for this game, it would not have been interesting to play it anymore.

4 The agreement space

In this section we give a formal definition of what we call a (multilateral) negotiation problem. The definitions and notation presented here will be used throughout the paper. These definitions hold for agents that aim to *minimize* their cost functions, but they could be changed straightforwardly to deal with maximization of utility functions instead.

Definition 1 A *negotiation problem* is a tuple $\langle A, \hat{\mathcal{O}}, \hat{f}, \mathcal{E}, \epsilon_0, t_{dead} \rangle$ where A is a set of agents $A = \{\alpha, \beta, \ldots\}$, where $\hat{\mathcal{O}}$ is a tuple of sets of actions, one for each agent: $\hat{\mathcal{O}} = (\mathcal{O}_\alpha, \mathcal{O}_\beta, \ldots)$, where $\hat{f}$ is a tuple of cost functions, one for each agent: $\hat{f} = (f_\alpha, f_\beta, \ldots)$, where $\mathcal{E}$ is a set of world states, $\epsilon_0 \in \mathcal{E}$ the initial world state, and $t_{dead} \in \mathbb{R}^+$ the deadline for the negotiations. These concepts are further explained below.

A negotiation problem consists of a number of agents[1] $A = \{\alpha, \beta, \ldots\}$ situated in a world state ϵ_0, which is an element of the set of all possible world states $\mathcal{E}$. Each agent $i \in A$ has a set of actions $\mathcal{O}_i$ to its disposal, and each of these actions can be executed, causing the state of the world to change. So each action ac is in fact a function $ac : \mathcal{E} \to \mathcal{E}$, $ac(\epsilon) = \epsilon'$, where ϵ is the current world state and ϵ' is a new world state. The union of all actions of all agents is denoted as $\mathcal{O} = \bigcup_{i \in A} \mathcal{O}_i$.

Definition 2 A *plan* p is a set of actions: $p \subseteq \mathcal{O}$.

A plan p acts on a world state ϵ by letting all the actions $ac \in p$ act on ϵ (to keep the notation and the discussion simple we assume that the order in which the actions are executed is irrelevant, although this restriction is not necessary for the algorithm to work).

Definition 3 The *Agreement Space* is the set of all possible plans: $2^{\mathcal{O}}$.

Each agent i has a cost function $f_i : \mathcal{E} \to \mathbb{R}$ that induces a preference relation over $\mathcal{E}$. An agent i prefers world state ϵ_1 over ϵ_2 iff $f_i(\epsilon_1) < f_i(\epsilon_2)$.

Some plans may be unfeasible in a certain world state (e.g. you cannot sell a car if you don't own a car). Formally, we do say that such a plan can be executed, but the execution of an unfeasible plan leaves the world state invariant. The set of feasible plans in world state ϵ is denoted as $fea(\epsilon)$.

Definition 4 The set of *feasible* plans in world state ϵ is the set of plans for which ϵ is not a fixed point: $fea(\epsilon) = \{p \in 2^{\mathcal{O}} \mid p(\epsilon) \neq \epsilon\}$. An action ac is feasible in ϵ if and only if the plan $\{ac\}$ is feasible in ϵ.

Definition 5 The set of *participating agents* $pa(p)$ of a plan p is the set of all agents that have at least one action in the plan: $pa(p) = \{i \in A \mid p \cap O_i \neq \emptyset\}$.

In the rest of this paper we will make the simplifying assumption that for any agent i a plan in which i does not participate does not have any influence on the cost of i. That is:

$$i \notin pa(p) \Rightarrow \forall \epsilon \in \mathcal{E} : \; f_i(p(\epsilon)) = f_i(\epsilon)$$

This assumption is not necessary for the NB^3 algorithm to work, but it highly simplifies the discussion and definitions in this paper.

Definition 6 The *reservation value* rv_i for an agent i is the cost it incurs if it does not participate in any plan: $rv_i = f_i(\epsilon_0)$.

The reservation value of an agent represents the maximum cost that that agent would be willing to incur from accepting any plan. In other words: no agent would ever accept any plan for which the cost is higher than the agent's reservation value. After all, if a proposed plan yields a higher cost than the reservation value, the agent would prefer not to participate in any plan at all.

[1] In this paper we use Greek letters to indicate specific *elements* of the set A (i.e. they are the names of the agents), while we use Latin letters as *variables* over the set A. The letter ϵ however is used to refer to world states, so it is not the name of any agent.

5 The unstructured communication protocol

We will now define the protocol that is used by the agents to negotiate. That is: we define the utterances agents can express, when they can express them, and what their formal consequences are. This protocol is entirely new and is called the *Unstructured Communication Protocol*.

Most previous studies have made use of the Alternating-Offers Protocol [40], or something alike. They assume that one agent makes a proposal to another agent, and then this other agent is obliged to either accept the proposal or reject it. If the agent rejects it, it is then its turn to make a proposal, etcetera. Alternative protocols are proposed for example in [3]: it describes multilateral negotiation with 1 buyer and n sellers. The buyer maintains a separate negotiation thread with each seller. Each of these threads however still follows an alternating offers protocol, so the agents are still restricted. In [35] bilateral negotiations are modeled in continuous time, without a strict protocol. They assume that the decision of an agent to make a proposal is determined by external factors, which they model as a random variable. Also [42] describes several alternative protocols for multilateral negotiations.

We consider however none of these protocols satisfactory for two reasons: firstly, because they make a strict distinction between buyers and sellers, which may not always exist in true negotiations (e.g. in the stock market people act both as buyers and as sellers). Secondly, they seem to be designed with the specific goal of making things easier for the designer of the experiments and the agents, while they ignore the fact that in real-world negotiations the players are autonomous and may therefore decide not to follow the protocol. Therefore, we here assume a different protocol, which is less strict.

5.1 Definition of the protocol

The Unstructured Communication Protocol applies to environments with any number of negotiating agents that propose joint plans to each other. A plan can involve any number of agents. When an agent proposes a plan this proposal is sent to all the other agents participating in the plan. Other agents, which do not receive the proposal, do not know anything about it, until the plan is executed (if ever). The agents are committed to the plan once *all agents participating in the plan* have accepted it.

At each moment each agent i can propose any plan, or accept any plan earlier proposed to i by any other agent j. When an agent has proposed or accepted a proposal it is still allowed to withdraw this proposal or acceptance again, as long as it is not committed to the plan yet.

The protocol defines two utterances that the agents can make: 'accept' and 'reject'.

Definition 7 An *utterance* is a tuple of the following form: $(\text{`}accept\text{'}, i, J, p, t)$ or $(\text{`}reject\text{'}, i, J, p, t)$ with $i \in A,\ J \subseteq A,\ p \in 2^{\mathcal{O}}$ and $t \in \mathbb{R}$. The agent i is called the *sender*, the set J is the set of *receivers*, p is the accepted or rejected plan, and $t \in \mathbb{R}$ is the *time stamp*: the time at which the utterance is made.

We do not explicitly define a 'propose' utterance. Instead, an agent proposes a plan by sending an 'accept' message. So we say informally that an agent is proposing a plan if it is the first to express an 'accept' utterance for that plan. The reason for this is that adding a 'propose' message would mean that we would also have to add the rule to the protocol that an agent cannot accept a deal before it has been proposed, while the main idea of the protocol is to enforce as few rules as possible.

Definition 8 A set of utterances C is called a *conversation* if it does not contain two utterances with the same sender and the same time stamp (an agent cannot make two utterances at the same time).

Definition 9 For any given time t we say a plan p is *accepted* in conversation C by agent i if there is an utterance $(\text{'}accept\text{'}, i, J, p, t_1) \in C$ with $t_1 \leq t$ and no utterance $(\text{'}reject\text{'}, i, J, p, t_2) \in C$ with $t_1 < t_2 \leq t$ for any J.

Definition 10 Given the deadline t_{dead}, we say the participating agents of a plan p are *committed* to the plan if there exists a $t < t_{dead}$ at which all agents in $pa(p)$ have accepted it and none of these agents is already committed to another plan p' that makes p unfeasible.

We now stress a number of important properties of this protocol. Note that all of these properties indeed follow implicitly from the fact that the protocol is entirely defined by the four definitions above.

A proposed plan may involve more than two agents This is different from most previous work in automated negotiations as one usually assumes only bilateral deals, even if there are more than two agents negotiating.

A proposal may be sent to any subset J of agents However, if an agent i that participates in the proposed plan is not contained in J it will never receive the proposal and therefore never be able to accept it. Therefore it does not make sense to send a proposal to J if J does not contain $pa(p)$. Nevertheless we do not force the agents to include $pa(p)$ in J, because we leave this responsibility to the agents themselves.

Agents can make more than one deal Negotiations do not stop after a deal has been made, so agents can continue making more deals. However, a new deal cannot be conflicting with any previously made deals (e.g. once you have sold a car, you cannot sell the same car again to another customer).

Agents can change their minds and reject proposals they earlier accepted, as long as they are not committed to it yet When an agent accepts a plan, this is not considered a binding agreement until all other agents participating in the plan have also accepted it. Therefore, an agent can reject an earlier accepted plan to prevent from getting committed to it. Note however that the last definition implies that once an agent is committed to a plan, it stays committed to it, even if it expresses a 'reject' utterance afterwards. A possible extension of the protocol could be to give agents the option to include an expiration time with every 'accept' message, meaning the proposal will automatically be rejected if it has not yet been accepted by all other participating agents after this expiration time. We have however not included this in our current work, to keep things simple.

The agents in this protocol do not take turns An agent can accept or reject any proposal at any time; it does not have to wait for 'its turn'. Moreover, this means that after making a proposal an agent does not have to wait for a counter-proposal, it can already make new proposals even before any agent has replied to the first proposal.

Agents are not obliged to reply to proposals If an agent does not want to accept a received proposal, it may or may not explicitly reject it. The agent may simply ignore the proposal without ever replying. Therefore, when an agent has made a proposal and waits for reply, it should decide for itself how long to wait for this reply. If it takes too long, the agent should consider the proposal as rejected, but it is up to itself when to do so.

When an agent makes a new proposal, it does not have to be compatible with any of the proposals it made before This means that if one of the proposed plans is executed, other proposed plans may become unfeasible. It is up to the agents themselves to determine whether standing proposals are still feasible or not.

5.2 Enforcement of commitments

In any automated negotiation scenario there needs to be some kind of system that enforces the fulfillments of the agents' commitments. This can be either by imposing a punishment to agents that violate their commitments, or by simply making it impossible to violate a commitment. Without such an enforcement system, commitments would not have any formal meaning and rational agents would determine their strategies according to ordinary non-cooperative game theory without agreements.

Enforcement could be taken care of by, for example, an electronic institution [5], but we will not go into any detail on this. For our experiments we have simply implemented our agents to always fulfill their commitments.

5.3 Motivation for the protocol

The reason that we have chosen this unstructured protocol is that we think that it resembles the way people negotiate in the real world. This protocol may be considered inconvenient for designers of agents, but this reflects the problems that negotiators also face in the real world. For example, if you make somebody an offer by e-mail, you have no guarantee that the recipient will ever reply to your mail. If he doesn't reply, you never know for sure whether the receiver is still deliberating over the offer, or is simply ignoring it. An agent implemented for the unstructured communication protocol is therefore much more robust against unexpected human behavior.

Also, the possibility of making several proposals that are mutually incompatible is very common in the real world. Think for example of a real estate vendor that offers a house to several potential customers. Obviously, he cannot sell the same house to all of them, so the customer who reacts first, or bids the highest price, wins. For all other costumers the deal then becomes unfeasible.

6 The Negotiating Salesmen Problem

We will now give an example of a negotiation problem to which our approach applies and which cannot be handled by existing algorithms. It is a new, artificial problem that we have used as a test case for our algorithm. We call this problem the *Negotiating Salesmen Problem* (NSP). It resembles the multiple traveling salesmen problem (mTSP) described in [7], but with the main difference that each agent in the NSP is only interested in minimizing its individual path, while in the mTSP the agents intend to minimize the total length of all the agents' paths together. Therefore, unlike the mTSP, the NSP is a game in which the agents are opponents. However, it is a game in which the agents are allowed to make agreements with each other. After giving the definition we will show that it is a negotiation problem as defined in Sect. 4.

6.1 Definition

The idea is that several agents (the salesmen) need to visit a set of cities. The salesmen all start at the same city (the home city), and all other cities should be visited by at least one agent. Initially, each city is assigned to one salesman that has to visit it. The salesmen are then allowed to exchange some of their cities, so that they are able to decrease the distances they have to cover. For example: if a city v is assigned to agent α, but α prefers to visit another

city v', which is assigned to agent β, then α may propose to β to exchange v for v'. If β however also prefers to have v' over v it will not accept this deal. If no other agent wants to accept v either, then α is obliged to travel along city v. However, we impose the restriction that not all cities are allowed to be exchanged. The cities that can be exchanged are referred to as the *interchangeable cities*, while the cities that cannot be exchanged are called the *fixed cities*. We will now give a formal definition of this problem.

Definition 11 An instance of the NSP is a tuple $\langle G, v_0, A, F, I, \epsilon_0, t_{dead}\rangle$, which consists of: a weighted graph G, a marked vertex v_0 of the graph, a set of agents A, a set of fixed cities F, a set of interchangeable cities I, an initial distribution of cities ϵ_0 and a deadline t_{dead}. These components are further explained below.

G is a finite, complete, weighted, undirected graph: $G = \langle V, w\rangle$ with V the set of vertices (the *cities*) and w the weight-function that assigns a cost to each edge: $w : V \times V \to \mathbb{R}^+$ and that satisfies the triangle inequality:

$$\forall a, b, c \in V : w(a, c) \leq w(a, b) + w(b, c)$$

One of the vertices is marked as the *home city*: $v_0 \in V$. Each agent has to start and end its trajectory in this city. We use the symbol $\overline{V}$ to denote the set of *destinations*, that is: all cities except the home city: $\overline{V} = V \setminus \{v_0\}$. The set of destinations is partitioned into two disjoint subsets: F and I, so: $\overline{V} = F \cup I$ and $F \cap I = \emptyset$. They are referred to as the set of *fixed cities* and the set of *interchangeable cities* respectively.

The set of agents (the *salesmen*) is denoted by $A = \{\alpha, \beta, \ldots\}$. Each destination is initially assigned to an agent, by the function $\epsilon_0 : \overline{V} \to A$. We use the symbol $\overline{V}_i$ to denote the subset of $\overline{V}$ consisting of all cities that are assigned by ϵ_0 to agent i. $\overline{V}_i = \{v \in \overline{V} \mid \epsilon_0(v) = i\}$. $\overline{V}_i$ is referred to as agent i's set of *preassigned* cities. The definitions above imply that for each agent its set of preassigned cities can be further subdivided into: $\overline{V}_i = F_i \cup I_i$ where F_i is defined as $\overline{V}_i \cap F$ and I_i is defined as $\overline{V}_i \cap I$.

Finally, the instance includes a real number t_{dead} that represents the deadline for the negotiations. Agents are allowed to negotiate over the assignment of cities, until this deadline has passed.

Definition 12 An outcome of an instance of the NSP is a map $\epsilon : \overline{V} \to A$ such that the restrictions of ϵ_0 and ϵ to F are equal: $\forall v \in F : \epsilon_0(v) = \epsilon(v)$.

The set of cities assigned to agent i in outcome ϵ is denoted as $\overline{V}_{\epsilon,i}$:

$$\overline{V}_{\epsilon,i} = \{v \in \overline{V} \mid \epsilon(v) = i\}$$

This means that in the outcome the cities are distributed between the agents according to ϵ, but the fixed cities F are still assigned to their original agents. So in the solution, the interchangeable cities are redistributed: $\overline{V} = \overline{V}_{\epsilon,\alpha} \cup \overline{V}_{\epsilon,\beta} \cup \ldots$, while the fixed cities are not: $\overline{V}_{\epsilon,i} \cap F_i = F_i$.

Each agent has a preference over the set of outcomes, defined by a cost function. In order to define this cost function we first need to introduce some more definitions.

Definition 13 Given any finite set $S = \{s_1, s_2, \ldots s_k\}$ of size k, and a permutation π of the integers 1 to k we say a *cycle* $T_{S,\pi}$ through S is an ordered sequence of size k consisting of the elements of S:

$$T_{S,\pi} = (s_{\pi(1)}, s_{\pi(2)}, \ldots s_{\pi(k)})$$

We use the notation $\mathcal{T}_S$ to denote the set of all cycles through S.

Definition 14 If S is a set of nodes from a weighted graph, with the weights denoted by $w(s_i, s_j)$, then the *length* $c(T_{S,\pi})$ of a cycle is defined as:

$$c(T_{S,\pi}) = w(s_{\pi(k)}, s_{\pi(1)}) + \sum_{j=2}^{k} w(s_{\pi(j-1)}, s_{\pi(j)}) \tag{1}$$

With these definitions we can now define the cost function c_i for an agent i over the set of outcomes.

Definition 15 The *cost function* c_i for an agent i is defined as:

$$c_i(\epsilon) = \min_{T \in \mathcal{T}_{\overline{V}_{\epsilon,i} \cup \{v_0\}}} c(T) \tag{2}$$

In words, this means that the cost of an agent i for a given assignment of cities ϵ is defined as the shortest path through the cities assigned to i, including the home city.

6.2 Reasons for rejection

We would like to stress that when an agent α makes a proposal to another agent β that benefits them both, β may still decide to reject the offer, for several reasons. Firstly because β may be planning a counter proposal that reduces his individual cost even more, but that would become impossible after accepting α's proposal. Since both agents explore the agreement space independently they have a different view of their possibilities and bargaining power. Secondly, agent β may also choose to make a deal with another agent γ, which is incompatible with the offer made by α. Thirdly, agent β may simply want to wait and continue searching for better deals.

6.3 The NSP as a negotiation problem

We show now that the NSP is indeed a negotiation problem as defined in Sect. 4.

In the NSP a world state is defined as an assignment of interchangeable cities to agents: $\epsilon : \overline{V} \to A$ (we have intentionally chosen to use the letter ϵ both for world states, and for assignments in the NSP, since an assignment of cities is in fact a world state). An action $ac \in \mathcal{O}$ in the NSP consists of one agent i giving an interchangeable city v to another agent j, so it is a triple (i, v, j), with $i, j \in A, i \neq j$ and $v \in I$. The set of actions $\mathcal{O}_i$ that agent i can choose to execute consists only of actions in which a city is given to i. The fact that i giving a city to j is considered as an action executed by j rather than by i, is because this action decreases the cost of i and therefore we can assume that i will never object to such an action, so j can perform this action autonomously.

$$\mathcal{O} = \{(i, v, j) \in A \times I \times A \mid i \neq j\}$$
$$\mathcal{O}_j = \{(i, v, j) \in A \times I \times \{j\} \mid i \neq j\}$$

The execution of an action alters the state of the world in the following way: if we have an action $ac = (i, v, j)$, and we define $\epsilon' = ac(\epsilon)$, then:

$$\epsilon'(v) = j \quad \text{and} \quad \forall z \in \overline{V} \setminus \{v\} : \epsilon'(z) = \epsilon(z)$$

In words: the city v is re-assigned to agent j, while all other cities remain assigned to the same owner as before. We may need to warn the reader here not to get confused because of the fact that actions and plans are defined as maps from world states to world states, while

in the NSP a world state is itself also defined as a map, namely from the set of cities to the set of agents.

In the NSP the cost function $f_i(\epsilon)$ is the length of the shortest path through all cities assigned to agent i under assignment ϵ.

In the NSP a single action is only feasible if the agent who gives away a city actually owns that city. A plan is considered feasible if each individual action is feasible and if no city is given away twice:

$$p \in fea(\epsilon) \Leftrightarrow \quad \forall(i, v, j) \in p : \epsilon(v) = i \quad \wedge \quad \forall(i, v, j), (i', v', j') \in p : v \neq v'$$

The definition that a plan is only feasible if no city is given away twice, is introduced to discard plans in which one agent gives away the same city twice. At first sight this condition might seem too strict, because it also discards plans in which an agent i gives a city v to an agent j, and then j passes v on to another agent k. However, that would be equivalent to the action in which i gives v directly to k, so it can be modeled as one single, feasible action. Note that it is still possible for a city to be passed on from one agent to another in two separate plans, but we just discard proposals in which our agent proposes such a two-step deal in one single plan.

6.4 The NSP without agreements

As explained in the introduction, the fact that players are allowed to make agreements allows them to reach efficient solutions that would otherwise be unstable. To illustrate this we here show that if we would not allow the agents to make agreements, the NSP would become trivial and the outcome would be inefficient.

We call the non-cooperative variant of the NSP the multiple traveling salesmen problem (nc-mTSP). Just as in the NSP every agent in the nc-mTSP has a set of fixed and interchangeable cities, and all definitions are the same. However, the agents do not communicate and cannot make any binding commitments. To make sure that this game is well defined we furthermore assume it is a turn taking game with a fixed number of rounds, where in each round, one player has the option (but not the obligation) to take one or more interchangeable cities from any other player. For each player the final payoff of the game is the negative of the length of the shortest path through all cities he owns at the end. We now claim that the equilibrium strategy of this game is trivial.

Lemma 1 *In the subgame perfect equilibrium of the nc-mTSP no player ever takes any city from any other player.*

Proof The player whose turn it is in the last round would not want to take any city from any other player, since this could only increase its cost. Knowing this, the player in the second last round would also not want to take any city. By backward induction it follows that the same holds in every round of the game. □

6.5 The NSP as a testbed for real world negotiations

The NSP is not meant as a realistic model for real traveling agents, but rather as a testbed to test algorithms for general, complex, negotiation scenarios. We will show now that a number of properties of real-world negotiations are also present in the NSP.

In many real world negotiations *the utility of a set of issues is non-additive*. That is: the value of a contract depends on the combination of issues. For example: when booking a holiday you need both a plane ticket to your destination and a hotel booking. The

hotel booking is worthless without the plane ticket and vice versa. So the value of having both a plane ticket and a hotel booking is higher than just the sum of their individual values.

This non-additivity also occurs in the NSP. For example: if agent α currently owns cities v_1 and v_2 and there are two cities v_3 and v_4 which are both farther away from the home city than v_1 and v_2. Then α is not interested in exchanging one of its cities for one of the other two cities. However, it could be that v_3 and v_4 lie very close to each other and therefore it would be profitable to exchange *both* v_1 and v_2 for *both* v_3 and v_4.

The fixed cities in the NSP represent the fact that in real negotiations *different agents have different preferences*. Without fixed cities, every agent would have exactly the same utility profile: the path between cities v_1, v_2 and v_3 is equally long for every agent. However, because each agent also has its own fixed cities, every agent would have to traverse a different path even if they would visit the same interchangeable cities. One agent may prefer to visit v_1,v_2 and v_3 because they are close to his fixed city v_4, while another agent may prefer to visit cities v_5,v_6 and v_7, because they are closer to his fixed city v_8. If one wants to model a negotiation scenario in which the preferences of the other agents are unknown, one can impose the restriction that position of any fixed cities is only known to the agent that owns it.

In real-world negotiations it is often *very hard to assign a precise utility value to a deal*. This is captured in the NSP by the fact that for each possible deal the agent has to solve a traveling salesman problem for each agent involved in it. This is very hard, and often it is better to make only a quick approximation rather than to do an exact calculation. Of course, in the NSP the hardness of calculating utility stems from the fact that it is computationally hard, while for many real world problems it is caused by lack of information, but the point is that in both cases the utility can only be approximated.

Finally, we would like to stress that the size of the agreement space in the NSP is very large. If the number of agents is denoted as a, and the number of interchangeable cities per agent by m, then there are in total $a \cdot m$ cities. A proposal in the NSP may assign an agent to every city, so there are a^{am} possible proposals. In the largest experiment we have conducted (see Sect. 10) there were 20 agents and 10 interchangeable cities per agent so there were 20^{200} possible proposals.

7 The NB^3 algorithm

BB is capable of searching through large spaces efficiently and has reasonable solutions available at any time. When searching for the best joint plan to propose, it is usually unnecessary to examine each possible plan. One can often, after examining only a partial joint plan, already discard all full joint plans that contain this partial plan. Furthermore, as the algorithm is running, it yields solutions that get closer to the optimal solution, so at any time it has a solution available that at least approximates the optimum. The fact that BB allows one to discard large parts of the search space and that it is an anytime algorithm makes it ideal to apply to our domain. For an in-depth description of BB algorithms we refer to [25,37].

We propose a family of negotiation algorithms called NB^3 that apply BB to explore the agreement space and discard those parts of it that contain sets of undesirable solutions. We next explain the different components of NB^3 assuming that it runs on an agent called α. The other agents might also run a copy of NB^3, but they might just as well run any other negotiation algorithm, or they could even be human.

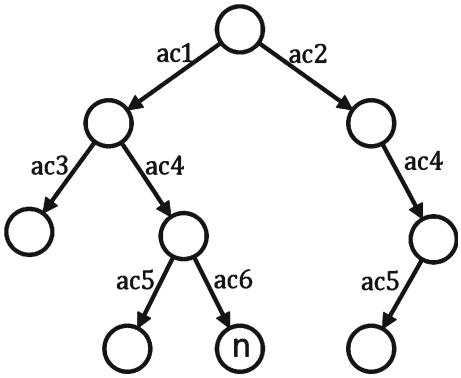

Fig. 1 The search tree. Node n represents the partial plan consisting of the actions $ac1$, $ac4$ and $ac6$

7.1 The search tree

An agent that runs the NB^3 algorithm builds a search tree which is explored according to a best-first strategy. Each arc between a node and its parent is labeled by a certain action from the set of possible actions $\mathcal{O}$. Each node can then be interpreted in four equivalent ways:

- Each node n represents the plan that consists of all the actions that label the arcs in the path from the root to n; this plan is denoted by $path(n)$. The root node corresponds to the empty plan.
- Equivalently, each node represents a set of plans,[2] denoted $plans(n)$, consisting of all plans that can be constructed by adding more actions to $path(n)$. The root node then represents the set of all possible plans and the children of a given node form a partition of the set of plans represented by the parent node. So if a node n has children n_1, n_2, n_3, then $plans(n) = plans(n_1) \cup plans(n_2) \cup plans(n_3)$.
- A third way of interpreting nodes is to see them as world states. The root node then represents the initial world state ϵ_0 and node n represents the world state ϵ_n that results from letting $path(n)$ act on the initial world state: $\epsilon_n = (path(n))(\epsilon_0)$.
- Finally, each node represents the set $\mathcal{E}_n$ of all world states that can be reached by the plans in $plans(n)$:

$$\epsilon \in \mathcal{E}_n \quad \text{iff} \quad \exists p \in plans(n) : p(\epsilon_0) = \epsilon$$

  The root node represents the set of all world states that can be reached by letting any plan act on the initial world state ϵ_0. If a node n has children n_1, n_2, n_3, then $\mathcal{E}_n = \mathcal{E}_{n_1} \cup \mathcal{E}_{n_2} \cup \mathcal{E}_{n_3}$.

To summarize: each node can be identified with a plan $path(n)$, a set of plans $plans(n)$, a world state ϵ_n, and a set of world states $\mathcal{E}_n$. The relationship between these objects is given by:

$$plans(n) = fea(\epsilon_n) = fea((path(n))(\epsilon_0))$$
$$\mathcal{E}_n = \{p(\epsilon_0) \mid p \in plans(n)\}$$

In Fig. 1 the node marked n represents the partial plan consisting of actions $ac1$, $ac4$ and $ac6$, so $path(n) = \{ac1, ac4, ac6\}$. The set $plans(n)$ consists of all feasible plans that include

[2] As mentioned before, to keep the discussion simple we assume that the order in which actions are taken is irrelevant for the outcome of the state of the world, even though our algorithm would work just as well without this restriction. Therefore, we see a plan as a set of actions, rather than a sequence of actions. So a set of plans is a set of sets of actions.

these three actions. The world state ϵ_n is the world state that would result from letting these actions act on the initial world state, and the set of world states $\mathcal{E}_n$ consist of all the world states that can still be realized after these actions have been executed.

7.2 Making decisions

In our environment the commitments among the negotiators have to be made during the search process. This is because an agent cannot wait until it finds the optimal plan before negotiating with other agents, as it might then be too late to get any commitment from them: they might have already signed commitments for incompatible plans. Therefore, a trade-off exists between optimality and commitment availability.

When α receives a proposal from another agent, it has to decide whether to accept it or not, but it may not take this decision immediately. It may prefer to expand the tree a bit more, in order to see if it can find a better alternative to the proposed plan. We see that the more the agent explores the tree before making any commitments, the more likely it is that it will find better plans. But, on the other hand, the less likely it becomes that it will get the other agents to accept those plans. How to solve this trade-off is a key decision when implementing an instance of NB^3.

Another important decision for any implementation of NB^3 is the question of which node to split and how to split it. This may depend on the ongoing negotiation thread. For example, when an agent (say agent β) rejects a plan proposed by α, this means the actions by β should get less priority in future selections to be made by the algorithm. The idea behind this is that if you are under time pressure and there are several negotiation partners, you would be more inclined to negotiate with those partners that are showing more interest in reaching an agreement with you. Otherwise, you would be wasting your time. Moreover, if someone is not willing to concede, you can put him under pressure by suspending negotiations with him and continue your negotiations with others. In Sects. 8 and 9 we show how we have solved these issues for a particular implementation of NB^3.

7.3 Bounding

BB algorithms require that each node n compute upper- and lower bounds for the cost function in the subspace corresponding to this node. In the case of negotiations, however, an agent should not only take its own cost into account but also the cost functions for its negotiation partners, because good solutions can often only be reached in cooperation with other agents, that is: a solution is only reasonable if it also decreases the costs to the other agents sufficiently. For this reason, each node does not only compute bounds for the cost of agent α, but also for every other agent.

The algorithm, running on agent α, is thus assumed to have a model[3] of the cost functions f_i of the other agents, and uses this model to calculate for every node n and every agent $i \in A$ the following bounds (we assume that the goal is to *minimize* a cost function and that the current state of the world is ϵ_0. Furthermore we define $\epsilon_n = (path(n))(\epsilon_0)$):

– For each node n and agent i an upper bound: $ub_i(n)$. This is the maximum cost i may incur from any plan compatible with the plan $path(n)$.

$$ub_i(n) = \max_{p \subset 2^{\mathcal{O}}} \{f_i(p(\epsilon_n)) \mid path(n) \subseteq p\}$$

[3] We will not discuss how it could obtain such a model, because there are many ways to do this and depends on the domain. In the case of NSP this is simple, because it is known that each agent wants to minimize its path.

– For each node n and agent i an intermediate value: $e_i(n)$. The cost agent i incurs if exactly the plan $path(n)$ is executed and no other actions.

$$e_i(n) = f_i(\epsilon_n) = f_i((path(n))(\epsilon_0)).$$

– For each node n and agent i a lower bound: $lb_i(n)$. The minimum cost that i may incur from any plan compatible with the plan $path(n)$.

$$lb_i(n) = \min_{p \subset 2^{\mathcal{O}}} \{f_i(p(\epsilon_n)) \mid path(n) \subseteq p\}$$

Lemma 2 *The upper bound is decreasing, and the lower bound is increasing. That is: for any node n and any child n' of n we have:*

$$ub_i(n) \geq ub_i(n') \quad and \quad lb_i(n) \leq lb_i(n')$$

This implies that the lower bound for agent i of the root node is the lowest cost agent i could ever achieve. Below we indicate the root node with n_0.

Definition 16 The *global lower bound* glb_i of an agent i is the lower bound for agent i in the root node.

$$glb_i = lb_i(n_0)$$

The following lemma follows directly from the definition of the reservation value in Sect. 4 and the definition of the intermediate value.

Lemma 3 *The reservation value of an agent i is equal to the intermediate value of the root node:*

$$rv_i = f_i(\epsilon_0) = e_i(n_0)$$

Proof This follows directly by combining the definition of the intermediate value with the definition of the reservation value given in Sect. 4. □

The bounds defined here, cannot always be calculated exactly, for two reasons. First, because α might not have complete knowledge of the world state and of the other agents' cost functions f_i. And second, because the time restrictions might make it impossible for α to compute these quantities exactly in real time, so α may only be able to estimate them. To be clear, in the rest of this paper we will add a superscript letter α to any quantity if it does not represent the exact value, but only the approximation that α makes of this quantity. So for example $ub_\beta(n)$ indicates the theoretical value of agent β's upper bound of node n, while $ub^\alpha_\beta(n)$ denotes the approximation that α makes about agent β's upper bound.

The intermediate value of a node is the cost that the agent will have to pay if exactly the actions in the path from this node to the root node are executed. So if $e_\alpha(n) \geq rv_\alpha$ the plan $path(n)$ is not profitable for α. Therefore we say a node n is *rational* for agent α iff $e_\alpha(n) < rv_\alpha$ (remember our assumption in Sect. 4 that plans that do not involve agent α do not influence α's costs).

Definition 17 We say agent α believes a node n to be rational for agent i iff $e^\alpha_i(n) < rv^\alpha_i$.

Definition 18 We say α believes a node n is individually rational iff it believes it is rational for all agents participating in $path(n)$.

7.4 Searching and pruning

Since NB^3 performs a best-first search, we need a heuristic h that calculates a value for each node: $h(n) \in \mathbb{R}^+$ to rank the nodes. We call this heuristic the *expansion heuristic*. Each time after splitting a node the algorithm picks the leaf node with the highest expansion heuristic from the tree to be split next. The value of h depends on the values of the bounds defined above. The precise way in which h is calculated from the bounds might depend on the domain, but in Sect. 7.5 we give an example of such a function that is domain independent.

The lower bound is used for pruning: it defines the lowest cost an agent could possibly achieve in any descendant of the node. If $lb_i^\alpha(n) > rv_i^\alpha$ for some agent i participating in $path(n)$, it means that not only this plan is unprofitable for agent i, but also any plan that extends $path(n)$ would be unprofitable for i, so in that case agent i would never agree with any plan descending from node n and therefore this node can be pruned. Of course α only has estimations of the true bounds for the utilities of the other agents, so it is essential that these estimations are good.

Note that for general BB algorithms a node is pruned if its lower bound exceeds a global upper bound, which is defined as the lowest upper bound among all leaf nodes. This is however not the case in NB^3. The reason for this is that, if we look at the node with the lowest upper bound, we cannot be sure that we can actually achieve the commitment of its corresponding plan, since we are never guaranteed that its other participating agents will agree on it. Therefore, we use the reservation value rather than the global upper bound to prune nodes.

One should note that the kind of pruning described here does not have to be done explicitly. After all, the heuristic h determines which node to expand next, so as long as we make sure that $h(n) = 0$ whenever $lb_i^\alpha(n) > rv_i^\alpha$ for some participating agent i, this node will always have lowest priority, which is essentially the same as being pruned.

However, NB^3 also applies another form of pruning which is done explicitly. Whenever agent α gets committed to a plan p, all actions in $\mathcal{O}$ that are incompatible with the actions in p become unfeasible so α can prune all nodes that have any of the incompatible actions in their paths to the root.

7.5 The expansion heuristic

One of the crucial properties of NB^3 is the expansion heuristic h, that determines in which order the nodes are to be split. We will now discuss the default implementation of this heuristic, which is independent of the domain in which the algorithm is used. It can however be improved for specific cases where knowledge about the domain may help guiding the search.

Definition 19 The set of participating agents of a node n is denoted as $pa(n)$ and is defined as the set of participating agents of its corresponding plan: $pa(n) = pa(path(n))$.

Definition 20 The utility of a node n for agent i is the difference between the reservation value and the intermediate value: $u_i(n) = rv_i - e_i(n)$.

The goal of NB^3 is to maximize the utility (i.e. minimize the cost) of agent α. However, α needs the cooperation of other agents in order to execute plans, so it is not enough to look only at the utility of α. It is better to say that the search algorithm aims to find the plan for which the *expectation value* of α's utility is maximized. That is: the plan for which the product of α's utility and the probability that all other participating agents accept it, is maximal.

Definition 21 The node value $V_\alpha^\alpha(n)$ of a node n for agent α is the utility of the node for α times the probability that the corresponding plan gets accepted by all participating agents.

$$V_\alpha^\alpha(n) = u_\alpha^\alpha(n) \cdot \prod_{i \in pa(n) \setminus \{\alpha\}} P^\alpha(acc_i(e_i^\alpha(n))) \tag{3}$$

Here $P^\alpha(acc_i(x))$ stands for the probability, estimated by α, that agent i would accept a plan that yields a cost of x. Of course it is impossible to calculate this probability exactly, but we will see below how it can be estimated.

NB^3 aims to generate nodes with high node value. Now the question is: which node should be expanded in order to generate descendant nodes with high node value? The expansion heuristic of a node n is therefore defined as the highest node value that we expect to find among the descendants n' of n.

$$h(n) = V_\alpha^\alpha(n^*) = max\{V_\alpha^\alpha(n') \mid n' \in desc(n)\} \tag{4}$$

with $n^* = \arg\max_{n' \in desc(n)} V_\alpha^\alpha(n')$ and $desc(n)$ denoting the set of nodes in the subtree under n. Of course, when the algorithm is calculating $h(n)$, the subtree under n has not been generated yet so it cannot directly calculate the value $V_\alpha^\alpha(n^*)$, but with some assumptions it can be estimated, as shown below. For this we need one more definition:

Definition 22 The offer value off_i^α of an agent i is the highest cost that i would incur from all the plans so far proposed or accepted by i.

$$off_i^\alpha = \max\{f_i^\alpha(p(\epsilon_0)) \mid p \in accept_i^\alpha\}$$

Here $accept_i^\alpha$ denotes the set of all plans (known to α) that were proposed or accepted by agent i.

The offer value represents the highest price that agent i has offered to pay, so far. Now we will show how, for any node n, NB^3 estimates the probability $P^\alpha(acc_i(e_i^\alpha(n)))$. It is safe to assume that i would never accept a plan for which its cost $e_i^\alpha(n)$ is higher than its reservation value rv_i^α. Also it is reasonable to assume that this probability decreases as the cost e_i^α for agent i increases. Furthermore, since agent i has already offered to incur a cost of off_i^α one can assume that i would accept any proposal for which i receives a cost lower than, or equal to off_i^α. Therefore, this probability is modeled as a linearly decreasing function from 1 to 0 between the values off_i^α and rv_i^α:

$$P^\alpha(acc_i(x)) = \begin{cases} 1 & \text{if } x \leq off_i^\alpha \\ \frac{rv_i^\alpha - x}{rv_i^\alpha - off_i^\alpha} & \text{if } off_i^\alpha < x < rv_i^\alpha \\ 0 & \text{if } x \geq rv_i^\alpha \end{cases} \tag{5}$$

With this formula, we can calculate the probability that an agent will accept a plan $path(n)$ that yields a cost of $e_i^\alpha(n)$. However, in order to use (4) we need to calculate the probability that n^* will be accepted, while n^* has not been generated yet, and therefore we cannot know the values $e_i^\alpha(n^*)$. Therefore, for n^* we estimate the probability of acceptance as follows (to simplify notation from now on we denote $e_i^\alpha(n^*)$ as e^*):

$$P^\alpha(acc_i(e^*)) = \int_0^\infty P^\alpha(acc_i(x)) \cdot P^\alpha(e^* = x)dx \tag{6}$$

In order to calculate this, the algorithm then needs to estimate a probability distribution $P^\alpha(e^* = x)$. From the definition of the lower bound it follows that e^* can never be lower

than the lower bound $lb_i^\alpha(n)$ of n. Furthermore, the algorithm assumes that the value e^* will not be higher than $e_i^\alpha(n)$ (so it makes the optimistic assumption that each node n always has some descendant node with lower intermediate value). Therefore, the probability distribution $P(e^* = x)$, is modeled as a uniform distribution between $e_i^\alpha(n)$ and $lb_i^\alpha(n)$. We can then rewrite (6) (we further simplify notation by denoting $e_i^\alpha(n)$ as e and $lb_i^\alpha(n)$ as lb) as:

$$P^\alpha(acc_i(e^*)) = \frac{1}{e - lb} \int_{lb}^{e} P^\alpha(acc_i(x))dx \tag{7}$$

Finally, we need to estimate the value of $u_\alpha^\alpha(n^*)$. We know that n^* is by definition in the subtree under n, which means that the plan $path(n)$ is a subplan of the plan $path(n^*)$, so we can assume that $u_\alpha^\alpha(n^*)$ is not very different from $u_\alpha^\alpha(n)$. Therefore, we make the simplifying assumption that $u_\alpha^\alpha(n^*) = u_\alpha^\alpha(n)$. Now we can calculate the expansion heuristic by combining (3) and (4):

$$h(n) = u_\alpha^\alpha(n) \cdot \prod_{i \in pa(n) \setminus \{\alpha\}} P^\alpha(acc_i(e^*)) \tag{8}$$

which can be calculated explicitly by combining it with (5) and (7).

One very important remark we would like to state here, is that every time an agent makes a proposal or accepts a proposal its offer value can change, which means the expansion heuristic of every node in the tree changes. If an agent concedes, its offer value increases, and therefore the expansion heuristic of every node in which this agent is participating increases. In other words: if β concedes, it becomes more attractive for α to explore plans in which β is participating. We see thus that not only is the negotiation influenced by the search, but also the other way around: *the search is influenced by the offers that are made by the other agents. Search and negotiation have become intimately intertwined with each other*. This is a unique property of NB^3.

7.6 Modeling preferences of other agents

As explained above, the NB^3 algorithm requires a model of the cost functions of the other agents. We do not see this as a limitation because we believe that knowledge of your opponents' preferences is essential in almost all negotiation scenarios. If a negotiator does not know anything about the preferences of its opponent, it is almost impossible to make any sensible proposal.

Agents may base their opponent models on prior knowledge of the domain and the opponents, on the proposals made by the other agents during the negotiations, on arguments exchanged between the agents, or on any other form of information provided by the other agents (see for example [43]).

In the case of NSP modeling the opponents' cost functions is easy, because we know that each agent is only interested in making its own path as short as possible and the positions of all cities are known. In many other problems it may be much more difficult to know the preferences of the other agents. Especially since agents might hide their preferences or lie about them.

In this paper we have intentionally chosen to use a domain in which opponent modeling is trivial, because we want to focus on the other aspects of the negotiation algorithm. We consider modeling the preferences of the opponents a domain specific matter and therefore we will not discuss how to do this for domains other than NSP. Moreover, the problem of modeling opponents has already been studied many times before, for example in [18]

and [47]. In order to use NB^3 in any realistic domain it should be augmented with such an opponent modeling algorithm to obtain the bounds of the tree nodes.

8 Negotiation strategy

In this section we explain the negotiation strategy applied by NB^3 for negotiations under the unstructured communication protocol. Although NB^3 applies a model of the opponents' utility functions, it does not apply any model of the opponents' negotiation strategies. We leave that as future work.

8.1 Proposing and accepting

As the search tree is expanding, some nodes that are being generated represent individually rational plans. The agent needs to determine which of them to propose to the other agents.

The question when to accept an offer and what to propose has been solved theoretically, under specific assumptions such as the presence of a discount factor that decreases the utility as time passes and the assumption that the players follow the alternating offers protocol, in [41]. For more general settings such as ours however, there is no known optimal strategy.

In many previous studies it is assumed the negotiators have strictly opposing interests. For example: a car salesman aims to sell the car for the highest possible price, while the client aims to buy it for the lowest possible price. In these cases, both negotiators usually start by proposing a selfish plan, but, as time passes, concede towards each other, reaching some intermediate solution. The amount of utility that an agent asks from its opponent at a certain time is called the 'aspiration level' [12]. The intuitive idea behind such strategies is that you try to guess how much the other is willing to concede and try to hide how much you are willing to concede. You don't want to concede too much, but if you concede too little your utility may decrease (in case there is a discount factor) or your opponent may prefer to continue negotiating with other players (in case of multilateral negotiations). Moreover, in the specific setting of this paper, the players need to search for good proposals at run time which makes it even more important not to concede too quickly, as you may find better options as time is passing.

The assumption of strictly opposing interests implies that 'concession' can be defined in two equivalent ways: either as 'demanding less utility from the opponent', or as 'offering more utility to the opponent'. In our situation this is no longer true. In our scenario the search for possible agreements takes place simultaneously with the negotiations meaning that new solutions are being found during the negotiations that may dominate earlier found solutions. Therefore agents sometimes propose new offers that increase their own utilities as well as their opponents'. Moreover, in most existing work when an agent receives a proposal it only has two options: to reject it or to accept it. In our situation however, there is a third option: to continue searching for better solutions.

These differences motivate us to define a new negotiation strategy, in which the agent takes into account not one, but two aspiration levels. We will explain this in the rest of this section. In order to keep the explanation simple we first present this negotiation strategy for the case of bilateral negotiations and then generalize it to the multilateral case.

8.1.1 Bilateral negotiation strategy

During the negotiations agent α regularly needs to make a choice between proposing an offer, accepting an offer, or continue searching. The agent bases its decision on three values: the

time t passed since the start of the negotiations, the normalized utility $\bar{u}_\alpha^\alpha$ it receives from a possible proposal and the normalized utility $\bar{u}_\beta^\alpha$ the opponent β receives from a possible proposal.

Definition 23 The normalized utility of a plan p for agent i, in world state ϵ_0, is defined as the utility divided by the maximum utility it could possibly achieve: $\bar{u}_i(p) = \frac{rv_i - f_i(p(\epsilon_0))}{rv_i - glb_i}$.

To make a decision, agent α compares these utility values with two values, denoted as $m_\alpha^\alpha(t)$ (the self-aspiration-level) and $m_\beta^\alpha(t)$ (the opponent-aspiration-level) respectively, which are fixed, time dependent functions. Note that although m_β^α represents an aspiration level for the utility of β, this value exists in the mind of α. It is the amount of utility that α believes to be necessary to offer to β.

Definition 24 A plan p is more selfish than plan p' iff $\bar{u}_\alpha^\alpha(p) > \bar{u}_\alpha^\alpha(p')$. For a given time instant t we say p is selfish enough iff $\bar{u}_\alpha^\alpha(p) > m_\alpha^\alpha(t)$. Given a set of plans P, the plan $p \in P$ that maximizes $\bar{u}_\alpha^\alpha(p)$ is called the most selfish plan of P.

Definition 25 A plan p is more altruistic than plan p' iff $\bar{u}_\beta^\alpha(p) > \bar{u}_\beta^\alpha(p')$. For a given time instant t we say p is altruistic enough if $\bar{u}_\beta^\alpha(p) > m_\beta^\alpha(t)$. Given a set of plans P, the plan $p \in P$ that maximizes $\bar{u}_\beta^\alpha(p)$ is called the most altruistic plan of P.

Notice that 'selfish' and 'altruistic' as defined here are not necessarily each other's opposites. If plan p yields more utility than p', for both negotiators, p is more selfish *and* more altruistic than plan p'.

At given moments t separated by time intervals of fixed length, α decides what to do: to propose a new plan, to accept a previously proposed plan, or to continue searching for better plans (the length of these intervals is a parameter of the algorithm). This decision is taken according to the following 6-step strategy:

1. First α determines the set X of all plans it has found so far and believes to be individually rational.
2. Then, it determines the subset $Y \subset X$ of all plans in X that are altruistic enough.
3. If Y is not empty, α picks the plan $p \in Y$ that is most selfish. If on the other hand Y is empty, then α picks the plan $p \in X$ that is most altruistic.
4. Next, α determines the set Z of all plans that have been proposed to it by other agents. From this set it picks the most selfish plan $p' \in Z$.
5. From the two plans p and p', it then picks the one which is most selfish.
6. Finally, α checks whether the plan chosen in the previous step is selfish enough. If yes, then this plan will be proposed or accepted. If however this plan is not selfish enough, the agent will continue to search for better plans.

To summarize this: α will only accept or propose any plan that is individually rational and selfish enough. It will only propose a new plan p if there is no standing proposal p' proposed to α that is more selfish than p (because then it prefers to accept p'). And from all candidate plans it could propose, it prefers the most selfish plan that is altruistic enough. If however no plan is altruistic enough, it prefers the plan that is most altruistic (also see Algorithm 5 in Sect. 9.5 for a description of this procedure in pseudo-code.)

Since α should start selfish, and concede as time passes, $m_\alpha^\alpha(t)$ is a decreasing function, so that less selfish plans are proposed as time advances, and $m_\beta^\alpha(t)$ is an increasing function, so that more altruistic plans are proposed as time advances.

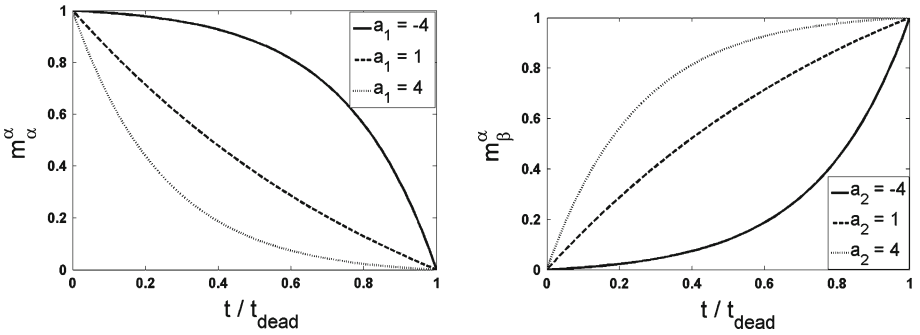

Fig. 2 The graphs of m^{α}_{α} and m^{α}_{β} for several values of a_1 and a_2

Notice that in order to be proposed or accepted, step 6 requires that the plan is selfish enough, while, because of step 3, it does not need to be altruistic enough. This is because if α has a plan that is probably not good enough for β to be accepted, it doesn't harm much to try and propose it anyway. On the other hand, if α would propose a plan that yields very little utility for itself, and it gets accepted by β, then α is committed to this deal, which might make other, more profitable deals in the future impossible.

For the aspiration levels we have chosen the following expressions:

$$m^{\alpha}_{\alpha}(t) = 1 - \frac{e^{-a_1 \frac{t}{t_{dead}}} - 1}{e^{-a_1} - 1} \tag{9}$$

$$m^{\alpha}_{\beta}(t) = \frac{e^{-a_2 \frac{t}{t_{dead}}} - 1}{e^{-a_2} - 1} \tag{10}$$

Their graphs are plotted in Fig. 2. Notice that m^{α}_{α} decreases from 1 to 0 and m^{α}_{β} increases from 0 to 1. The higher the values of a_1 and a_2, the faster the agent concedes. Therefore a_1 and a_2 are called the *concession degrees*. The strategy of the agent can be adapted by adjusting these two parameters.

The fact that m^{α}_{α} and m^{α}_{β} go to 1 and 0 respectively makes this strategy a very weak one for bilateral negotiations, since it can be easily countered by any opponent. The opponent β would simply not concede, but wait until m^{α}_{β} is so high that α will propose a plan that is highly favorable to β. However, one should keep in mind that this strategy is developed for multilateral negotiations. In the multilateral case, agent β does not have the opportunity to wait until α makes a highly altruistic offer, since β has competition from other agents. If β does not concede, α might reach deals with some of the other agents, leaving β with nothing.

The algorithm intends to find plans for which the opponent-utility is as close as possible to the opponent-aspiration-level. The self-aspiration level imposes an extra criterion, that determines whether the plan is selfish enough to be proposed, or whether it is better to continue searching instead. This solves the trade-off problem discussed in Sect. 7.2.

Finally, note that this strategy never rejects any proposal. Instead, it simply ignores bad proposals. The advantage of this is that our agent never has to worry about the question when to definitely reject a proposal and that it always keeps every proposal as an option to accept in the future. On the other hand, rejecting proposals would have the advantage that our agent could inform the other agents about its preferences which could improve the proposals made by them.

8.1.2 Comparison with single aspiration level

To further justify why we are using *two* aspiration levels, we will now take a look at what would happen if one of the two aspiration-levels would be set to zero, so that there is effectively only one aspiration-level.

In general it can happen that the plan p chosen in step 3 is very unprofitable for agent α, even though it's the most selfish one. But step 6 then makes sure that such a plan is not proposed because it is not selfish enough. Now if m_α^α would be zero however, every plan would be considered selfish enough so even bad plans would be proposed. In simple negotiations, where the entire set of solutions is known, this would not be a problem because p would simply be the most selfish plan existing, so there would not exist any better solution for α.

However, in our situation, because the search for good solutions runs simultaneously with the negotiations, it is only sure that plan p is the most selfish solution *found so far*. It would therefore be better to continue searching than to propose the bad plan. This is exactly the reason why we have m_α^α: it determines whether the plan is selfish enough to propose, or if it is better to continue searching for a better plan before proposing it. The more time available, the better it is to continue searching. If however there is very little time left before the deadline, there is little hope of finding a better plan. Therefore, the selfishness-criterion should become weaker as the deadline approaches, so m_α^α must be a decreasing function of time.

Now instead suppose that m_β^α is always zero. Each plan would be considered altruistic enough and the agent would only have the following two options: propose the most selfish plan (if it is selfish enough) or continue searching until it finds a plan that is selfish enough. But in this way α might never concede, because there might exist a lot of very selfish plans. An agent that does not concede at all is generally not a good negotiator, especially in a multilateral environment where your opponents have the possibility to ignore you and come to agreements with each other.

8.1.3 Characterization of strategies

Our concession strategy is parametric in the two concession degrees. We will now discuss how the various settings of these parameters would affect negotiations. We define four strategies by setting the values of a_1 and a_2 either high or low.

A: low a_1, low a_2. Greedy agent. Only proposes very selfish plans. If it hasn't found any plans that are selfish enough, it prefers to continue searching for them than to concede.

B: high a_1, low a_2. Lazy agent. Proposes very selfish plans, but if it can't find any, it will propose less selfish plans, rather than to search for better solutions.

C: low a_1, high a_2. Picky agent. This agent is willing to propose altruistic plans, but only if they are also selfish, otherwise it prefers to continue searching. So it keeps searching until it finds a plan that is both very selfish and very altruistic.

D: high a_1, high a_2. Desperate agent. Concedes fast, even if it costs him a lot of utility.

Roughly we can say that the higher the value of a_1, the less the agent likes to search. The higher the value of a_2, the more altruistic the plans are that the agent proposes (or is willing to accept). Strategy *A* should only be played if the agent has little competition. If the agent knows it has a stronger position than its opponents it can use this strategy to exploit them. Strategy *D* on the contrary, should only be played in a highly competitive environment. If there is a lot of competition it is better to try to come to an agreement as soon as possible, before the competition takes away all the good deals.

B and *C* are more moderate strategies. *B* should be played if good plans are scarce. In such an environment it is not likely to find many plans that are better than your current options, so

it is better to give up some utility than to continue searching for a better plan. If good plans are abundant, it is better to play C. In that case, if your current options are not good enough, instead of giving up utility it is better to keep searching a bit more because it is likely that you will find some better plan.

From basic experimentation we have concluded that good values of the concession degrees for the NSP are $a_1 = 2$ and $a_2 = 4$ and have fixed these values for our further experiments in Sect. 10. We leave a more thorough experimentation to determine the best values for these parameters as future work.

8.1.4 Multilateral negotiations

Things become more complicated when we want to apply our strategy to multilateral negotiations. In order to make the agent capable of multilateral negotiation, we have chosen a simplified model in which the agent treats the set of all opponents as if it were one opponent, and then follows the same concession strategy as above. It defines the opponent-utility of a plan as the product of all the normalized utilities of the other agents participating in the plan. When choosing whether to propose, accept, or wait, it applies exactly the same procedure as in the bilateral case, only the quantity $\bar{u}^{\alpha}_{\beta}$ is replaced by $\bar{u}^{\alpha}_{pa}$ with:

$$\bar{u}^{\alpha}_{pa}(p) = \prod_{i \in pa(p) \setminus \{\alpha\}} \bar{u}^{\alpha}_{i}.$$

with the extra restriction that $\bar{u}^{\alpha}_{pa}(p)$ is zero if $\bar{u}^{\alpha}_{i}$ is negative for any $i \in pa(p)$.

This multilateral concession strategy does not take into account which agents are involved in the proposals α makes. So when one proposal p_1, made by α, is not accepted, α will try to make a new proposal p_2 that gives more utility to its participating agents than p_1 did, even though the agents participating in p_2 might be completely different from the ones in p_1. The idea behind this is that α considers all agents as equivalent. After all, we assume that the agents are unknown, so we cannot distinguish between them. In the rest of this paper we will denote the opponent aspiration-level for multilateral negotiations as m^{α}_{pa} instead of m^{α}_{β}.

A possible way of improving the multilateral strategy, which we leave for future work, would be to store information about the opponents obtained during the negotiations. We could then make a profile of each opponent and use this to set a different opponent-aspiration-level for each individual opponent. Furthermore we could try alternative definitions of $\bar{u}^{\alpha}_{pa}$, such as the minimum of the opponent-utilities. Again, we leave this as future work.

9 Branesal: NB^3 applied to the NSP

Up until now the description of NB^3 has been as general as possible. In this section however, we describe how we have implemented NB^3 to be applied to the NSP defined in Sect. 6. We call this implementation *BRAnch and bound NEgotiating Salesmen ALgorithm (Branesal)*.

9.1 Calculating the bounds

We will now explain how the bounds of the search tree are calculated in the case of the NSP. To calculate these bounds the agent needs to know the shortest path that goes through a given set of cities. It is however much too costly to calculate this length exactly. Therefore, we consider an estimation of the shortest path by calculating the *greedy path* instead.

Definition 26 The greedy path through a set of cities is calculated as the path that goes from the home city v_0 to its nearest neighbor v_1, then from v_1 to v_1's nearest neighbor v_2, etc. until we have visited all cities and come back to v_0.

So for any world state ϵ its cost for agent i, $f_i(\epsilon)$, is estimated by agent α as the length of the greedy path through the set of cities assigned to i in the world state ϵ. This estimated value is denoted by $f_i^\alpha(\epsilon)$.

This greedy heuristic may not be very accurate, but from experiments it turns out to work quite well in practice. We have tried to use more accurate heuristics, but the extra computations this involved were so costly that it was not worth using them.

In the NSP an 'action' consists of one agent giving one city to another agent. Each arc between two tree nodes is labeled by such an action, and the path from a node back to the root represents the partial plan consisting of all the actions that form the labels along the path. One can check that the definitions below are consistent with the general definitions of the bounds as given in Sect. 7.3. The calculations of the intermediate values and the lower bounds are also given in pseudo-code in Algorithm 6.

Upper bound

Although the specification of the NB^3 algorithm defines an upper bound for every node and every agent, the Branesal implementation does not calculate it. The reason for this is that the expansion heuristic defined in Sect. 7.5 does not use it. Other implementations of NB^3 may however use another implementation of the expansion heuristic.

Intermediate value

In order to calculate the intermediate value of node n, we take the set of cities currently assigned to i, remove all the cities that are given away by i in any of the actions in $path(n)$, and add all the cities that are acquired by i in any of the actions in $path(n)$. We then calculate the greedy path through this new set.

Lower bound

The lower bound $lb_i^\alpha(n)$ is the minimum path length that i could possibly achieve in any plan descending from node n. The most optimistic scenario is the case in which i is able to give away all its interchangeable cities, except those that it has acquired in $path(n)$. The idea is that once a city v is acquired from some agent j in $path(n)$ it will not be given away again to some other agent k in any plan that descends from n (in fact we even declared these kinds of plans unfeasible in Sect. 7). This is because our agent would not consider such a plan because it would already consider an equivalent plan, in which v is given directly from j to k, somewhere else in the tree.

The lower bound of a node n for an agent i is therefore calculated as the length of the greedy path through the home city, the fixed cities owned by i and the cities given to i in any of the actions in $path(n)$.

9.2 Splitting

Besides the bounding and the expansion heuristic, a very important design-issue for any BB algorithm is the question of how to split the nodes. Since a plan is build up from actions,

it would be natural to split a node according to the several alternative actions that can be performed. This would mean adding a new node for each action that is compatible with $path(n)$. However, in many cases this would result in a huge amount of child nodes, making the algorithm very inefficient (if there are m interchangeable cities per agent and a agents, then the total number of actions is in the order of $m \cdot a^2$). Therefore we have chosen an implementation in which an 'action' is split up in smaller components.

An action in the NSP consists of three components: an agent that gives away one of its cities (the donor) the city that is being given away, and the agent that receives the city (the acquirer). So instead of adding a child node for each possible action, we first add a child for each possible donor, then pick the best of these children (i.e. the child node with the highest expansion heuristic) and expand it by adding a child for each of the cities that the donor can give away. Finally, we pick the best of these nodes and add to it a set of children, each one of which corresponds to one of the possible acquirers. So each arc between two nodes is not labeled by an action but by a component of an action, and a path of three consecutive nodes corresponds to one action. In this way, the maximum number of children that could be needed to be generated in each cycle is reduced to only $m + 2a$ (only a of these nodes however correspond to a new plan).

Since the precise implementation is not relevant for the rest of the paper, we will not give a more detailed description of this procedure and therefore continue as if these three steps were taken in one step. That is: as if each arc between a parent and a child node is labeled with an action, rather than only one component of an action.

9.3 Handling proposals

When a proposal from another agent is received, α adds a new branch to the tree to represent the proposed plan. The bounds and expansion heuristic are then calculated for every new node in this branch, and all these new nodes are put into the open list. In this way α can explore extensions or adaptations to the received proposal.

If a proposal is accepted by all participating agents, the corresponding plan is immediately executed, which alters the state of the world (we have implemented a system that sends a message to all agents whenever a plan is executed, to let all agents know that the world state has changed). Therefore, α has to update its internal representation of the world. Many of the branches in the search tree will then become unfeasible, since they represent plans that are incompatible with the new world state, so these branches are removed from the tree.

9.4 Data structures

In order to describe the Branesal algorithm we first describe its most important data structures: *WorldState*, *Action*, *Message*, *Node* and *Tree*. These data structures will then be used in the next section to describe the implementation of Branesal in pseudo code.

We assume there is a set of names of cities given, as well as a set of names of agents:

$$\begin{array}{l} City = \{v_0, v_1, v_2, ...\} \\ Agent = \{\alpha, \beta, \gamma, ...\} \end{array}$$

The program contains one *WorldState* object, that represents the current assignment of the cities of the NSP instance. There is a home city, a set of fixed cities and a set of interchangeable cities. The *assign* function represents the assignment of the fixed cities and interchangeable cities to the agents.

WorldState

$homeCity : City$
$fixedCities : 2^{City}$
$intCities : 2^{City}$
$assign : fixedCities \cup intCities \rightarrow Agent$

The components of the WorldState structure satisfy the following constraints:

$$\forall i \in Agent : |\{v \in fixedCities \mid assign(v) = i\}| = 1$$
$$fixedCities \cap intCities = \emptyset$$
$$homeCity \notin fixedCities \cup intCities$$
$$homeCity \cup fixedCities \cup intCities = City$$

The first of these constraints says that each agent has exactly one fixed city assigned to it. The other three constraints together state that the set of fixed cities, the set of interchangeable cities and the home city together form a partition of the set of all cities.

An *Action* object represents the action of one agent giving one city to one other agent.The agent that gives the city is called the 'donor' and the agent that receives the city is called the 'acquirer'.

Action

$donor : Agent$
$city : City$
$acquirer : Agent$

Agents send messages to each other to propose, accept or reject plans. Note that in Sect. 5 we mentioned that the protocol does not formally include a 'propose' message because a proposal is simply the first 'accept' message in the conversation regarding to a certain plan. It turns out however that the algorithm is easier to implement if it internally does make a distinction between a proposal and an acceptance. Therefore, whenever the algorithm receives an accept message for a new plan, the algorithm internally treats it as a 'propose' message, and whenever the algorithm proposes a new plan, the communication layer of the agent converts it to an 'accept' message to comply with the protocol.

Message

$type : \{propose, accept, reject\}$
$sender : Agent$
$receivers : 2^{Agent}$
$plan : 2^{Action}$

The tree nodes of the search tree are implemented as the *Node* data structure. Each node contains a reference to its parent node, and is labeled by an *Action*. The set of all labels of all the ancestors of the node, including the node itself, is what we call the *path* of the node and the set of all donors and acquirers of all the actions in the path forms the set of *participating agents* (*pa*). Furthermore, each node contains an intermediate value, a lower bound and an expansion heuristic for each agent. From these values one can calculate the normalized utility $\overline{u}^{\alpha}_{\alpha}$ and the opponent-utility $\overline{u}^{\alpha}_{pa}$. Finally, the node contains the set of participating agents that still need to accept the plan (as we will see later, it is initialized as the set of participating agents *pa* of the node, and each time one of these agents accepts the plan, this agent is removed from the set).

Node

$parent : Node$
$label : Action$
$pa : 2^{Agent}$
$e^{\alpha} : Agent \rightarrow \mathbb{R}^{+}$
$lb^{\alpha} : Agent \rightarrow \mathbb{R}^{+}$
$h : \mathbb{R}$
$\bar{u}^{\alpha} : Agent \rightarrow \mathbb{R}$
$\bar{u}^{\alpha}_{pa} : \mathbb{R}$
$haveNotAcceptedYet : 2^{Agent}$

Tree represents the search tree, which maintains a root node, an open list, and for every agent a reservation value and a global lower bound.

Tree

$root : Node$
$rv^{\alpha} : Agent \rightarrow \mathbb{R}^{+}$
$glb^{\alpha} : Agent \rightarrow \mathbb{R}^{+}$
$openList : 2^{Node}$

9.5 Procedures

We now describe the Branesal algorithm itself, which is given in Algorithm 1. We see that after initialization, it consists of a while loop that repeatedly calls three functions: *expand*, *handleIncomingMessages* and *acceptOrPropose*, which are described below. The algorithm keeps looping until the deadline for the negotiations has passed.

Algorithm 1 Branesal

Require: startTime, timePassed, expandInterval, lastAcceptOrProposeCall, $t_{dead} : \mathbb{R}$
Require: ϵ : WorldState
Require: theTree : Tree
Require: foundByMe $= \emptyset$
Require: proposedToMe $= \emptyset$
Require: $m^{\alpha}_{\alpha} : [0, t_{dead}] \rightarrow [0, 1]$
Require: $m^{\alpha}_{pa} : [0, t_{dead}] \rightarrow [0, 1]$
Require: $off^{\alpha} : Agent \rightarrow \mathbb{R}^{+}$
1: initializeTree(ϵ, theTree)
2: startTime $\leftarrow$ getCurrentTime()
3: timePassed $\leftarrow 0$
4: lastAcceptOrProposeCall $\leftarrow 0$
5:
6: **while** timePassed $< t_{dead}$ **do**
7: expand(ϵ, theTree, foundByMe)
8: handleIncomingMessages(theTree, off^{α})
9: **if** timePassed - lastAcceptOrProposeCall > expandInterval **then**
10: acceptOrPropose(foundByMe, proposedToMe, $m^{\alpha}_{\alpha}, m^{\alpha}_{pa}, timePassed$)
11: lastAcceptOrProposeCall $\leftarrow$ timePassed
12: **end if**
13: timePassed $\leftarrow$ getCurrentTime() - startTime
14: **end while**

The *expand* method starts by extracting the node with the highest expansion heuristic from the open list and determining for which actions we should add child nodes to it (we do not provide here the implementation of the function that determines what actions to split

Algorithm 2 initializeTree(ϵ, theTree)

Require: home, fixed, interchangeable, current, minimal : 2^{City}
1: home $\leftarrow$ {ϵ.homeCity}
2: **for all** $i \in$ Agent **do**
3: fixed $\leftarrow$ $\{v \in \epsilon.\text{fixedCities} | \epsilon.assign(v) = i\}$
4: interchangeable $\leftarrow$ $\{v \in \epsilon.\text{intCities} | \epsilon.assign(v) = i\}$
5:
6: current $\leftarrow$ home $\cup$ fixed $\cup$ interchangeable
7: minimal $\leftarrow$ home $\cup$ fixed
8:
9: theTree.root.e_i^α $\leftarrow$greedyPath(current)
10: theTree.root.lb_i^α $\leftarrow$greedyPath(minimal)
11:
12: theTree.$rv_i^\alpha \leftarrow root.e_i^\alpha$
13: theTree.$glb_i^\alpha \leftarrow root.lb_i^\alpha$
14:
15: theTree.root.$\bar{u}_i^\alpha \leftarrow 0$
16: **end for**
17: theTree.root.$\bar{u}_{pa}^\alpha \leftarrow 0$
18: theTree.openList $\leftarrow$ {theTree.root}

over, but works as explained in Sect. 9.2). For each action to split over, we create a new node, label it with the given action, set its participating agents, calculate its bounds (see Algorithm 6 for details on that), calculate its expansion heuristic (in the way explained in Sect. 7.5), add the new node as a child of the original node and add the new node to the open list. Finally, if the new node is individually rational (i.e. for each participating agent the intermediate value is lower than the reservation value) we can add it to the list of plans that are candidates to be proposed. *HandleIncomingMessages* checks whether a message has been received from any of the other agents. If not, the method returns. Otherwise, if the incoming message is a proposal, then a new node n' is created that corresponds to the proposed plan and is added to the tree. The agent does not decide how to reply to this proposal yet, because this is done later by the *acceptOrPropose()* function (Algorithm 5). If the incoming message is an acceptance of a plan, the agent retrieves the plan from the tree. Note that this plan can indeed be found in the tree, because the agent had stored the plan in the tree when the plan was first proposed.

In either case, the agent checks whether the proposing or accepting agent is offering to pay a higher cost than it has offered before, and if that is indeed the case the offer value of that agent is adapted. This means the expansion heuristic of each node in the open list needs to be recalculated.

Finally, if the plan in the incoming message is accepted by all participating agents, the execute method is called (Algorithm 7), which updates the world state and resets the root of the tree. The *acceptOrPropose* method determines whether α itself should accept or propose a plan, according to the procedure described in Sect. 8.1. The *calculateBounds* function (Algorithm 6) calculates the intermediate values and the lower bounds of each agent. Also, it uses the reservation values and the global lower bounds to calculate the normalized utility of each agent. Finally it calculates the opponent utility by taking the product of all the normalized utilities of the other agents participating in the node's plan.

9.6 Complexity

We will now discuss the amount of time and memory that is needed for each new plan generated by the search algorithm.

Algorithm 3 expand(ϵ, theTree, foundByMe)

Require: n : Node
Require: splitActions : 2^{Action}
1: //Get the node with the highest expansion heuristic and remove it from the open list:
2: $n \leftarrow \arg\max_m \{m.h \mid m \in openList\}$
3: theTree.openList $\leftarrow$ theTree.openList $\setminus \{n\}$
4:
5: //Get the set of actions to split over.
6: splitActions $\leftarrow$ chooseSplitActions(n)
7:
8: //For each such action: create a new node, calculate its properties and add it to the tree.
9: **for all** action $\in$ splitActions **do**
10: n' $\leftarrow$ new Node
11: n'.label $\leftarrow$ action
12: n'.pa $\leftarrow$ n.pa $\cup$ {action.acquirer}
13: n'.haveNotAcceptedYet $\leftarrow$ n'.pa
14: calculateBounds(ϵ, n', theTree)
15: $n'.h \leftarrow$ calculateExpansionHeuristic(n')
16: theTree.openList $\leftarrow$ theTree.openList $\cup \{n'\}$
17: n'.parent $\leftarrow$ n
18:
19: //If the node is individually rational, then add it to the list of proposals we might want to propose.
20: **if** $\forall i \in n'.\text{pa} : rv_i^{\alpha} > n'.e_i^{\alpha}$ **then**
21: foundByMe $\leftarrow$ foundByMe $\cup \{n'\}$
22: **end if**
23: **end for**

9.6.1 Time complexity

The most time consuming part of the algorithm is the calculation of the bounds and the expansion heuristic each time a new node is added. The time complexity of calculating the expansion heuristic is proportional to the number of agents $O(a)$ (for each participating agent we have to calculate the value of $P^{\alpha}(ac_i(e^*))$).

Calculating the bounds of a node for one agent is quadratic in the number of cities that that agent owns. This is because finding the greedy path involves finding the nearest neighbor for each city owned by a certain agent. The bounds have to be calculated for each agent, so if there are m interchangeable cities per agent, calculating the bounds has a time-cost of $O(am^2)$. Generating a new node therefore has a time complexity of $O(a + am^2)$.

Each time a node is split we need to generate $m + 2a$ new children (as explained in Sect. 9.2). Splitting a node therefore has a time cost in the order of $(m + 2a) \cdot (a + am^2)$, that is: $O(a^2m^2 + am^3)$. Each time that such a split is made, there are a plans generated, so we can say that the amount of time needed to explore one possible plan is $O(am^2 + m^3)$.

Another point regarding the time complexity that we would like to stress, is that each time the agent receives a proposal (or the acceptance of an earlier made proposal), the offer value of the agent that made the proposal (or accepted the proposal) must be adapted, which means that for every node in the open list the expansion heuristic needs to be recalculated. Moreover, the open list has to be reordered (it is implemented as a priority queue so the node with highest expansion heuristic can be retrieved fast).

In order to recalculate the expansion heuristic of a node, we only need to update the value $P^{\alpha}(ac_i(e^*))$ of the agent for which the offer value has changed. Therefore, this can be done in constant time and thus the recalculation of the expansion heuristics of all of the nodes is done in $O(k)$ time (with k the size of the open list).

Algorithm 4 handleIncomingMessages(theTree, off^{α})

```
Require: msg : Message
Require: n' : Node
Require: i : Agent
msg ← getMessageFromMessageQueue()
i ← msg.sender

if msg.type = propose then
  n' ← insertProposedPlanIntoTree(theTree.root, msg.plan)
  proposedToMe ← proposedToMe ∪{n'}

  //If the proposer offers to pay a higher price than he it offered before,
  //update its offer value and re-calculate the expansion heuristic for every node in the
   //tree.
  if off_i^α < n'.e_i^α then
    off_i^α ← n'.e_i^α
    for all n ∈ theTree.openList do
      n.h ← calculateExpansionHeuristic(n)
    end for
  end if
end if

if msg.type = accept then
  n' ← getNodeCorrespondingToPlan(msg.plan)

  //If the accepting agent accepts a price to pay higher than it has offered before,
  //update its offer value and re-calculate the expansion heuristic for every node in the
  //tree.
  if off_i^α < n'.e_i^α then
    off_i^α ← n'.e_i^α
    for all n ∈ theTree.openList do
      n.h ← calculateExpansionHeuristic(n)
    end for
  end if

  //The sender of the message has accepted the proposed plan,
  //so we can remove it from the list of agents that have not accepted it yet.
  //If all agents have accepted the plan, it can be executed.
  n'.haveNotAcceptedYet ← n'.haveNotAcceptedYet \ {msg.sender}
  if n'.haveNotAcceptedYet = ∅ then
    execute(n')
  end if
end if
```

Reordering the open list can be done in $O(k\ log(k))$ time. Although k can be a very large number, it turns out from experiments that the time spent on updating the open list is in practice negligible. This is because the number of times that a proposal or acceptance from another agent is received is very small compared to the number of times that a node is expanded.

9.6.2 Space complexity

Regarding the space complexity it is important to note that in each node we need to store the bounds for each agent, so the memory needed for each node is $O(a)$. Therefore, each time we expand a node, the extra memory we need is $(m+2a)\cdot a$, that is: $O(a^2+am)$. And, since

Algorithm 5 acceptOrPropose(foundByMe, proposedToMe, m^{α}_{α}, m^{α}_{pa}, $timePassed$)

Require: myAspiration, opponentAspiration : $\mathbb{R}$
Require: bestFound : Node
Require: bestProposed : Node
myAspiration $\leftarrow m^{\alpha}_{\alpha}(timePassed)$
opponentAspiration $\leftarrow m^{\alpha}_{pa}(timePassed)$

bestFound $\leftarrow$ getMostSelfish(foundByMe)

//Get the most selfish node that is altruistic enough
repeat
  bestProposed $\leftarrow$ getMostSelfish(proposedToMe)
until bestProposed.$\bar{u}^{\alpha}_{pa}$ > opponentAspiration OR proposedToMe = $\emptyset$

//If no node is altruistic enough, then get the most altruistic one instead
if bestProposed.$\bar{u}^{\alpha}_{pa} \leq$ opponentAspiration **then**
  bestProposed $\leftarrow$ getMostAltruistic(proposedToMe)
end if

//If the best plan found by us (or proposed to us) is selfish enough, then propose it (or accept it).
if bestFound.$\bar{u}^{\alpha}_{\alpha}$ >bestProposed.$\bar{u}^{\alpha}_{\alpha}$ **then**
  if bestFound.$\bar{u}^{\alpha}_{\alpha}$ > myAspiration **then**
    propose(bestFound)
    foundByMe $\leftarrow$ foundByMe \{bestFound}
  end if
else
  if bestProposed.$\bar{u}^{\alpha}_{\alpha}$ > myAspiration **then**
    accept(bestProposed)
    proposedToMe $\leftarrow$ proposedToMe \{bestProposed}

    //If we are accepting a plan that all others already have accepted,
    //then the plan will be executed.
    if bestProposed.haveNotAcceptedYet = {α} **then**
      execute(bestProposed)
    end if
  end if
end if

Algorithm 6 calculateBounds(ϵ, n, theTree)

Require: home, fixed, interchangeable, acquired, donated, current, minimal : 2^{City}
Require: path : 2^{Action}
path $\leftarrow$ getPath(n)
home $\leftarrow$ {ϵ.homeCity}
for all $i \in n$.pa **do**
  fixed $\leftarrow$ {$v \in \epsilon$.fixedCities|$\epsilon.assign(v) = i$}
  interchangeable $\leftarrow$ {$v \in \epsilon$.intCities|$\epsilon.assign(v) = i$}
  acquired $\leftarrow$ {ac.city | ac $\in$ path, ac.acquirer = i}
  donated $\leftarrow$ {ac.city | ac $\in$ path, ac.donor = i}

  current $\leftarrow$ home $\cup$ fixed $\cup$ acquired $\cup$ interchangeable $\setminus$ donated
  minimal $\leftarrow$ home $\cup$ fixed $\cup$ acquired

  $n.e^{\alpha}_{i} \leftarrow$greedyPath(current)
  $n.lb^{\alpha}_{i} \leftarrow$greedyPath(minimal)

  $n.\bar{u}^{\alpha}_{i} \leftarrow (theTree.rv^{\alpha}_{i} - n.e^{\alpha}_{i})/(theTree.rv^{\alpha}_{i} - theTree.glb^{\alpha}_{i})$
end for
$n.\bar{u}^{\alpha}_{pa} \leftarrow \prod_{i \in pa \setminus \{\alpha\}} \bar{u}^{\alpha}_{i}$

Algorithm 7 execute(n, theTree)

Require: path : 2^{Action}
1: //Get the set of actions that form the path from the root to n
2: path $\leftarrow$ getPath(n)
3:
4: //Update the world state by letting the actions in $path$ act on it.
5: //That is: change the assignment of the cities to the agents
6: **for all** $ac \in n$.path **do**
7: $\quad \epsilon$.assign $\cup\{v \mapsto ac.acquirer\} \setminus \{v \mapsto ac.donor\}$
8: **end for**
9:
10: //Node n becomes the new root node
11: //and the rv and glb are set equal to the bounds of this new root.
12: theTree.root $\leftarrow n$
13: theTree.$rv^{\alpha} \leftarrow n.e^{\alpha}$
14: theTree.$glb^{\alpha} \leftarrow n.lb^{\alpha}$

expanding a node yields a new plans, the average amount of memory needed for generating a single new plan is $O(a + m)$.

10 Experiments and results

We have conducted a number of experiments with Branesal and in this section we present their results. Before we present these results however, we will first discuss in Sect. 10.1 why we cannot compare our algorithm with existing algorithms, and in Sect. 10.2 we describe how we have set up the experiments.

10.1 Comparing with other algorithms

In order to test our algorithm, we would like to see how it performs against other algorithms. That is: to have an NB^3 agent engage in negotiations with agents running other negotiation algorithms, and see if our agent scores better than the others. Unfortunately, we do not know of any existing algorithm that can be applied to our domain. Existing algorithms only work for bilateral negotiations, require a mediator, or assume that all possible offers and their utility values are known beforehand. Testing NB^3 against such algorithms would mean applying it to a domain it was not developed for. Experimental results from such a domain would be meaningless.

The claim we make in this paper is that we are the first to successfully combine search and negotiation in a scenario as complex as ours. We do not claim that our implementation of BB is better than other kinds of search. It may very well be that we could obtain better results if we would combine our negotiation strategy with, for example, genetic algorithms, simulated annealing, taboo search, or other forms of tree search. We do plan to test these alternatives, but we leave that as future work.

Furthermore, we would like to stress that it does not make sense to compare NB^3 with a pure search algorithm, because one always needs to implement a negotiation strategy that determines which solutions to propose, and when to accept which proposals made by the other agents. Simply *finding* a good solution has no meaning in the context of automated negotiations, since it needs to be *accepted* by the participating agents as well. The final outcome of a negotiation therefore depends highly on the negotiation strategies applied by the agents.

What we can test however, is how the algorithm scales with increasing complexity of the problem instances. For that purpose we conducted a number of experiments in which all agents were running NB^3, and compared the results for different numbers of agents, different numbers of cities, and different deadlines. Also, we have tested how the algorithm performs when negotiating against a simplified version of itself that applies random search. Furthermore, we have compared the solutions found by the algorithm with a certain notion of optimality.

10.2 Experimental setup

For our experiments we have made use of two types of NSP instances that differ in the way they are generated: *random instances* and *simple instances*. For the random instances all cities are represented as points in the two-dimensional plane, with the home city located at the coordinates $(0, 0)$. The x and y coordinates of all other cities are integers randomly chosen from a uniform distribution over the interval $[-100, 100]$. After generating the coordinates of the cities, the cities are randomly divided among the agents, such that every agent owns the same amount of cities. Also, for each agent, one of its cities is randomly chosen to be its fixed city. All other cities (except the home city) are interchangeable. The distance between two cities is given by the Euclidean distance. In all experiments except those in Sect. 10.7 we have used the random instances. The generation of simple instances is explained there.

For each run of the experiments we store for each agent the coordinates of the cities it initially owns and the coordinates of the cities it owns after the negotiations. When the run has finished we find the shortest path through each of these sets of cities, by feeding them into the *Concorde TSP Solver* [4].

We denote the length of the shortest path through the initial set of cities owned by agent i as C_i^{in}, and we denote the length of the shortest path through the final set of cities owned by agent i as C_i^{fin}. With this notation we then define our performance measure for the random instances as the percentual cost reduction averaged over all agents:

$$Q = \frac{100}{|A|} \sum_{i \in A} \frac{C_i^{in} - C_i^{fin}}{C_i^{in}} \tag{11}$$

This is the result of one run. The results presented in this section are all averaged over 100 runs, each with the same parameters, but with a different instance of the NSP. We should stress however, that the agents do not try to optimize this value, but rather each agent tries to minimize its individual path length. Therefore, *we are not so much interested in the value of Q, but rather in how it changes as the problem instances get more complex.*

Note that in the literature on bilateral negotiations one often uses the product of the agents' utility gains (the Nash Product [31]) rather than the sum to define a measure of performance. The problem with this is that when we are dealing with multilateral negotiations, the set of agents participating in a deal is often a subset of all the agents involved in the negotiations. Therefore it can happen that one agent does not decrease its cost at all, while all other agents do manage to obtain low costs. If we would then take the product of all utility gains, the result would be zero, because of the single agent that did not succeed in its negotiations.

For each data point we have also calculated the *standard error*, as $\frac{\sigma_n}{\sqrt{n}}$ with $n = 100$, where σ_n is the standard deviation of Q over n runs. We will not give the standard error of every data point in this paper, but, in order to indicate the accuracy of the experiments, we will mention for each experiment the highest and lowest standard errors among the data points.

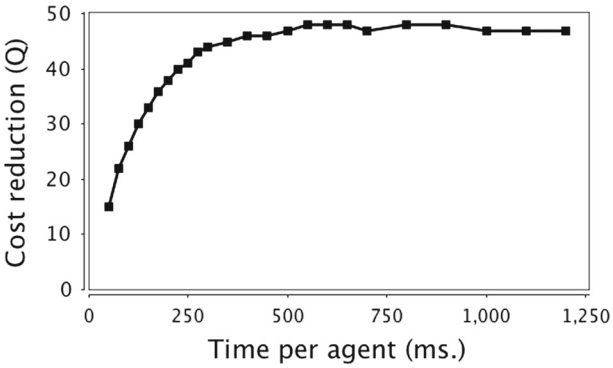

Fig. 3 Cost reduction as a function of time

All experiments were conducted on an iMac with 3.4 GHz Intel Core i7 processor and 8 GB of memory. The agents were implemented in Java on top of the Jade [20] platform.

10.3 Varying negotiation length

In order to determine how the results improve with longer negotiations, we have done a number of tests, each with the negotiation length set to a different value. Each of these tests involved 10 agents, all running the Branesal algorithm, with 10 interchangeable cities per agent.

Because in some of the following experiments we vary the number of agents, we always measure the length of negotiations in milliseconds per agent. So if the negotiation length is 500 ms per agent and there are 10 agents, the deadline of the negotiations is set to 5 seconds. We should remark however that, since all agents are running on the same machine, we had no control over the amount of CPU cycles assigned to each agent, since this is the responsibility of the Java Virtual Machine and the operating system. Therefore, we can only be sure that the amount of time that an agent has to run the algorithm is 500 ms *on average*.

The results are presented in Fig. 3. We see that *the costs of the agents decrease significantly* and that the results get better as the available time increases. After all, the more time the agents have, the more good plans they will find, and therefore the better the final agreements they will make. The highest value (48 %) is reached with 550 ms per agent. It seems this value does not improve with more time. The standard errors of these data points lie between 0.38 and 0.69.

10.4 Varying the number of agents

To determine how the algorithm scales with the number of agents, we have performed a number of tests with each a different number of agents, all running the Branesal algorithm. In each test 10 interchangeable cities were assigned to each agent and the negotiation length was set to 250 ms per agent. According to the results presented in the previous section, 250 ms is not enough for the agents to reach their maximum score. We have chosen this value however in order to put the agents under pressure, increasing the contrast between the various tests.

The results are presented in the left graph of Fig. 4. Interestingly, it seems from this graph that at first, *the results get better as the number of agents increases, even though the problem becomes more complex*. Apparently, the increased computing power resulting from the larger number of agents and the fact that agents can profit from the plans discovered by other agents

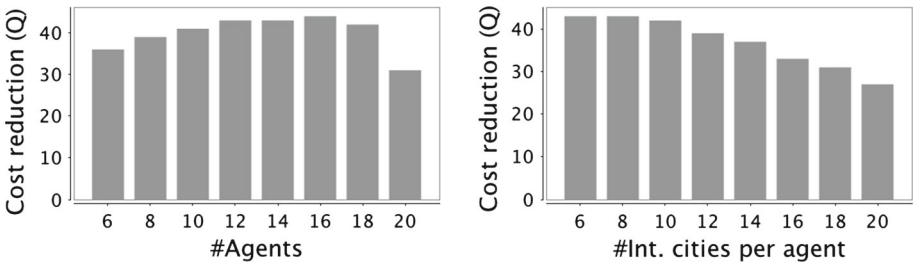

Fig. 4 Cost reduction as a function of the number of agents and of the number of cities

outweighs the increased complexity of the problem. Unfortunately this only remains true as long as the number of agents is less than or equal to 16. With more than 16 agents we see that the complexity of the problem becomes more important and the results start to decrease. It is still unclear to us why this turning point takes place at 16 agents. The standard errors of these data points lie between 0.35 and 0.98.

10.5 Varying the number of cities per agent

We now look at what happens if we make the agreement space larger (more interchangeable cities per agent), while keeping the number of agents constant. In each test there were 10 agents, all running the Branesal algortithm, with a negotiation length of 250 ms per agent. The results are presented in the right graph of Fig. 4.

As expected, with an increasing number of cities, the results decrease, but we think this decrease is relatively small, as the number of cities more than triples, while the value of Q only drops from 43 to 27. This can be explained by the fact that the expansion heuristic successfully manages to steer the search such that only interesting plans are explored and the unprofitable plans are skipped. In this way the increased size of the problem hardly decreases the efficiency of the algorithm. Therefore, we can conclude from this that *the expansion heuristic successfully manages to limit the number of redundant nodes that are explored*, as it is supposed to do. The standard errors of these data points lie between 0.41 and 1.07.

10.6 Comparing with random search

In the previous sections all agents have been running the Branesal algorithm. However, the most important question is how it performs when negotiating with agents that run different algorithms. Since there exists however no comparable negotiation algorithm, we have tested it instead against a copy of itself that applies random search.

We let some agents running Branesal (the "smart agents") negotiate with a number of agents running a random search (the "dumb agents"). With "random search" we mean that the agent is running an algorithm that is identical to Branesal, except that the expansion heuristic for each node is replaced with a random number.

We did four tests. Each test involved 10 agents, but for each test the number of dumb agents among those 10 was different. The negotiation length was set to 250 ms per agent. The results are presented in Fig. 5. For each test we show the average score of the dumb agents (the left graph), the average score of the smart agents (center), and the score averaged over all agents together (right). When we compare the left graph with the middle graph, we can clearly see that, as expected, *the smart agents score significantly better than the dumb agents*. In other words: the expansion heuristic is effective. It is also interesting to see that if there

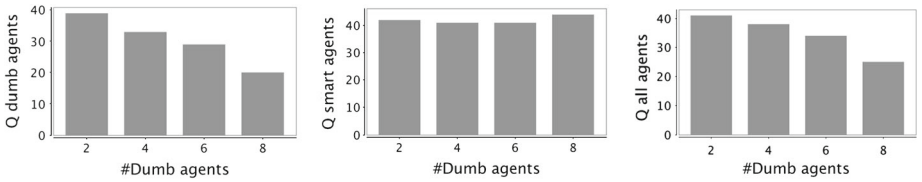

Fig. 5 Cost reduction of dumb agents (*left*), smart agents (*center*), and all agents (*right*), as a function of the number of dumb agents

are only a few dumb agents the dumb agents still manage to decrease their cost considerably. This can be explained by the fact that, although the plans they discover are bad, they still get offered good proposals from the smart agents. Since the plans found by the smart agents are generally better than the ones found by the dumb agents, the smart agents have more bargaining power, and are thus able to exploit the dumb agents. The standard errors of the results of the dumb agents lie between 0.54 and 1.05, the standard errors of the results of the smart agents lie between 0.47 and 1.30, and the standard errors of the overall results lie between 0.43 and 0.59.

10.7 Comparing with the optimal solution

In this section we compare the results of our algorithm with the optimal solution. Note however, that the notion of 'optimal solution' can be difficult to define in automated negotiations. A common way of defining optimality in games is to use some equilibrium concept such as the Nash Equilibrium [32]. The problem however is that a game often needs very specific properties in order to be able to calculate such an equilibrium (e.g. the presence of a discount factor in bargaining games). Moreover, even if one is able to play a strategy to reach the equilibrium solution, this would only be optimal under the assumption that the opponent also plays that strategy. A negotiator that manages to exploit suboptimal play of its opponents would be even better. Another definition of 'optimal solution' would be the outcome in which your agent achieves the minimum possible cost. However, this definition is unpractical since it is highly unlikely that the opponents of the agent would accept such a solution. For example, when two agents negotiate on how to divide a pie between them, the optimal outcome for agent α would be that α gets all of the pie and β gets nothing. Of course, β would never agree with such a deal, so this definition of optimality is unrealistic. A third way of defining the optimal solution would be to define it as the solution that minimizes the social cost, that is: sum of the costs of all agents. The problem with this however is that the agents are simply not interested in reaching the social optimum. Every agent would prefer to try to obtain another, more selfish, solution.

So generally speaking, there is no such thing as an 'optimal solution' in automated negotiations. This is a fact that actually occurs in many games. Take for example robot-soccer. When one develops a robot-soccer team one can only compare it to other teams and see which one is best, but there is no way to compare the team with any kind of theoretically optimal soccer team.

Nevertheless, we have come up with a solution that allows us to define a kind of optimal solution in special cases. We have created a set of special instances of the NSP that have the nice property that there exists one specific solution that is clearly the most reasonable one, because any other solution for which one agent decreases its costs would imply a strong increase in the cost of another agent and would therefore be unrealistic. The idea is that the cities are distributed in clusters around the fixed cities of the agents. The optimal solution is

reached whenever each agent has exactly those cities in the cluster around its fixed city. We call these instances *simple instances*.

The cities of these simple instances are again given as 2-dimensional coordinates and their distances are the Euclidean distances. The graphs are however generated in two stages: in the first stage we create a random cities (with a the number of agents: $a = |A|$), far away from each other. Each of these cities is assigned to one of the agents as its fixed city (each agent gets exactly one fixed city). For each such fixed city we then randomly generate m cities nearby that city. In this way we have created a clusters of $m + 1$ cities each. So after this first stage all the cities of one cluster are assigned to the same agent. We refer to this assignment as the *'optimal assignment'*. This assignment is optimal in the sense that every agent owns a set of cities that lie very close to each other so the agents cannot decrease their path length any further by negotiation.

Then, in the second stage, for each agent we 'swap' some of its cities with cities from other clusters. A swap means that we randomly pick one city assigned to the agent, and one city assigned to an other agent and interchange them. For each agent we make $m/3$ swaps, so after swapping each agent owns at least one city from another cluster, and on average for each agent two thirds of its interchangeable cities are in another cluster (we make $m/3$ swaps for each agent, and each swap involves 2 cities, so in total $2/3 \cdot m \cdot a$ cities change owner). We refer to this new assignment as the *'initial assignment'*, because this is the assignment of the cities at the start of the negotiations.

The length of the shortest path through the set of cities that are assigned to agent i in the optimal assignment, is denoted by C_i^*. The score of a test is calculated as follows:

$$Q_{simple} = \frac{100}{|A|} \sum_{i \in A} \frac{C_i^{in} - C_i^{fin}}{C_i^{in} - C_i^*} \tag{12}$$

We have only used NSP instances for which $C_i^{in} - C_i^* \geq 10$ for each agent. With these instances we have repeated the experiments of Sect. 10.3 with four different values of m, namely: 6, 9, 12, and 15.

We see that *the algorithm is able to reach a score of 80 % of the optimal solution*. Furthermore, we note that as the number of cities increases, the algorithm converges more slowly, but still manages to reach 80 %. The standard errors of the data points in these four graphs lie between 0.72 and 1.70 (Fig. 6).

We expect that non-selfish negotiating agents could reach a higher score than this. Also, a (distributed) constraint optimization algorithm would probably be more successful in decreasing the social cost. However, it is important to note that NB^3 is designed to optimize *individual* costs, rather than social cost, so one cannot compare this result with results from non-selfish scenarios (one could make NB^3 a non-selfish algorithm by defining the individual cost functions to be equal to the social cost, but it would then still be less efficient then other non-selfish algorithms, because the agents could be searching through overlapping regions of the agreement space).

Also one should note that the fact that a score of 100 % is not *reached*, does not mean that the optimal solution has not been *found* by any of the agents. Even if an agent finds the solution that minimizes social cost it may still try to propose other, more selfish, solutions. It might for example happen that agents α and β come to a deal that yields low cost to both agents, but that is incompatible with the socially optimal solution, especially if the resulting individual costs for α and β are lower than what they would get in the socially optimal solution. And even if a deal is individually worse than the social optimum, agents still might prefer to come to a quick individually suboptimal solution rather than wait and hope they can

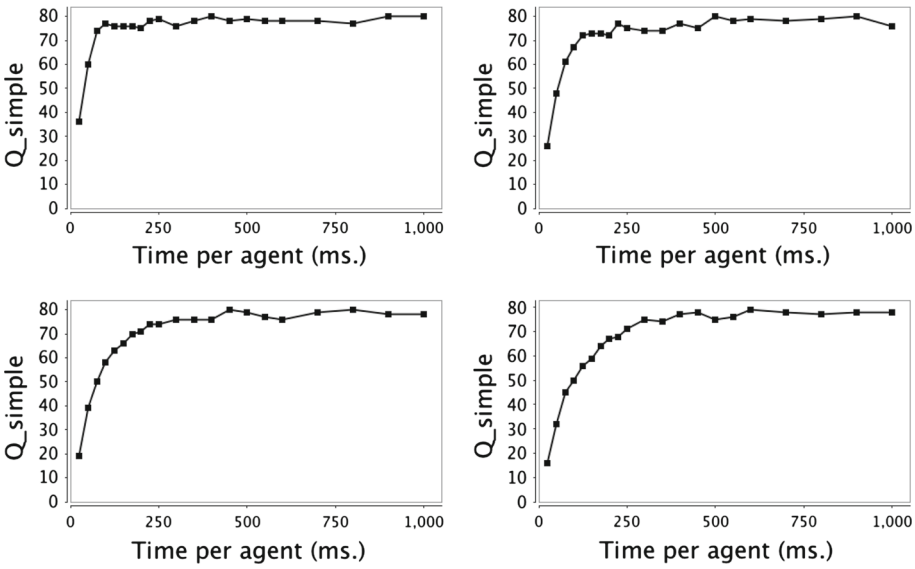

Fig. 6 Increasing negotiation length, with simple NSP instances. *Top left* 6 interchangeable cities per agent, *top right* 9 interchangeable cities per agent, *bottom left* 12 interchangeable cities per agent, *bottom right* 15 interchangeable cities per agent

find a better solution, since the available time for search is limited and agents fear missing good deals because of competition.

One alternative way of solving the NSP might be to use clustering rather than heuristic search to find good solutions. We have intentionally not tried to do this, because clustering would *only* be applicable to NSP and not to general negotiation problems. Our goal was not to find the best solution to the NSP, but to use NSP as a testbed for general negotiation algorithms.

Yet another way of solving the NSP would be to apply a centralized approach in which a mediator finds a solution that benefits all agents and is the most fair solution according to some fairness criterion. However, once again, this is not the goal of our work. In real-world situations it is not always possible to find an impartial mediator, to have all agents to agree on the definition of fairness, or to make agents cooperate in finding social solutions.

11 Conclusions and future work

In this paper we have introduced a new family of negotiation algorithms for very large and complex agreement spaces, with multiple selfish agents, non-linear utility functions and a limited amount of time. This family is called NB^3 and applies best-first Branch and Bound to search for good proposals.

Our main motivation for doing so is to bring automated negotiations closer to real-world negotiations. Therefore, we have had to discard a number of assumptions that are usually made in existing literature, as we consider them unrealistic.

One of those assumptions is the application of the Alternating Offers protocol. In order to discard the alternating offers protocol we have introduced a new protocol for multilateral negotiations that assumes as few restrictions as possible. We call this the Unstructured Communication Protocol.

Moreover, we have introduced a new test bed for multilateral, non-linear negotiations called the NSP. We introduced this problem because in most existing work the effort necessary to determine the value of a deal is ignored. It is often assumed that given a deal, its corresponding utility can be calculated quickly, which we consider unrealistic. In the NSP on the other hand, the calculation of the utility (or cost) of a given deal is computationally expensive.

We have defined a general purpose heuristic to guide the Branch & Bound search of the NB^3 algorithm. Furthermore we have defined a new negotiation strategy that does not use a single aspiration level for the utility, as in most existing work, but that uses two aspiration levels because it considers the utility aspired by our agent and the utility to be conceded to the opponents as two separate quantities. This allows our agent to not only determine what to propose, but also to determine whether it should make a proposal or rather continue searching for better proposals.

We have implemented an instance of NB^3 for the NSP that we call Branesal and we have performed several experiments with it, with extremely large search spaces (of the order 10^{100}). From these experiments we draw the following conclusions:

- Our agent indeed manages to decrease its costs significantly by negotiation.
- Most of this decrease is obtained within half a second.
- If we increase the complexity of the problem by increasing the number of agents, the results remain stable, up to 18 agents.
- If we increase the complexity of the problem by increasing the number of cities, the results only get slightly worse.
- The heuristic search applied by our algorithm successfully manages to prune redundant nodes.
- The heuristic search applied by our algorithm is significantly better than random search.
- For problem instances that have a clear optimal solution, the algorithm manages to reach a solution which is at 80 % of the optimal solution.

For the future, we are planning to fine-tune some of the components of NB^3 such as the expansion heuristic and the negotiation strategy. We plan to experiment with different values of the concession degrees, for example, and with different initial and final values of the aspiration levels, which currently only take the values 0 and 1. Moreover, as explained in Sect. 8, in the current implementation the agent treats the set of opponents as if it were one opponent to which it should concede. We will improve the negotiation strategy by dropping this assumption, so that our agent can treat every single other agent as a different opponent. Furthermore, we have mentioned in Sect. 8.1.1 that there is a parameter that determines how long the agent continues expanding the search tree before deciding whether to make a new proposal or not (the expandInterval parameter in Algorithm 1). We will investigate the influence of the value of this parameter on the results.

In the NSP, the preferences of the agents are expressed explicitly as utility values. We think that for practical applications it is unrealistic to assume that user preferences can be expressed explicitly as numerical utility functions. Therefore, we will adapt the algorithm so that it can handle qualitative preference relations instead. Furthermore, the current implementation assumes there is a straightforward way to estimate the opponents' utility functions. To make this more difficult we plan to develop an implementation of NB^3 for the Diplomacy game, which has complex rules that make it much more difficult to determine the utility of a deal. Moreover, Diplomacy has an extremely large search space.

Finally, we should implement other negotiation algorithms by combining our negotiation strategy with different kinds of search, such as genetic algorithms, simulated annealing, and taboo search.

Acknowledgments Supported by the Agreement Technologies CONSOLIDER Project, Contract CSD2007-0022 and INGENIO 2010 and CHIST-ERA Project ACE and the Spanish Ministry of Education and Science TIN2010-16306 Project CBIT and EU Project 318770 PRAISE and Project PACES; EPSRC EP/J012149/1.

References

1. An, B., Gatti, N., & Lesser, V. (2009). Extending alternating-offers bargaining in one-to-many and many-to-many settings. In *Proceedings of the 2009 IEEE/WIC/ACM International Joint Conference on Web Intelligence and Intelligent Agent Technology, WI-IAT '09* (Vol. 02, pp. 423–426). Washington, DC: IEEE Computer Society. doi:10.1109/WI-IAT.2009.188.
2. An, B., Gatti, N., & Lesser, V. (2013). Bilateral bargaining with one-sided uncertain reserve prices. *Autonomous Agents and Multi-Agent Systems*, *26*(3), 420–455. doi:10.1007/s10458-012-9198-5.
3. An, B., Sim, K. M., Tang, L., Li, S., & Cheng, D. (2006). Continuous time negotiation mechanism for software agents. *IEEE Transactions on Systems, Man and Cybernetics, Part B: Cybernetics*, *36*(6), 1261–1272.
4. Applegate, D., Bixby, R. E., Chvátal, V., & Cook, W. J. (2012). http://www.tsp.gatech.edu/concorde. Accessed 18 Aug 2014.
5. Arcos, J. L., Esteva, M., Noriega, P., Rodríguez-Aguilar, J. A., & Sierra, C. (2005). Engineering open environments with electronic institutions. *Engineering Applications of Artificial Intelligence*, *18*(2), 191–204.
6. Baarslag, T., Hindriks, K., Jonker, C. M., Kraus, S., & Lin, R. (2010). The first automated negotiating agents competition (ANAC 2010). In T. Ito, M. Zhang, V. Robu, S. Fatima, & T. Matsuo (Eds.), *New trends in agent-based complex automated negotiations, series of studies in computational intelligence*. Berlin: Springer-Verlag.
7. Bektas, T. (2006). The multiple traveling salesman problem: An overview of formulations and solution procedures. *Omega*, *34*(3), 209–219.
8. DAIDE. (2013). Diplomacy ai development environment. http://www.daide.org.uk. Accessed 18 Aug 2014.
9. Endriss, U. (2006). Monotonic concession protocols for multilateral negotiation. In *Proceedings of the Fifth International Joint Conference on Autonomous Agents and Multiagent Systems, AAMAS '06* (pp. 392–399). New York, NY: ACM. doi:10.1145/1160633.1160702.
10. Fabregues, A., & Sierra, C. (2010). An agent architecture for simultaneous bilateral negotiations. In *Proceedings of the 13è Congrés Internacional de l'Associació Catalana d'Intel·ligència Artificial (CCIA 2010)* (pp. 29–38). Tarragona: Espluga de Francolí.
11. Fabregues, A., & Sierra, C. (2011). Dipgame: A challenging negotiation testbed. *Engineering Applications of Artificial Intelligence*, *24*, 1137–1146.
12. Faratin, P., Sierra, C., & Jennings, N. R. (1998). Negotiation decision functions for autonomous agents. *Robotics and Autonomous Systems*, *24*(3–4), 159–182. doi:10.1016/S0921-8890(98)00029-3, http://www.sciencedirect.com/science/article/pii/S0921889098000293. Multi-Agent Rationality.
13. Faratin, P., Sierra, C., & Jennings, N. R. (2000). Using similarity criteria to make negotiation trade-offs. In *Proceedings of the 4th International Conference on Multi-Agent Systems (ICMAS 2000)*, Boston (pp. 119–126).
14. Fatima, S., Wooldridge, M., & Jennings, N. R. (2009). An analysis of feasible solutions for multi-issue negotiation involving nonlinear utility functions. In *Proceedings of The 8th International Conference on Autonomous Agents and Multiagent Systems, AAMAS '09* (Vol. 2, pp. 1041–1048). Richland, SC: International Foundation for Autonomous Agents and Multiagent Systems. http://dl.acm.org/citation.cfm?id=1558109.1558158. Accessed 18 Aug 2014.
15. Gatti, N., Giunta, F. D., & Marino, S. (2008). Alternating-offers bargaining with one-sided uncertain deadlines: An efficient algorithm. *Artificial Intelligence*, *172*(8–9), 1119–1157. doi:10.1016/j.artint.2007.11.007, http://www.sciencedirect.com/science/article/pii/S0004370207001981.
16. Gendron, B., & Crainic, T. G. (1994). Parallel branch-and-bound algorithms: Survey and synthesis. *Operations Research*, *42*, 1042–1066.

17. Hemaissia, M., El Fallah Seghrouchni, A., Labreuche, C., & Mattioli, J. (2007). A multilateral multi-issue negotiation protocol. In *Proceedings of the 6th International Joint Conference on Autonomous Agents and Multiagent Systems, AAMAS '07* (pp. 155:1–155:8). New York, NY: ACM. doi:10.1145/1329125.1329314.
18. Hindriks, K., & Tykhonov, D. (2008). Opponent modelling in automated multi-issue negotiation using bayesian learning. In *Proceedings of the 7th International Joint Conference on Autonomous Agents and Multiagent Systems, AAMAS '08* (Vol. 1, pp. 331–338). Richland, SC: International Foundation for Autonomous Agents and Multiagent Systems. http://dl.acm.org/citation.cfm?id=1402383.1402433. Accessed 18 Aug 2014.
19. Ito, T., Klein, M., & Hattori, H. (2008). A multi-issue negotiation protocol among agents with nonlinear utility functions. *Multiagent Grid System*, *4*, 67–83. http://dl.acm.org/citation.cfm?id=1378675.1378678.
20. Jade. (2012). Java agent development framework. http://jade.tilab.com. Accessed 18 Aug 2014.
21. Klein, M., Faratin, P., Sayama, H., & Bar-Yam, Y. (2003). Protocols for negotiating complex contracts. *IEEE Intelligent Systems*, *18*(6), 32–38. doi:10.1109/MIS.2003.1249167.
22. Koenig, S., Tovey, C., Lagoudakis, M., Markakis, V., Kempe, D., Keskinocak, P., Kleywegt, A., Meyerson, A., & Jain, S. (2006). The power of sequential single-item auctions for agent coordination. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)* (pp. 1625–1629).
23. Krishna, V., & Serrano, R. (1996). Multilateral bargaining. *Review of Economic Studies*, *63*(1), 61–80. http://EconPapers.repec.org/RePEc:bla:restud:v:63:y:1996:i:1:p:61–80.
24. Lai, G., Sycara, K., & Li, C. (2008). A decentralized model for automated multi-attribute negotiations with incomplete information and general utility functions. *Multiagent Grid System*, *4*, 45–65. http://dl.acm.org/citation.cfm?id=1378675.1378677. Accessed 18 Aug 2014.
25. Lawler, E. L., & Wood, D. E. (1966). Branch-and-bound methods: A survey. *Operations Research*, *14*(4), 699–719.
26. Lin, R., & Kraus, S. (2010). Can automated agents proficiently negotiate with humans? *Communications of the ACM*, *53*, 78–88. doi:10.1145/1629175.1629199.
27. Marsa-Maestre, I., Lopez-Carmona, M. A., Velasco, J. R., & de la Hoz, E. (2009). Effective bidding and deal identification for negotiations in highly nonlinear scenarios. In *Proceedings of The 8th International Conference on Autonomous Agents and Multiagent Systems, AAMAS '09* (Vol. 2, pp. 1057–1064). Richland, SC: International Foundation for Autonomous Agents and Multiagent Systems. http://dl.acm.org/citation.cfm?id=1558109.1558160. Accessed 18 Aug 2014.
28. Marsa-Maestre, I., Lopez-Carmona, M. A., Velasco, J. R., Ito, T., Klein, M., & Fujita, K. (2009). Balancing utility and deal probability for auction-based negotiations in highly nonlinear utility spaces. In *Proceedings of the 21st International Jont Conference on Artifical Intelligence, IJCAI'09* (pp. 214–219). San Francisco, CA: Morgan Kaufmann Publishers Inc. http://dl.acm.org/citation.cfm?id=1661445.1661480.
29. MJC2: (2011). http://www.mjc2.com/transport_logistics_management.htm. Accessed 18 Aug 2014.
30. Modi, P. J., Shen, W. M., Tambe, M., & Yokoo, M. (2005). Adopt: Asynchronous distributed constraint optimization with quality guarantees. *Artificial Intelligence*, *161*(1–2), 149–180. doi:10.1016/j.artint.2004.09.003.
31. Nash, J. (1950). The bargaining problem. *Econometrica*, *18*, 155–162.
32. Nash, J. (1951). Non-cooperative games. *Annals of Mathematics*, *54*(2), pp. 286–295. http://www.jstor.org/stable/1969529. Accessed 18 Aug 2014.
33. Nguyen, T. D., & Jennings, N. R. (2004). Coordinating multiple concurrent negotiations. In *Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems, AAMAS '04* (Vol. 3, pp. 1064–1071). Washington, DC: IEEE Computer Society. doi:10.1109/AAMAS.2004.94.
34. Norman, D. (2012). http://www.ellought.demon.co.uk/dipai. Accessed 18 Aug 2014.
35. Ortner, J. M. (2012). A continuous time model of bilateral bargaining. http://people.bu.edu/jortner/index_files/CTBargaining.pdf.
36. Osborne, M., & Rubinstein, A. (1994). *A course in game theory*. Cambridge: MIT Press.
37. Papadimitriou, C. H. (1994). *Computational complexity*. Reading: Addison-Wesley.
38. Poundstone, W. (1993). *Prisoner's dilemma* (1st ed.). New York, NY: Doubleday.
39. Robu, V., Somefun, D. J. A., & Poutré, J. A. L. (2005). Modeling complex multi-issue negotiations using utility graphs. In *Proceedings of AAMAS'05* (pp. 280–287)
40. Rosenschein, J. S., & Zlotkin, G. (1994). *Rules of encounter*. Cambridge: MIT Press.
41. Rubinstein, A. (1982). Perfect equilibrium in a bargaining model. *Econometrica*, *50*(1), 97–109. http://ideas.repec.org/a/ecm/emetrp/v50y1982i1p97-109.html. Accessed 18 Aug 2014.
42. Serrano, R. (2008). Bargaining. In S. N. Durlauf & L. E. Blume (Eds.), *The new palgrave dictionary of economics*. Basingstoke: Palgrave Macmillan.
43. Sierra, C., & Debenham, J. (2007). The logic negotiation model. In *Sixth International Joint Conference on Autonomous Agents and Multiagent Systems, AAMAS '07* (pp. 1026–1033). New York: ACM.

44. Stahl, I. (1972). *Bargaining theory*. Stockholm: EFI.
45. Vieira Moura, A., & Augusto Scaraficci, R. (2010). A grasp strategy for a more constrained school timetabling problem. *International Journal of Operational Research*, *7*(2/2010), 152–170.
46. Willemen, R. (2002). *School timetable construction: Algorithms and complexity*. Eindhoven: T.U. Eindhoven. http://library.tue.nl/csp/dare/LinkToRepository.csp?recordnumber=553569.
47. Williams, C. R., Robu, V., Gerding, E. H., & Jennings, N. R. (2011). Using gaussian processes to optimise concession in complex negotiations against unknown opponents. In *Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence, IJCAI'11* (Vol. 1, pp. 432–438). AAAI Press. doi:10.5591/978-1-57735-516-8/IJCAI11-080.
48. Yokoo, M., Durfee, E. H., Ishida, T., & Kuwabara, K. (1998). The distributed constraint satisfaction problem: Formalization and algorithms. *IEEE Transactions on Knowledge and Data Engineering*, *10*, 673–685.